

ApossC SDK - V01.15

Software Reference

Table Of Contents

1 About	1
2 ChangeLog	2
3 HowToUse	3
4 File Index	4
4.1 File List	4
5 File Documentation	5
5.1 SDK_Amplifier_DS402_StateMachine.mc File Reference	5
5.2 SDK_Amplifier_Epos4.mc File Reference	35
5.3 SDK_Amplifier_MACS.mc File Reference	56
5.4 SDK_Amplifier_MiniMACS6_DS402_Slave.mc File Reference	68
5.5 SDK_Amplifier_MotorAlignment.mc File Reference	78
5.6 SDK_Amplifier_MotorCommissioning.mc File Reference	83
5.7 SDK_ApossC.mc File Reference	89
5.8 SDK_Axis_Setup.mc File Reference	90
5.9 SDK_Bussystem_EtherCat.mc File Reference	98
5.10 SDK_Communication_Ethernet.mc File Reference	102
5.11 SDK_Encoder_Setup.mc File Reference	105
5.12 SDK_Error_Description.mc File Reference	113
5.13 SDK_Information_General.mc File Reference	120
5.14 SDK_Kinematics_Setup.mc File Reference	123
5.15 SDK_Miscellaneous_IO.mc File Reference	134
5.16 SDK_Miscellaneous_Recording.mc File Reference	138
5.17 SDK_Motion_Movement.mc File Reference	141
5.18 SDK_SignalGenerator.mc File Reference	143
5.19 SDK_VirtualModule_AxisSetup.mc File Reference	149
5.20 SDK_VirtualModule_MasterSetup.mc File Reference	151
5.21 zub_About.txt File Reference	157
5.22 zub_ChangeLog.txt File Reference	157
5.23 zub_HowToUse.txt File Reference	157

6 Example Documentation

158

6.1 CAN_1Ax_DS402_ProfilePositionMode-ppm-Test.mc	158
6.2 CAN_1Ax_DS402_ProfileVelocityMode-pvm-Test.mc	159
6.3 CAN_1Ax_EPOS4-Test_csp.mc	161
6.4 CAN_1Ax_EPOS4-Test_cst.mc	164
6.5 CAN_1Ax_EPOS4-Test_csv.mc	167
6.6 CAN_2Ax_EPOS4-Test_csp.mc	170
6.7 CAN_4Ax_MiniMACS6_csp.mc	173
6.8 CAN_4Ax_MiniMACS6_cst.mc	176
6.9 CAN_4Ax_MiniMACS6_csv.mc	180
6.10 CAN_MultipleAx_DS402_ProfilePositionMode-ppm-Test.mc	184
6.11 CAN_MultipleAx_DS402_ProfilePositionMode-pvm-Test.mc	186
6.12 DeltaRobot_No_SM_EPOS4_ECAT.mc	186
6.13 DeltaRobot_SM.mc	190
6.14 ECAT_1Ax_EPOS4-Test_cst.mc	191
6.15 ECAT_3Ax_Epos4-Test_csp.mc	193
6.16 EthernetSocket_OpenClient.mc	196
6.17 EthernetSocket_OpenServer.mc	198
6.18 EthernetUDPSocket_OpenClient.mc	199
6.19 EthernetUDPSocket_OpenServer.mc	200
6.20 HowToUse_entire_SDK.mc	201
6.21 HowToUse_individual_SDK.mc	201
6.22 InformationGeneral.mc	202
6.23 Maxon_EC45_flat_1ax_BC_Enc.mc	202
6.24 Maxon_EC45_flat_1ax_BC_Hall.mc	204
6.25 Maxon_EC45_flat_1ax_SC_Hall_Inc.mc	206
6.26 Maxon_ECi30_2ax_SC_Hall_Inc.mc	208
6.27 Maxon_ECi40_1ax_SC_OL_Inc.mc	211
6.28 Maxon_ECi40_3ax_SC_OL_Inc.mc	213
6.29 Maxon_ECi52_1ax_SC_SSI.mc	215
6.30 Maxon_RE_40_1ax_Inc.mc	218
6.31 Maxon_RE_40_1ax_OL.mc	220
6.32 MotorCommissioning_MaxonECi30.mc	222
6.33 PWM_1Ax_ESCON.mc	225
6.34 ScaraRobot_SM.mc	226

Table of Contents

6.35 SDK_Amplifier_DS402_StateMachine.mc

227

6.36 SetupAbsSSIEncoder.mc

227

6.37 SetupIncEncoder.mc

227

6.38 SetupSignalGeneratorAxis.mc

228

6.39 SetupSignalGeneratorOpenloop.mc

229

6.40 SetupSinCosEncoder.mc

230

6.41 Stepper_XY_CL.mc

230

6.42 Stepper_XY_OL.mc

232

6.43 VirtualMaster_ProfileMode.mc

234

Description of the idea and purpose of the SDK. The SDK allows a quick start with ApossIDE and simplifies the use of the tool. The SDK is written in ApossC. To guarantee the functionality of the provided functions, it is recommended to use the latest version of ApossIDE. As minimum requirement an ApossIDE version greater than 7.00.00 is required. In case of questions or ambiguities the support of Zub machine control ag can be contacted.

Contact

zub machine control AG
Buzibachstrasse 31
6023 Rothenburg
Switzerland
Phone +41 41 54150-40
E-Mail info-zub@maxongroup.com

Listing of the changes and history of a development.

The changelog is currently not embedded in the PDF version of the ApossC Help and can be found in the main directory of the SDK folder

The following section demonstrates the first steps with the ApossC SDK. Simplified examples of how the SDK can be used are shown here using sample programs.

- [Entire SDK](#)
- [Individual adoption of the SDK](#)

A detailed description of how to use the SDK can be found in the document [ApossIDE_ApossCSDK_↔HowToUse.pdf](#).

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

SDK_Amplifier_DS402_StateMachine.mc	5
This file provides all the function used for DS402 PPM and PVM	
SDK_Amplifier_Epos4.mc	35
This file provides all the function used to setup a Epos4 drive	
SDK_Amplifier_MACS.mc	56
Functions for the integrated amplifier setup	
SDK_Amplifier_MiniMACS6_DS402_Slave.mc	68
This file provides all the function used to setup a MiniMACS6 DS402 slave	
SDK_Amplifier_MotorAlignment.mc	78
Functions for aligning a motor	
SDK_Amplifier_MotorCommissioning.mc	83
Functions for commissioning a motor	
SDK_ApossC.mc	89
File to include the entire SDK	
SDK_Axis_Setup.mc	90
Functions for the axis setup	
SDK_Bussystem_EtherCat.mc	98
Functions to work with an EtherCat Master/ slave	
SDK_Communication_Ethernet.mc	102
Functions to work with an Ethernet as communication	
SDK_Encoder_Setup.mc	105
Functions for the general amplifier setup	
SDK_Error_Description.mc	113
SDK_Information_General.mc	120
Functions to get general informations	
SDK_Kinematics_Setup.mc	123
Functions for the kinematics setup	
SDK_Miscellaneous_IO.mc	134
Functions with IO samples	

File Documentation

SDK_Miscellaneous_Recording.mc	
Functions with recording samples	138
SDK_Motion_Movement.mc	
Functions for the simplification of movements	141
SDK_SignalGenerator.mc	
Functions for the operation of the signal generator	143
SDK_VirtualModule_AxisSetup.mc	
Functions to work with virtual axis modules	149
SDK_VirtualModule_MasterSetup.mc	
Functions to work with a virtual master	151

5 File Documentation

5.1 SDK_Amplifier_DS402_StateMachine.mc File Reference

This file provides all the function used for DS402 PPM and PVM.

```
#include <SysDef.mh>
#include "SDK_Amplifier_DS402_StateMachine.mh"
```

Macros

- #define [DS402_PPM_IMMEDIATELY](#) 1

Functions

- long [sdkDS402_ReadStatusWord](#) (long busId)
DS 402 State-Machine functions.
- long [sdkDS402_ReadControlWord](#) (long busId)
Read control word.
- void [sdkDS402_WriteControlWord](#) (long busId, long value)
Write control word.
- long [sdkDS402_GetActDriveState](#) (long busId)

Get the actual drive state.

- void [sdkDS402_PrintState](#) (long state)

Print the name of certain drive state.

- long [helperDS402_ExecuteTransition](#) (long busId, long transition)

Execute a state transition.

- long [sdkDS402_TransitionToState](#) (long busId, long newState)

Transition to another state.

- long [sdkDS402_WaitTransitionToState](#) (long busId, long targetState, long timeout)

Wait transition to another state.

- void [sdkDS402_SetProfileAcceleration](#) (long busId, long acc)

Profile Mode settings.

- void [sdkDS402_SetProfileDeceleration](#) (long busId, long dec)

Set the profile deceleration DS402_PROFILE_DECELERATION(0x6084)

- void [sdkDS402_SetProfileVelocity](#) (long busId, long vel)

Set the profile velocity DS402_PROFILE_VELOCITY(0x6081)

- void [sdkDS402_SetMotionProfileType](#) (long busId, long type)

Set the motion profile type DS402_MOTION_PROFILE_TYPE(0x6086)

- void [sdkDS402_SetMaxProfileVelocity](#) (long busId, long vel)

Set the max profile velocity DS402_MAX_PROFILE_VELOCITY(0x607F)

- void [sdkDS402_SetMaxMotorSpeed](#) (long busId, long vel)

Set the max profile velocity DS402_MAX_MOTOR_SPEED(0x6080)

- void [sdkDS402_SetMaxAcceleration](#) (long busId, long acc)

Set the max acceleration DS402_MAX_ACCELERATION(0x60C5)

- void [sdkDS402_SetQuickStopDeceleration](#) (long busId, long dec)

Set quick stop deceleration DS402_QUICK_STOP_DECELERATION(0x6085)

- void [sdkDS402_SetProfileMovementParameter](#) (long busId, long acc, long dec, long velocity)

Set quick stop deceleration DS402_QUICK_STOP_DECELERATION(0x6085)

- void [sdkDS402_SetOpationMode](#) (long busId, long operationMode)

Profile Mode functions.

- long [sdkDS402_GetOpationMode](#) (long busId)
Get the mode of operation DS402_MODES_OF_OPERATION(0x6060)
- void [sdkDS402_QuickStop](#) (long busId)
Execute quick stop.
- void [sdkDS402_Halt](#) (long busId)
Stop the motor.
- void [sdkDS402_ResetFault](#) (long busId)
Reset fault.
- void [sdkDS402_Print_StatusWord](#) (long busId)
Print the status word.
- void [sdkDS402_Print_ControlWord](#) (long busId)
Print the control word.
- long [helperDS402_PPM_PosStart](#) (long busId, long pos, long relative, long startImmediately)
Profile Position Mode functions.
- long [sdkDS402_PPM_PosAbsStart](#) (long busId, long pos, long startImmediately)
Start absolute positioning.
- long [sdkDS402_PPM_PosRelStart](#) (long busId, long pos, long startImmediately)
Start relative positioning.
- long [sdkDS402_PPM_TargetReached](#) (long busId)
Checks whether the target was reached.
- long [sdkDS402_PPM_WaitTargetReached](#) (long busId, long timeout)
Wait until target is reachd.
- void [sdkDS402_PVM_SetTargetVelocity](#) (long busId, long targetVelocity)
Profile Velocity Mode functions.
- long [sdkDS402_PVM_CvelStart](#) (long busId)
Start continuous velocity (PVM)
- long [sdkDS402_PVM_CvelStop](#) (long busId)
Stop continuous velocity (PVM)
- long [sdkDS402_PVM_TargetReached](#) (long busId)
Checks whether the target velocity has reached.
- long [sdkDS402_PVM_WaitTargetReached](#) (long busId, long timeout)
Wait until target is reachd.

5.1.1 Detailed Description

This file provides all the function used for DS402 PPM and PVM.

A DS402 slave can be operated in the different modes PPM and PVM The amplifiers can be controlled via EtherCAT or CAN bus.

5.1.2 Macro Definition Documentation

5.1.2.1 DS402_PPM_IMMEDIATELY `#define DS402_PPM_IMMEDIATELY 1`

Examples

[CAN_1Ax_DS402_ProfilePositionMode-ppm-Test.mc](#), and [CAN_MultipleAx_DS402_ProfilePositionMode-p](#)

5.1.3 Function Documentation

5.1.3.1 `helperDS402_ExecuteTransition()` `long helperDS402_ExecuteTransition (` `long busId,` `long transition)`

Execute a state transition.

This function executes a state transition within the DS402 state machine. DO NOT USE this function for state changes. It is just a helper function for the function [sdkDS402_TransitionToState\(\)](#). If you want to execute a state transition, use the function [sdkDS402_TransitionToState\(\)](#).

The function writes value to the control word, to set the bits for state changes.

Bit 0: Switched on
 Bit 1: Enable voltage
 Bit 2: Quick stop
 Bit 3: Enable operation
 ..
 Bit 7: Fault reset
 T

[sdkDS402_TransitionToState\(\)](#)

File Documentation

Parameters

<i>busId</i>	Bus ID of the connected slave
<i>transition</i>	Value of the transition to execute

Returns

value: Process value
value > 0 Transition successful
value < 0 Invalid transition

Definition at line 152 of file SDK_Amplifier_DS402_StateMachine.mc.

```

153 {
154     long controlWord = sdkDS402_ReadControlWord(busId);
155     long retVal = 1;
156
157     switch(transition) {
158         case DS402_TRANSITION_SHUTDOWN:
159             //Transition Shutdown: CW 0xxx x110
160             printf("...: execute Transition: DS402_TRANSITION_SHUTDOWN\n", busId);
161             controlWord = (controlWord & 0xFF78) | 0x06;
162             sdkDS402_WriteControlWord(busId, controlWord);
163             break;
164         case DS402_TRANSITION_SWITCH_ON:
165             printf("...: execute Transition: DS402_TRANSITION_SWITCH_ON\n", busId);
166             //Transition Switch On: CW 0xxx x111
167             controlWord = (controlWord & 0xFF78) | 0x07;
168             sdkDS402_WriteControlWord(busId, controlWord);
169             break;
170         case DS402_TRANSITION_ENABLE_OPERATION:
171             printf("...: execute Transition: DS402_TRANSITION_ENABLE_OPERATION\n", busId);
172             //Transition Enable Operation: CW 0xxx 1111
173             controlWord = (controlWord & 0xFF70) | 0x0F;
174             sdkDS402_WriteControlWord(busId, controlWord);
175             break;
176         case DS402_TRANSITION_DISABLE_VOLTAGE:
177             printf("...: execute Transition: DS402_TRANSITION_DISABLE_VOLTAGE\n", busId);
178             //Transition Disable Voltage: CW 0xxx xx0x
179             controlWord = (controlWord & 0xFF7D);
180             sdkDS402_WriteControlWord(busId, controlWord);
181             break;
182         case DS402_TRANSITION_QUICK_STOP:
183             printf("...: execute Transition: DS402_TRANSITION_QUICK_STOP\n", busId);
184             //Transition Quick Stop: CW 0xxx x01x
185             controlWord = (controlWord & 0xFF79) | 0x02;
186             sdkDS402_WriteControlWord(busId, controlWord);
187             break;
188         case DS402_TRANSITION_DISABLE_OPERATION:
189             printf("...: execute Transition: DS402_TRANSITION_DISABLE_OPERATION\n", busId);
190             //Transition Disable Voltage: CW 0xxx 0111
191             controlWord = (controlWord & 0xFF70) | 0x07;
192             sdkDS402_WriteControlWord(busId, controlWord);
193             break;
194         case DS402_TRANSITION_FAULT_RESET:
195             printf("...: execute Transition: DS402_TRANSITION_FAULT_RESET\n", busId);
196             //Transition Fault Reset: CW 0xxx xxxx -> 1xxx xxxx
197             controlWord = (controlWord & 0xFF7F);
198             sdkDS402_WriteControlWord(busId, controlWord);
199             //Delay(1);
200             controlWord = controlWord | 0x80;
201             sdkDS402_WriteControlWord(busId, controlWord);
202             //Transition switch on disable
203             controlWord = controlWord & 0xFF7F;

```

```
204         sdkDS402_WriteControlWord(busId, controlWord);
205         break;
206     default:
207         printf("Invalid state transition\n");
208         retVal = -1;
209         break;
210     }
211
212     return(retVal);
213 }
```

5.1.3.2 helperDS402_PPM_PosStart() long helperDS402_PPM_PosStart (

long busId,

long pos,

long relative,

long startImmediately)

Profile Position Mode functions.

Start positioning relative or absolute

This function starts positioning of an axis relative or absolute. Do not use this funtion. Use [sdkDS402_PPM_PosAbsStart\(\)](#) or [sdkDS402_PPM_PosRelStart\(\)](#) instead. This funtion is blocking until the setpoint acknowledge is received (max 2ms)

Parameters

<i>busId</i>	Bus ID of the connected slave
<i>pos</i>	target position
<i>relative</i>	excute the positioning relative or absolute 0: absolute 1: relative
<i>startImmediately</i>	start the movement immediately 0: Finish actual positioning 1: Abort actual positioning and start next positioning

Returns

- value: Process value
- value = 1 successful, new setpoint acknowlegded
- value = -1 failed, new setpoint not acknowlegded
- value = -2 failed, previous setpoint has been assumed and no additional setpoint may be accepted

Definition at line 721 of file SDK_Amplifier_DS402_StateMachine.mc.

```
722 {
```

File Documentation

```

723     long retVal = 1;
724     long controlWord = 0;
725     long setpointAcknowledge=1;
726     long time = 0;
727     long actState = -1;
728
729     printf("...: new setpoint: %d\n", pos);
730
731     if(sdkDS402_ReadStatusWord(busId).i[DS402_SW_PPM_BIT_SETPOINT_ACK]==1)
732     {
733         print("...: the previous setpoint has been assumed and no additional setpoint may be
accepted");
734         return(-2);
735     }
736
737     actState = sdkDS402_GetActDriveState(busId);
738     if(actState != DS402_DRIVE_STATE_OPERATION_ENABLED)
739     {
740         print("...: drive state is not in operation enabled");
741         return(-1);
742     }
743
744     SdoWrite(busId, DS402_TARGET_POSITION, 0 ,pos);
745
746     controlWord = sdkDS402_ReadControlWord(busId);
747     controlWord.i[DS402_CW_PPM_BIT_NEW_SETPOINT] = 1;
748     controlWord.i[DS402_CW_PPM_BIT_CHANGE_IMMEDIATELY] = startImmediately;
749     controlWord.i[DS402_CW_PPM_BIT_RELATIVE_MOVEMENT] = relative;
750     controlWord.i[DS402_CW_PPM_BIT_ENDLESS_MOVEMENT] = 0;
751
752     sdkDS402_WriteControlWord(busId, controlWord);
753
754     time = Time();
755     while(sdkDS402_ReadStatusWord(busId).i[DS402_SW_PPM_BIT_SETPOINT_ACK]!=1)
756     {
757         if((Time()-time) > 2) //is blocking until the new setpoint has been acknowledged
758         {
759             print("...: no setpoint acknowledge received");
760             setpointAcknowledge = 0;
761             retVal = -1;
762             break;
763         }
764     }
765     controlWord = sdkDS402_ReadControlWord(busId);
766     controlWord.i[DS402_CW_PPM_BIT_NEW_SETPOINT] = 0;
767     controlWord.i[DS402_CW_PPM_BIT_CHANGE_IMMEDIATELY] = 0;
768     controlWord.i[DS402_CW_PPM_BIT_RELATIVE_MOVEMENT] = 0;
769     controlWord.i[DS402_CW_PPM_BIT_ENDLESS_MOVEMENT] = 0;
770
771     sdkDS402_WriteControlWord(busId, controlWord);
772
773     if(setpointAcknowledge)
774     {
775         print("...: setpoint acknowleged");
776     }
777
778     return(retVal);
779 }

```

5.1.3.3 sdkDS402_GetActDriveState() long sdkDS402_GetActDriveState (long busId)

Get the actual drive state.

This function gets the actual state in DS402 state machine. The state is depending on the following bits in the status word.

Bit0: Ready to switch on
Bit1: Switched on
Bit2: Operation enabled
Bit3: Fault
Bit4: *(don't care)
Bit5: Quick stop (low active)
Bit6: switch on disabled

Parameters

<i>busId</i>	Bus ID of the connected slave
--------------	-------------------------------

Returns

value: drive state (value mapping: in SDK_Amplifier_DS402_StateMachine.mh)

Examples

[CAN_1Ax_DS402_ProfilePositionMode-ppm-Test.mc](#), [CAN_1Ax_DS402_ProfileVelocityMode-pvm-Test.mc](#)
and [CAN_MultipleAx_DS402_ProfilePositionMode-ppm-Test.mc](#).

Definition at line 83 of file SDK_Amplifier_DS402_StateMachine.mc.

```
84 {  
85     long status = sdkDS402_ReadStatusWord(busId);  
86  
87     switch(status & 0x006F) //mask bits xxxx xxxx x11x 1111  
88     {  
89  
90         case 0x0000:         return(DS402_DRIVE_STATE_NOT_READY_TO_SWITCH_ON);      //xxxx xxxx x00x 0000  
91         case 0x0040:         return(DS402_DRIVE_STATE_SWITCH_ON_DISABLED);          //xxxx xxxx x10x 0000  
92         case 0x0021:         return(DS402_DRIVE_STATE_READY_TO_SWITCH_ON);          //xxxx xxxx x01x 0001  
93         case 0x0023:         return(DS402_DRIVE_STATE_SWITCHED_ON);                 //xxxx xxxx x01x 0011  
94         case 0x0027:         return(DS402_DRIVE_STATE_OPERATION_ENABLED);           //xxxx xxxx x01x 0111  
95         case 0x0007:         return(DS402_DRIVE_STATE_QUICK_STOP_ACTIVE);           //xxxx xxxx x01x 0001  
96         case 0x000F:         return(DS402_DRIVE_STATE_FAULT_REACTION_ACTIVE);        //xxxx xxxx x00x 1111  
97         case 0x0008:         return(DS402_DRIVE_STATE_FAULT);                       //xxxx xxxx x00x 1000  
98         default:             return(-1);  
99     }  
100 }
```

5.1.3.4 sdkDS402_GetOperationMode() long sdkDS402_GetOperationMode (
long busId)

Get the mode of operation DS402_MODES_OF_OPERATION(0x6060)

Get the mode of operation.

<i>bus</i> ↔ <i>Id</i>	Bus ID of the connected slave
---------------------------	-------------------------------

Returns

operationMode Definition of the operation mode
DS402_OP_PPM (1): Profile Position Mode (PPM)
DS402_OP_PVM (3): Profile Velocity Mode (PVM)

Definition at line 531 of file SDK_Amplifier_DS402_StateMachine.mc.

```
532 {  
533     long operationMode = SdoRead( busId, DS402_MODES_OF_OPERATION, 0);  
534     return(operationMode);  
535 }
```

5.1.3.5 **sdkDS402_Halt()** void sdkDS402_Halt (
 long busId)

Stop the motor.

This function stops the motor with the defined profile deceleration. The deceleration can be set with method `sdkDS402_SetProfileDeceleration`

Parameters

<i>bus</i> ↔ <i>Id</i>	Bus ID of the connected slave
---------------------------	-------------------------------

Examples

[CAN_1Ax_DS402_ProfileVelocityMode-pvm-Test.mc.](#)

Definition at line 557 of file SDK_Amplifier_DS402_StateMachine.mc.

```
558 {  
559     long controlWord;  
560     printf("Bus Id %d, Halt\n", busId);  
561  
562     controlWord = sdkDS402_ReadControlWord(busId);  
563     controlWord.i[DS402_CW_BIT_HALT] = 1;  
564  
565     sdkDS402_WriteControlWord(busId, controlWord);  
566 }
```

5.1.3.6 sdkDS402_PPM_PosAbsStart()

long sdkDS402_PPM_PosAbsStart (

long *busId*,

long *pos*,

long *startImmediately*)

Start absolute positioning.

This function starts absolute positioning This funtion is blocking until the setpoint acknowledge is received (max 2ms)

Parameters

<i>busId</i>	Bus ID of the connected slave
<i>pos</i>	target position
<i>startImmediately</i>	start the movement immediately <div> <div>0: Finish actual positioning</div> <div>1: Abort actual positioning and start next positioning</div> </div>

Returns

- value: Process value
- value = 1 successful, new setpoint acknowlegded
- value = -1 failed, new setpoint not acknowlegded
- value = -2 failed, previous setpoint has been assumed and no additional setpoint may be accepted

Examples

[CAN_1Ax_DS402_ProfilePositionMode-ppm-Test.mc](#), and [CAN_MultipleAx_DS402_ProfilePositionMode-p](#)

Definition at line 797 of file SDK_Amplifier_DS402_StateMachine.mc.

798 {

799

800 printf("Bus Id %d, sdkDS402_PPM_PosAbsStart:\n", busId);

801 return helperDS402_PPM_PosStart(busId, pos, DS402_PPM_ABSOLUTE, startImmediately);

802 }

5.1.3.7 sdkDS402_PPM_PosRelStart()

long sdkDS402_PPM_PosRelStart (

long *busId*,

long *pos*,

long *startImmediately*)

Start relative positioning.

This function starts relative positioning This funtion is blocking until the setpoint acknowledge is received (max 2ms)

<i>busId</i>	Bus ID of the connected slave
<i>pos</i>	relative movement value
<i>startImmediately</i>	start the movement immediately 0 : Finish actual positioning 1 : Abort actual positioning and start next positioning

Returns

value: Process value
value = 1 successful, new setpoint acknowledged
value = -1 failed, new setpoint not acknowledged
value = -2 failed, previous setpoint has been assumed and no additional setpoint may be accepted

Examples

[CAN_1Ax_DS402_ProfilePositionMode-ppm-Test.mc](#), and [CAN_MultipleAx_DS402_ProfilePositionMode-ppm-Test.mc](#)

Definition at line 819 of file SDK_Amplifier_DS402_StateMachine.mc.

```
820 {  
821     printf("Bus Id %d, sdkDS402_PPM_PosRelStart:\n", busId);  
822     return helperDS402_PPM_PosStart(busId, pos, DS402_PPM_RELATIVE, startImmediately);  
823 }
```

5.1.3.8 sdkDS402_PPM_TargetReached() long sdkDS402_PPM_TargetReached (long busId)

Checks whether the target was reached.

This function verifies whether the axis reached the target position This function is not blocking until the axis reaches the target.

Parameters

<i>busId</i>	Bus ID of the connected slave
--------------	-------------------------------

Returns

value: Process value
value = 1 Target reached
value = 0 Target NOT reached

[CAN_MultipleAx_DS402_ProfilePositionMode-ppm-Test.mc.](#)

Definition at line 836 of file SDK_Amplifier_DS402_StateMachine.mc.

```
837 {
838     return (sdkDS402_ReadStatusWord(busId) .i[DS402_SW_BIT_TARGET_REACHED]);
839 }
```

5.1.3.9 sdkDS402_PPM_WaitTargetReached() long sdkDS402_PPM_WaitTargetReached (
 long *busId*,
 long *timeout*)

Wait until target is reachd.

This function waits (blocks) until the axis has reached the target position whthin the defined timeout.

Parameters

<i>busId</i>	Bus ID of the connected slave
<i>timeout</i>	Timeout until the function is aborted >=0: timeout in [ms] < 0: infinite timeout

Returns

value: Process value
value > 0 Target reached
value < 0 Timeout expired

Examples

[CAN_1Ax_DS402_ProfilePositionMode-ppm-Test.mc.](#)

Definition at line 853 of file SDK_Amplifier_DS402_StateMachine.mc.

```
854 {
855     long time = Time();
856     long targetReached = 1;
857
858     printf("Bus Id %d, sdkDS402_PPM_WaitTargetReached:\n", busId);
859
860     while (sdkDS402_PPM_TargetReached(busId) == 0 && targetReached)
861     {
862         if (timeout >= 0 && (Time() - time) > timeout)
863         {
864             targetReached = -1;
865             printf("...: target position not reached within timeout (%d ms)\n", timeout);
866         }
867     }
868     if (targetReached)
869     {
```

File Documentation

```

870         print("...:  target position reached");
871     }
872
873     return(targetReached);
874 }
```

5.1.3.10 sdkDS402_Print_ControlWord() void sdkDS402_Print_ControlWord (
 long busId)

Print the control word.

This function writes the control word split into individual bits with their description. It is depending on the actual modes of operation (PPM/PVM). This function can be used for debugging purposes.

Parameters

<i>busId</i>	Bus ID of the connected slave
--------------	-------------------------------

Definition at line 656 of file SDK_Amplifier_DS402_StateMachine.mc.

```

657 {
658     long value;
659     long controlWord;
660     long operationMode;
661
662     operationMode = SdoRead(busId, DS402_MODES_OF_OPERATION, 0);
663     controlWord = sdkDS402_ReadControlWord(busId);
664
665     print("CONTROL WORD");
666     value = controlWord.i[DS402_CW_BIT_SWITCH_ON];
667     printf("\tSWITCH_ON:\t\t%d\n", value);
668
669     value = controlWord.i[DS402_CW_BIT_ENABLE_VOLTAGE];
670     printf("\tENABLE_VOLTAGE:\t\t%d\n", value);
671
672     value = controlWord.i[DS402_CW_BIT_QUICK_STOP];
673     printf("\tQUICK_STOP:\t\t%d\n", value);
674
675     value = controlWord.i[DS402_CW_BIT_ENABLE_OPERATION];
676     printf("\tENABLE_OPERATION:\t\t%d\n", value);
677
678     value = controlWord.i[DS402_CW_BIT_HALT];
679     printf("\tHALT:\t\t\t%d\n", value);
680
681     if(operationMode == DS402_OP_PPM)
682     {
683         value = controlWord.i[DS402_CW_PPM_BIT_NEW_SETPOINT];
684         printf("\tNEW_SETPOINT:\t\t%d\n", value);
685
686         value = controlWord.i[DS402_CW_PPM_BIT_CHANGE_IMMEDIATELY];
687         printf("\tCHANGE_IMMEDIATELY:\t\t%d\n", value);
688
689         value = controlWord.i[DS402_CW_PPM_BIT_RELATIVE_MOVEMENT];
690         printf("\tRELATIVE_MOVEMENT:\t\t%d\n", value);
691
692         value = controlWord.i[DS402_CW_PPM_BIT_ENDLESS_MOVEMENT];
693         printf("\tENDLESS_MOVEMENT:\t\t%d\n", value);
694     }
```

5.1.3.11 sdkDS402_Print_StatusWord()

void sdkDS402_Print_StatusWord (
long busId)

Print the status word.

This function print the status word split into individual bits with their description. It is depending on the actual modes of operation (PPM/PVM). This function can be used for debugging purposes.

Parameters

<i>busId</i>	Bus ID of the connected slave
--------------	-------------------------------

Examples

[CAN_1Ax_DS402_ProfilePositionMode-ppm-Test.mc](#), and [CAN_1Ax_DS402_ProfileVelocityMode-pvm-Test.mc](#)

Definition at line 588 of file SDK_Amplifier_DS402_StateMachine.mc.

```
589 {  
590     long value;  
591     long status;  
592     long operationMode;  
593     operationMode = SdoRead(busId, DS402_MODES_OF_OPERATION, 0);  
594     status = sdkDS402_ReadStatusWord(busId);  
595  
596  
597     print("STATUS WORD");  
598  
599     value = status.i[DS402_SW_BIT_READY_TO_SWITCH_ON];  
600     printf("\tREADY_TO_SWITCH_ON:\t%d\n", value);  
601  
602     value = status.i[DS402_SW_BIT_SWITCHED_ON];  
603     printf("\tSWITCHED_ON:\t\t%d\n", value);  
604  
605     value = status.i[DS402_SW_BIT_OPERATION_ENABLED];  
606     printf("\tOPERATION_ENABLED:\t%d\n", value);  
607  
608     value = status.i[DS402_SW_BIT_FAULT];  
609     printf("\tFAULT:\t\t\t%d\n", value);  
610  
611     value = status.i[DS402_SW_BIT_VOLTAGE_ENABLED];  
612     printf("\tVOLTAGE_ENABLED:\t%d\n", value);  
613  
614     value = status.i[DS402_SW_BIT_QUICK_STOP];  
615     printf("\tQUICK_STOP:\t\t%d\n", value);  
616  
617     value = status.i[DS402_SW_BIT_SWITCH_ON_DISABLED];  
618     printf("\tSWITCH_ON_DISABLED:\t%d\n", value);  
619  
620     value = status.i[DS402_SW_BIT_WARNING];  
621     printf("\tWARNING:\t\t\t%d\n", value);  
622  
623     value = status.i[DS402_SW_BIT_REMOTE];  
624     printf("\tREMOTE:\t\t\t\t%d\n", value);  
625
```

File Documentation

```
626     value = status.i[DS402_SW_BIT_TARGET_REACHED];
627     printf("\tTARGET_REACHED:\t\t%d\n", value);
628
629     value = status.i[DS402_SW_BIT_INTERNAL_LIMIT];
630     printf("\tINTERNAL_LIMIT:\t\t%d\n", value);
631
632     if(operationMode == DS402_OP_PPM)
633     {
634         value = status.i[DS402_SW_PPM_BIT_SETPPOINT_ACK];
635         printf("\tSETPPOINT_ACK:\t\t%d\n", value);
636
637         value = status.i[DS402_SW_PPM_BIT_FOLLLWING_ERROR];
638         printf("\tFOLLLWING_ERROR:\t\t%d\n", value);
639     }
640
641     if(operationMode == DS402_OP_PVM)
642     {
643         value = status.i[DS402_SW_PVM_BIT_SPEED];
644         printf("\tSPEED:\t\t\t%d\n", value);
645     }
646 }
```

5.1.3.12 sdkDS402_PrintState() void sdkDS402_PrintState (
long state)

Print the name of certain drive state.

This function prints the state name from a state vlaue. It can be used to create human readable print outputs for debugging reason.

Parameters

state	value of the state to print (value mapping: in SDK_Amplifier_DS402_StateMachine.mh)
-------	---

Definition at line 111 of file SDK_Amplifier_DS402_StateMachine.mc.

```
112 {
113     switch(state)
114     {
115         case DS402_DRIVE_STATE_NOT_READY_TO_SWITCH_ON: printf("DRIVE_STATE_NOT_READY_TO_SWITCH_ON");
break;
116         case DS402_DRIVE_STATE_SWITCH_ON_DISABLED:      printf("DRIVE_STATE_SWITCH_ON_DISABLED");
break;
117         case DS402_DRIVE_STATE_READY_TO_SWITCH_ON:      printf("DRIVE_STATE_READY_TO_SWITCH_ON");
break;
118         case DS402_DRIVE_STATE_SWITCHED_ON:             printf("DRIVE_STATE_SWITCHED_ON"); break;
119         case DS402_DRIVE_STATE_OPERATION_ENABLED:       printf("DRIVE_STATE_OPERATION_ENABLED");
break;
120         case DS402_DRIVE_STATE_QUICK_STOP_ACTIVE:       printf("DRIVE_STATE_QUICK_STOP_ACTIVE");
break;
121         case DS402_DRIVE_STATE_FAULT_REACTION_ACTIVE:   printf("DRIVE_STATE_FAULT_REACTION_ACTIVE");
break;
122         case DS402_DRIVE_STATE_FAULT:                   printf("DRIVE_STATE_FAULT"); break;
123         default:                                         printf("%d",
state);
124     }
125 }
```

Start continuous velocity (PVM)

Parameters

$bus \leftarrow Id$	Bus ID of the connected slave
---------------------	-------------------------------

Returns

```
value: Process value
value 1 Successful
value -1 Failed, drive is not in 'Operation enabled' state
```

Examples

CAN 1Ax DS402 ProfileVelocityMode-pvm-Test.mc.

Definition at line 904 of file SDK_Amplifier_DS402_StateMachine.mc.

```

905 {
906     long actState;
907     long controlWord;
908
909     printf("Bus Id %d, sdkDS402_PPM_CvelStart:  \n", busId);
910
911     actState = sdkDS402_GetActDriveState(busId);
912     if(actState != DS402_DRIVE_STATE_OPERATION_ENABLED)
913     {
914         printf("...:  drive state is not in operation enabled\n");
915         return(-1);
916     }
917
918     //Reset the Halt bit to start the execution
919     controlWord = sdkDS402_ReadControlWord(busId);
920     controlWord.i[DS402_CW_BIT_HALT] = 0;
921     sdkDS402_WriteControlWord(busId, controlWord);
922
923     printf("...:  started\n");
924
925     return(1);
926 }

```

Stop continuous velocity (PVM)

This function stops the axis.

<i>busId</i>	Bus ID of the connected slave
--------------	-------------------------------

Returns

value: Process value
value 1 Successful

Examples

[CAN_1Ax_DS402_ProfileVelocityMode-pvm-Test.mc.](#)

Definition at line 935 of file SDK_Amplifier_DS402_StateMachine.mc.

```

936 {
937     long controlWord;
938     printf("Bus Id %d, sdkDS402_PPM_CvelStop:\n", busId);
939
940     //Set the Halt bit, to stop the execution
941     controlWord = sdkDS402_ReadControlWord(busId);
942     controlWord.i[DS402_CW_BIT_HALT] = 1;
943
944     sdkDS402_WriteControlWord(busId, controlWord);
945     printf("...: stopped\n");
946
947     return(1);
948 }
```

5.1.3.15 sdkDS402_PVM_SetTargetVelocity() void sdkDS402_PVM_SetTargetVelocity (
long *busId*,
long *targetVelocity*)

Profile Velocity Mode functions.

Set target velocity for PVM

This function sets the target velocity for the Profile Position Mode (PVM)

Parameters

<i>busId</i>	Bus ID of the connected slave
<i>targetVelocity</i>	Target velocity

Examples

[CAN_1Ax_DS402_ProfileVelocityMode-pvm-Test.mc.](#)

Definition at line 888 of file SDK_Amplifier_DS402_StateMachine.mc.

```
889 {
890     printf("Bus Id %d, PVM: Set target velocity to %d\n", busId, targetVelocity);
891     SdoWrite(busId, DS402_TARGET_VELOCITY, 0, targetVelocity);
892 }
```

5.1.3.16 sdkDS402_PVM_TargetReached() long sdkDS402_PVM_TargetReached (
long busId)

Checks whether the target velocity has reached.

This function verifies whether the axis reached the target velocity

Parameters

busId	Bus ID of the connected slave
-------	-------------------------------

Returns

- value: Process value
- value = 1 Target reached
- value = 0 Target NOT reached

Definition at line 959 of file SDK_Amplifier_DS402_StateMachine.mc.

```
960 {
961     return sdkDS402_ReadStatusWord(busId) .i[DS402_SW_BIT_TARGET_REACHED];
962 }
```

5.1.3.17 sdkDS402_PVM_WaitTargetReached() long sdkDS402_PVM_WaitTargetReached (
long busId,
long timeout)

Wait until target is reachd.

This function waits (blocks) until the axis has reached the target velocity whthin the defined timeout.

Parameters

busId	Bus ID of the connected slave
timeout	Timeout until the function returns >=0: timeout in [ms] < 0: infinite timeout

value: Process value
value > 0 Target reached
value < 0 Timeout expired

Examples

[CAN_1Ax_DS402_ProfileVelocityMode-pvm-Test.mc.](#)

Definition at line 976 of file SDK_Amplifier_DS402_StateMachine.mc.

```
977 {
978     long time = Time();
979     long targetReached = 1;
980
981     printf("Bus Id %d, sdkDS402_PVM_WaitTargetReached:\n", busId);
982
983     while (sdkDS402_PPM_TargetReached(busId) == 0 && targetReached)
984     {
985         if (timeout >= 0 && (Time() - time) > timeout)
986         {
987             targetReached = -1;
988             printf("...: target velocity not reached within timeout (%d ms)\n", timeout);
989         }
990     }
991     if (targetReached)
992     {
993         print("...: target velocity reached");
994     }
995
996     return (targetReached);
997 }
```

5.1.3.18 **sdkDS402_QuickStop()** void sdkDS402_QuickStop (long busId)

Execute quick stop.

This function executes a quick stop with the defined deceleration. The deceleration can be set with method [sdkDS402_SetQuickStopDeceleration\(\)](#)

Parameters

<i>busId</i>	Bus ID of the connected slave
--------------	-------------------------------

Examples

[CAN_1Ax_DS402_ProfilePositionMode-ppm-Test.mc](#), [CAN_1Ax_DS402_ProfileVelocityMode-pvm-Test.mc](#), and [CAN_MultipleAx_DS402_ProfilePositionMode-ppm-Test.mc](#).

Definition at line 544 of file SDK_Amplifier_DS402_StateMachine.mc.

```
545 {
546     printf("Bus Id %d, Quick Stop\n", busId);
547     sdkDS402_TransitionToState(busId, DS402_DRIVE_STATE_QUICK_STOP_ACTIVE);
548 }
```

5.1.3.19 **sdkDS402_ReadControlWord()** `long sdkDS402_ReadControlWord (`
`long busId)`

Read control word.

This function reads the control word form index DS402_CONTROLWORD(0x6040) and subindex 0 The bits have different meaning for Profile Position Mode and Profile Velocity Mode

Parameters

<i>busId</i>	Bus ID of the connected slave
--------------	-------------------------------

Returns

value: control word

Definition at line 46 of file SDK_Amplifier_DS402_StateMachine.mc.

```
47 {
48     return SdoRead(busId, DS402_CONTROLWORD, 0);
49 }
```

5.1.3.20 **sdkDS402_ReadStatusWord()** `long sdkDS402_ReadStatusWord (`
`long busId)`

DS 402 State-Machine functions.

Read status word

This function reads the status word form index DS402_STATUSWORD(0x6041) and subindex 0 The bits have different meaning for Profile Position Mode and Profile Velocity Mode

Parameters

<i>busId</i>	Bus ID of the connected slave
--------------	-------------------------------

value: status word

Examples

[CAN_1Ax_DS402_ProfilePositionMode-ppm-Test.mc](#), [CAN_1Ax_DS402_ProfileVelocityMode-pvm-Test.mc](#), and [CAN_MultipleAx_DS402_ProfilePositionMode-ppm-Test.mc](#).

Definition at line 33 of file SDK_Amplifier_DS402_StateMachine.mc.

```
34 {
35     return SdoRead(busId, DS402_STATUSWORD, 0);
36 }
```

5.1.3.21 sdkDS402_ResetFault() void sdkDS402_ResetFault (
long busId)

Reset fault.

This function resets the fault execute a state transition to 'Switch On disabled'

Parameters

busId	Bus ID of the connected slave
-------	-------------------------------

Examples

[CAN_1Ax_DS402_ProfilePositionMode-ppm-Test.mc](#), [CAN_1Ax_DS402_ProfileVelocityMode-pvm-Test.mc](#), and [CAN_MultipleAx_DS402_ProfilePositionMode-ppm-Test.mc](#).

Definition at line 574 of file SDK_Amplifier_DS402_StateMachine.mc.

```
575 {
576     printf("Bus Id %d, Reset Fault\n", busId);
577     sdkDS402_TransitionToState(busId, DS402_DRIVE_STATE_SWITCH_ON_DISABLED);
578 }
```

5.1.3.22 sdkDS402_SetMaxAcceleration() void sdkDS402_SetMaxAcceleration (
long busId,
long acc)

Set the max acceleration DS402_MAX_ACCELERATION(0x60C5)

Set the maximum allowed acceleration to prevent mechanical damage.

Parameters

<i>busId</i>	Bus ID of the connected slave
<i>acc</i>	acceleration value

Definition at line 464 of file SDK_Amplifier_DS402_StateMachine.mc.

```
465 {
466     SdoWrite(busId, DS402_MAX_ACCELERATION, 0, acc);
467 }
```

5.1.3.23 sdkDS402_SetMaxMotorSpeed() `void sdkDS402_SetMaxMotorSpeed (`
 `long busId,`
 `long vel)`

Set the max profile velocity DS402_MAX_MOTOR_SPEED(0x6080)

Set the maximum allowed speed for the motor. It serves as protection for the motor. The value is given in [rpm].

Parameters

<i>busId</i>	Bus ID of the connected slave
<i>vel</i>	speed value [velocity unit]

Definition at line 452 of file SDK_Amplifier_DS402_StateMachine.mc.

```
453 {
454     SdoWrite(busId, DS402_MAX_MOTOR_SPEED, 0, vel);
455 }
```

5.1.3.24 sdkDS402_SetMaxProfileVelocity() `void sdkDS402_SetMaxProfileVelocity (`
 `long busId,`
 `long vel)`

Set the max profile velocity DS402_MAX_PROFILE_VELOCITY(0x607F)

Set the velocity limit in a PPM or PVM move.

Parameters

<i>busId</i>	Bus ID of the connected slave
<i>vel</i>	velocity value [velocity unit]

File Documentation

Definition at line 439 of file SDK_Amplifier_DS402_StateMachine.mc.

440 {

441 SdoWrite(busId, DS402_MAX_PROFILE_VELOCITY, 0, vel);

442 }

5.1.3.25 sdkDS402_SetMotionProfileType()

void sdkDS402_SetMotionProfileType (

long busId,

long type)

Set the motion profile type DS402_MOTION_PROFILE_TYPE(0x6086)

Set the type of motion profile trajectory

Parameters

<i>busId</i>	Bus ID of the connected slave
<i>type</i>	motion profile type value 0 : linear ramp (trapezoidal profile)

Definition at line 427 of file SDK_Amplifier_DS402_StateMachine.mc.

428 {

429 SdoWrite(busId, DS402_MOTION_PROFILE_TYPE, 0, type);

430 }

5.1.3.26 sdkDS402_SetOpartionMode()

void sdkDS402_SetOpartionMode (

long busId,

long operationMode)

Profile Mode functions.

Set the mode of operation DS402_MODES_OF_OPERATION(0x6060)

Set the mode of operation.

Parameters

<i>busId</i>	Bus ID of the connected slave
<i>operationMode</i>	Definition of the operation mode DS402_OP_PPM (1): Profile Position Mode (PPM) DS402_OP_PVM (3): Profile Velocity Mode (PVM)

[CAN_1Ax_DS402_ProfilePositionMode-ppm-Test.mc](#), [CAN_1Ax_DS402_ProfileVelocityMode-pvm-Test.m](#)
 and [CAN_MultipleAx_DS402_ProfilePositionMode-ppm-Test.mc](#).

Definition at line 508 of file SDK_Amplifier_DS402_StateMachine.mc.

```

509 {
510     if( operationMode == DS402_OP_PPM)
511     {
512         printf("Bus Id %d, setup profile position mode\n" ,busId);
513         SdoWrite( busId, DS402_MODES_OF_OPERATION, 0, DS402_OP_PPM);
514     }
515     else if(operationMode == DS402_OP_PVM)
516     {
517         printf("Bus Id %d, setup profile velocity mode\n" ,busId);
518         SdoWrite( busId, DS402_MODES_OF_OPERATION, 0, DS402_OP_PVM);
519     }
520 }
```

5.1.3.27 sdkDS402_SetProfileAcceleration() void sdkDS402_SetProfileAcceleration (

long *busId*,

long *acc*)

Profile Mode settings.

Set the profile acceleration DS402_PROFILE_ACCELERATION(0x6083)

Set the acceleration for PPM (Profile Position Mode) and PVM (Profile Velocity Mode)

Parameters

<i>busId</i>	Bus ID of the connected slave
<i>acc</i>	acceleration value [acceleration unit]

Examples

[CAN_1Ax_DS402_ProfileVelocityMode-pvm-Test.mc](#).

Definition at line 389 of file SDK_Amplifier_DS402_StateMachine.mc.

```

390 {
391     SdoWrite(busId, DS402_PROFILE_ACCELERATION, 0, acc);
392 }
```

5.1.3.28 sdkDS402_SetProfileDeceleration() void sdkDS402_SetProfileDeceleration (

long *busId*,

long *dec*)

Set the deceleration for PPM (Profile Position Mode) and PVM (Profile Velocity Mode)

Parameters

<i>bus↔ Id</i>	Bus ID of the connected slave
<i>acc</i>	deceleration value [acceleration unit]

Examples

[CAN_1Ax_DS402_ProfileVelocityMode-pvm-Test.mc](#).

Definition at line 401 of file SDK_Amplifier_DS402_StateMachine.mc.

```
402 {
403     SdoWrite(busId, DS402_PROFILE_DECELERATION, 0, dec);
404 }
```

5.1.3.29 sdkDS402_SetProfileMovementParameter() `void sdkDS402_SetProfileMovementParameter (`
 `long busId,`
 `long acc,`
 `long dec,`
 `long velocity)`

Set quick stop deceleration DS402_QUICK_STOP_DECELERATION(0x6085)

Set the deceleration of the quick stop profile.

Parameters

<i>bus↔ Id</i>	Bus ID of the connected slave
<i>dec</i>	deceleration value

Examples

[CAN_1Ax_DS402_ProfilePositionMode-ppm-Test.mc](#), and [CAN_MultipleAx_DS402_ProfilePositionMode-p](#)

Definition at line 488 of file SDK_Amplifier_DS402_StateMachine.mc.

```
489 {
490     sdkDS402_SetProfileAcceleration(busId, acc);
491     sdkDS402_SetProfileDeceleration(busId, dec);
492     sdkDS402_SetProfileVelocity(busId, velocity);
493 }
```

5.1.3.30 sdkDS402_SetProfileVelocity() `void sdkDS402_SetProfileVelocity (`
`long busId,`
`long vel)`

Set the profile velocity DS402_PROFILE_VELOCITY(0x6081)

Set the deceleration for PPM (Profile Position Mode) The target velocity for PVM has to be set with the function [sdkDS402_PVM_TargetReached\(\)](#)

Parameters

<i>busId</i>	Bus ID of the connected slave
<i>vel</i>	velocity value [velocity unit]

Definition at line 414 of file SDK_Amplifier_DS402_StateMachine.mc.

```
415 {
416     SdoWrite(busId, DS402_PROFILE_VELOCITY, 0, vel);
417 }
```

5.1.3.31 sdkDS402_SetQuickStopDeceleration() `void sdkDS402_SetQuickStopDeceleration (`
`long busId,`
`long dec)`

Set quick stop deceleration DS402_QUICK_STOP_DECELERATION(0x6085)

Set the deceleration of the quick stop profile.

Parameters

<i>busId</i>	Bus ID of the connected slave
<i>dec</i>	deceleration value

Examples

[CAN_1Ax_DS402_ProfilePositionMode-ppm-Test.mc](#), [CAN_1Ax_DS402_ProfileVelocityMode-pvm-Test.mc](#),
and [CAN_MultipleAx_DS402_ProfilePositionMode-ppm-Test.mc](#).

Definition at line 476 of file SDK_Amplifier_DS402_StateMachine.mc.

```
477 {
478     SdoWrite(busId, DS402_QUICK_STOP_DECELERATION, 0, dec);
479 }
```

```
long busId,  
long newState )
```

This function executes a state transition within the DS402 state machine. It gets the actual state and performs a transition to a new state, if the transition between those two states is allowed. It is not checked whether the target state has been achieved.

<i>busId</i>	Bus ID of the connected slave
<i>newState</i>	target state

- value: Process value
- value > 0 Transition executed
- value < 0 Invalid transition

CAN 1Ax DS402 ProfilePositionMode-ppm-Test.mc, and CAN MultipleAx DS402 ProfilePositionMode-p

```

230                                     {
231
232     long transition = -1;
233     long actState = sdkDS402_GetActDriveState(busId);
234
235     printf("Bus Id %d, Transition from ", busId);
236     sdkDS402_PrintState(actState);
237     printf(" to ");
238     sdkDS402_PrintState(newState);
239     print();
240
241     switch(actState) {
242     case DS402_DRIVE_STATE_NOT_READY_TO_SWITCH_ON:
243         if (newState == DS402_DRIVE_STATE_SWITCH_ON_DISABLED) {
244             // Automatic transition
245         }
246         break;
247     case DS402_DRIVE_STATE_SWITCH_ON_DISABLED:
248         if (newState == DS402_DRIVE_STATE_READY_TO_SWITCH_ON) {
249             transition = DS402_TRANSITION_SHUTDOWN;
250         }
251         break;
252     case DS402_DRIVE_STATE_READY_TO_SWITCH_ON:
253         if (newState == DS402_DRIVE_STATE_SWITCH_ON_DISABLED) {
254             transition = DS402_TRANSITION_DISABLE_VOLTAGE;
255         }
256         else if (newState == DS402_DRIVE_STATE_SWITCHED_ON) {
257             transition = DS402_TRANSITION_SWITCH_ON;
258         }
259         else if (newState == DS402_DRIVE_STATE_OPERATION_ENABLED) {
260             transition = DS402_TRANSITION_ENABLE_OPERATION;
261         }

```

File Documentation

```

262         break;
263     case DS402_DRIVE_STATE_SWITCHED_ON:
264         if (newState == DS402_DRIVE_STATE_SWITCH_ON_DISABLED) {
265             transition = DS402_TRANSITION_DISABLE_VOLTAGE;
266         }
267         else if (newState == DS402_DRIVE_STATE_READY_TO_SWITCH_ON) {
268             transition = DS402_TRANSITION_SHUTDOWN;
269         }
270         else if (newState == DS402_DRIVE_STATE_OPERATION_ENABLED) {
271             transition = DS402_TRANSITION_ENABLE_OPERATION;
272         }
273         break;
274     case DS402_DRIVE_STATE_OPERATION_ENABLED:
275         if (newState == DS402_DRIVE_STATE_SWITCH_ON_DISABLED) {
276             transition = DS402_TRANSITION_DISABLE_VOLTAGE;
277         }
278         else if (newState == DS402_DRIVE_STATE_READY_TO_SWITCH_ON) {
279             transition = DS402_TRANSITION_SHUTDOWN;
280         }
281         else if (newState == DS402_DRIVE_STATE_SWITCHED_ON) {
282             transition = DS402_TRANSITION_DISABLE_OPERATION;
283         }
284         else if (newState == DS402_DRIVE_STATE_QUICK_STOP_ACTIVE) {
285             transition = DS402_TRANSITION_QUICK_STOP;
286         }
287         break;
288     case DS402_DRIVE_STATE_QUICK_STOP_ACTIVE:
289         if (newState == DS402_DRIVE_STATE_SWITCH_ON_DISABLED) {
290             transition = DS402_TRANSITION_SHUTDOWN;
291         }
292         else if (newState == DS402_DRIVE_STATE_OPERATION_ENABLED) {
293             transition = DS402_TRANSITION_ENABLE_OPERATION;
294         }
295         break;
296     case DS402_DRIVE_STATE_FAULT_REACTION_ACTIVE:
297         if (newState == DS402_DRIVE_STATE_FAULT) {
298             // Automatic transition
299         }
300         else if (newState == DS402_DRIVE_STATE_SWITCH_ON_DISABLED) {
301             transition = DS402_TRANSITION_FAULT_RESET;
302         }
303         break;
304     case DS402_DRIVE_STATE_FAULT:
305         if (newState == DS402_DRIVE_STATE_SWITCH_ON_DISABLED) {
306             transition = DS402_TRANSITION_FAULT_RESET;
307         }
308         break;
309     default:
310         printf("...: invalid actual drive state: %d\n", busId, actState);
311         break;
312 }
313
314 if(transition != -1)
315 {
316     helperDS402_ExecuteTransition(busId, transition);
317 }
318 else
319 {
320     printf("...: invalid state transition\n", busId);
321 }
322
323 return(transition);
324 }

```

5.1.3.33 sdkDS402_WaitTransitionToState() long sdkDS402_WaitTransitionToState (

long busId,

```

long targetState,
long timeout )

```

Wait transition to another state.

This function waits (block) until a target state in DS402 state machine has been reached or the timeout expierd. It gets the actual state and performs a transition to a new state, if the transition between those two states is allowed. Because this function is blocking, just use it for a single axis. If you would like to change the state of several axis use [sdkDS402_TransitionToState\(\)](#) funciton and verify the transition in the application.

sdkDS402_TransitionToState

Parameters

<i>busId</i>	Bus ID of the connected slave
<i>newState</i>	target state
<i>Timeout</i>	until the function is aborted >=0 : timeout in [ms] < 0: infinite timeout

Returns

value: Process value
 value > 0 Transition executed (number of transition)
 value -1 Invalid transition
 value -2 Timeout expired
 value -3 Invalid actual state

Examples

[CAN_1Ax_DS402_ProfilePositionMode-ppm-Test.mc](#), and [CAN_1Ax_DS402_ProfileVelocityMode-pvm-Test.mc](#)

Definition at line 347 of file SDK_Amplifier_DS402_StateMachine.mc.

```

348 {
349     long actState, transition;
350     long time = Time();
351
352     printf("Bus Id %d, sdkDS402_WaitTransitionToState\n" ,busId);
353
354     transition = sdkDS402_TransitionToState(busId, targetState);
355     if(transition < 0)
356     {
357         //Invalid transition
358         return(-1);
359     }
360
361     actState = sdkDS402_GetActDriveState(busId);
362     while(actState != targetState)
363     {
364         if(timeout >= 0 && (Time() - time) > timeout)
365         {
366             //Timeout elapsed
367             printf("...: target state not reached within timeout (%d ms)\n", timeout);

```

File Documentation

```
368         return(-2);
369     }
370     actState = sdkDS402_GetActDriveState(busId);
371 }
372
373 return(transition);
374 }
```

5.1.3.34 sdkDS402_WriteControlWord() void sdkDS402_WriteControlWord (

```
    long busId,
    long value )
```

Write control word.

This function writes the control word to index DS402_CONTROLWORD(0x6040) and subindex 0 The bits have different meaning for Profile Position Mode and Profile Velocity Mode

Parameters

<i>busId</i>	Bus ID of the connected slave
<i>value</i>	Value of the control word

Returns

value: control word

Definition at line 60 of file SDK_Amplifier_DS402_StateMachine.mc.

```
61 {
62     SdoWrite(busId, DS402_CONTROLWORD, 0, value);
63 }
```

5.2 SDK_Amplifier_Epos4.mc File Reference

This file provides all the function used to setup a Epos4 drive.

```
#include <SysDef.mh>
#include "SDK_Amplifier_Epos4.mh"
```

Functions

- long [sdkEpos4_SetupECatBusModule](#) (long axis, long busId, long pdoNumber, long operationMode)
- Setup the ECAT bus module for an Epos4.*

- long [sdkEpos4_SetupECatVirtAmp](#) (long axis, long maxRpm, long operationMode)
Setup the virtual amplifier for an Epos4 with EtherCat.
- long [sdkEpos4_SetupECatVirtCntin](#) (long axis, long operationMode)
Setup the virtual counter input for an Epos4 with EtherCat.
- long [sdkEpos4_SetupECatSdoParam](#) (long busId, long pdoNumber, long axisPolarity, long operationMode)
Setup the Sdo parameter for an Epos4 with EtherCat.
- long [sdkEpos4_SetupCanBusModule](#) (long axis, long busId, long pdoNumber, long operationMode)
Setup the Can bus module for an Epos4.
- long [sdkEpos4_SetupCanVirtAmp](#) (long axis, long maxRpm, long operationMode)
Setup the virtual amplifier for an Epos4 with Can bus.
- long [sdkEpos4_SetupCanVirtCntin](#) (long axis, long operationMode)
Setup the virtual counter input for an Epos4 with Can bus.
- long [sdkEpos4_SetupCanSdoParam](#) (long busId, long pdoNumber, long axisPolarity, long operationMode)
Setup the Sdo parameter for an Epos4.
- long [sdkEpos4_AxisHomingStart](#) (long axis, long busId, long operationMode, long &homingState)
State machine function for performing a homing on an EPOS4.
- void [sdkEpos4_PrintErrorDescription](#) (long errorCode)
Prints a error description for Epos4 errors.

5.2.1 Detailed Description

This file provides all the function used to setup a Epos4 drive.

The Epos4 can be operated in the different modes csp and csv The amplifiers can be controlled via EtherCAT or CAN bus. The motor, encoder and controller parameters are set directly with "Epos Studio".

Revision

274

5.2.2 Function Documentation

5.2.2.1 sdkEpos4_AxisHomingStart() long sdkEpos4_AxisHomingStart (

```

    long axis,
    long busId,
    long operationMode,
    long & homingState )

```

State machine function for performing a homing on an EPOS4.

This function sets the EPOS4 in homing mode and starts the homing defined on the EPOS4. Before calling this function, all homing parameters must be configured on the EPOS4. The function is not blocking. The return parameter must be checked.

<i>axis</i>	Axis module number
<i>busId</i>	Bus ID of the connected slave
<i>operationMode</i>	Operation mod which should be set after homing.
<i>homingState</i>	Call-by-reference variable for the iteration of the different homing states. Must be initialized with 0.

Returns

value: Process value
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[CAN_1Ax_EPOS4-Test_csp.mc](#), [CAN_1Ax_EPOS4-Test_cst.mc](#), [CAN_1Ax_EPOS4-Test_csv.mc](#),
[CAN_2Ax_EPOS4-Test_csp.mc](#), [ECAT_1Ax_EPOS4-Test_cst.mc](#), and [ECAT_3Ax_Epos4-Test_csp.mc](#).

Definition at line 607 of file SDK_Amplifier_Epos4.mc.

```

609 {
610     long time, displayMode;
611     long homingMethode = 0;
612
613     time= Time()%1000;
614
615     switch (homingState) {
616         case 0:
617             print("sdkEpos4_AxisHomingStart:  ",axis);
618             homingMethode = SdoRead(busId, EPOS4_HOMING_METHOD,0);
619             print("...: EPOS4 Homing AxisNo:  ",axis," - Homing methode:  ",homingMethode);
620             homingState = 1;
621             break;
622         case 1:
623             print("...: EPOS4 Homing AxisNo:  ",axis," - Enable drive");
624             // The trajectory generator of the EPOS4 is used for homing. With this command the axis
is switched on
625             // and the trajectory generator of the MACS is switched off. Thus REG_COMPOS = REG_ACTPOS
is set.
626             AxisControl(axis, USERREFVEL);
627
628             // x6060 Change operation mode to "6: Homing mode."
629             SdoWriten( busId, EPOS4_MODES_OF_OPERATION, 0, 1, EPOS4_OP_HMM);
630             print("...: EPOS4 Homing AxisNo:  ",axis," - Set mode of OP: 0x06");
631
632             homingState = 2;
633
634         case 2:
635             displayMode = SdoRead(busId, EPOS4_MODES_OF_OPERATION_DISPLAY, 0);
636             if(displayMode== EPOS4_OP_HMM)
637             {
638                 print("...: EPOS4 Homing AxisNo:  ",axis," - Display mode of OP: ",displayMode);
639                 homingState = 3;
640             }
641             else if(time==0)
642             {
643                 printf("...: EPOS4 Homing AxisNo:  %ld - Waiting for OP mode display:
%lx\n",axis, VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS));
644                 // This section must be called in a 1 ms loop because of the print function.
645                 Delay(1);

```

```
646         }
647         break;
648
649     case 3:
650         // Bit 2 Operation enabled - EPOS4 is ready to start
651         // Bit 10 Target reached - Homing procedure is interrupted or not started
652         if(VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS).i[2]==1 &&
VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS).i[10]==1)
653         {
654             printf("...: EPOS4 Homing AxisNo: %ld - Homing start signal Status:
%lx\n",axis,VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS));
655             VIRTAMP_PARAM(axis, VIRTAMP_CNTRLW_PWRONENP) = 0x1F;
656             VIRTAMP_PARAM(axis, VIRTAMP_CNTRLW_PWRONENN) = 0x1F;
657             homingState = 4;
658         }
659         // Bit 13 Homingerror - Homing error occurred
660         else if(VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS).i[13]==1)
661         {
662             printf("...: EPOS4 Homing AxisNo: %ld - Homing Error - Status: %lx\n",axis,
VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS));
663             return(-1);
664         }
665         else if(time==0)
666         {
667             printf("...: EPOS4 Homing AxisNo: %ld - Waiting for ready bit: %lx\n",axis,
VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS));
668             // This section must be called in a 1 ms loop because of the print function.
669             Delay(1);
670         }
671         break;
672
673     case 4:
674         // Homing procedure is completed successfully
675         if (VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS).i[12]==1)
676         {
677             printf("...: EPOS4 Homing AxisNo: %ld - Homing done - Status: %lx\n",axis,
VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS));
678
679             VIRTAMP_PARAM(axis, VIRTAMP_CNTRLW_PWRONENP) = 0x0F;
680             VIRTAMP_PARAM(axis, VIRTAMP_CNTRLW_PWRONENN) = 0x0F;
681
682             SdoWriten(busId, 0x6060, 0, 1, operationMode ); // 0x6060 Change operation
mode to "xxx mode."
683
684             homingState = 5;
685             break;
686         }
687         // Bit 13 Homingerror - Homing error occurred
688         else if(VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS).i[13]==1)
689         {
690             printf("...: EPOS4 Homing AxisNo: %ld - Homing Error - Status: %lx\n",axis,
VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS));
691             return(-1);
692         }
693         // Homing procedure is in progress
694         else if(time==0)
695         {
696             printf("...: EPOS4 Homing AxisNo: %ld - Homing in progress - Status:
%lx\n",axis, VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS));
697             // This section must be called in a 1 ms loop because of the print function.
698             Delay(1);
699         }
700         break;
701
702     case 5:
703         displayMode = SdoRead(busId, EPOS4_MODES_OF_OPERATION_DISPLAY, 0);
704         if(displayMode== operationMode)
705         {
706             print("...: EPOS4 Homing AxisNo: ",axis," - Display mode of OP: ",displayMode);
707             homingState = 6;
```

File Documentation

```

708         }
709         else if (time==0)
710         {
711             printf("...: EPOS4 Homing AxisNo: %ld - Waiting for OP mode display:
712             %lx\n",axis, VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS));
713             // This section must be called in a 1 ms loop because of the print function.
714             Delay(1);
715         }
716         case 6:
717             print("...: EPOS4 Homing AxisNo: ",axis," - Disable Drive");
718             AxisControl(axis, OFF);
719             homingState = 7;
720         case 7:
721             return(1);
722         default :
723             print("...: EPOS4 Homing AxisNo: ",axis," - Incorrect hominState value: ",
724             homingState);
725             return(-1);
726     }
727     return(0);
728 }
```

5.2.2.2 sdkEpos4_PrintErrorDescription() void sdkEpos4_PrintErrorDescription (long errorCode)

Prints a error description for Epos4 errors.

This function prints a description for Epos4 error codes

Parameters

<i>errorCode</i>	Epos4 error code
------------------	------------------

Examples

[CAN_1Ax_EPOS4-Test_csp.mc](#), [CAN_1Ax_EPOS4-Test_cst.mc](#), [CAN_1Ax_EPOS4-Test_csv.mc](#), [CAN_2Ax_EPOS4-Test_csp.mc](#), [ECAT_1Ax_EPOS4-Test_cst.mc](#), and [ECAT_3Ax_Epos4-Test_csp.mc](#).

Definition at line 735 of file SDK_Amplifier_Epos4.mc.

```

736 {
737     switch (errorCode) {
738         case EPOS4_F_NO_ERROR:
739             printf("No error.");
740             break;
741         case EPOS4_F_GENERIC:
742             printf("Generic error.");
743             break;
744         case EPOS4_F_GENERIC_INIT1:
745         case EPOS4_F_GENERIC_INIT2:
746         case EPOS4_F_GENERIC_INIT3:
747         case EPOS4_F_GENERIC_INIT4:
748         case EPOS4_F_GENERIC_INIT5:
749         case EPOS4_F_GENERIC_INIT6:
750         case EPOS4_F_GENERIC_INIT7:
751         case EPOS4_F_GENERIC_INIT8:
752         case EPOS4_F_GENERIC_INIT9:
```

```
753         printf("Generic initialization error.");
754         break;
755     case EPOS4_F_FIRMWARE_INCOMPATIBILITY:
756         printf("Firmware incompatibility.");
757         break;
758     case EPOS4_F_OVERCURRENT:
759         printf("Overcurrent.");
760         break;
761     case EPOS4_F_POWER_STAGE_PROTECTION:
762         printf("Power stage protection.");
763         break;
764     case EPOS4_F_OVERVOLTAGE:
765         printf("Overvoltage.");
766         break;
767     case EPOS4_F_UNDERVOLTAGE:
768         printf("Undervoltage.");
769         break;
770     case EPOS4_F_THERMAL_OVERLOAD:
771         printf("Thermal overload.");
772         break;
773     case EPOS4_F_THERMAL_MOTOR_OVERLOAD:
774         printf("Thermal motor overload.");
775         break;
776     case EPOS4_F_LOGIC_VOLTAGE_LOW:
777         printf("Logic supply voltage too low.");
778         break;
779     case EPOS4_F_HARDWARE_DEFECT:
780         printf("Hardware defect.");
781         break;
782     case EPOS4_F_HARDWARE_INCOMPATIBILITY:
783         printf("Hardware incompatibility.");
784         break;
785     case EPOS4_F_HARDWARE1:
786     case EPOS4_F_HARDWARE2:
787     case EPOS4_F_HARDWARE3:
788     case EPOS4_F_HARDWARE4:
789         printf("Hardware error.");
790         break;
791     case EPOS4_F_SIGN_OF_LIFE:
792         printf("Sign of life.");
793         break;
794     case EPOS4_F_EXT_1_WATCHDOG:
795         printf("Extension 1 watchdog.");
796         break;
797     case EPOS4_F_INTERNAL_SOFTWARE:
798         printf("Internal software.");
799         break;
800     case EPOS4_F_SOFTWARE_PARAM:
801         printf("Software parameter error.");
802         break;
803     case EPOS4_F_PERSISTENT_PARAM_CORRUPT:
804         printf("Persistent param corrupt.");
805         break;
806     case EPOS4_F_POSITION_SENSOR:
807         printf("Position sensor error.");
808         break;
809     case EPOS4_F_POSITION_BREACH:
810         printf("Position breach.");
811         break;
812     case EPOS4_F_POSITION_RESOLUTION:
813         printf("Position resolution error.");
814         break;
815     case EPOS4_F_POSITION_INDEX:
816         printf("Position index.");
817         break;
818     case EPOS4_F_HALL_SENSOR:
819         printf("Hall sensor error.");
820         break;
821     case EPOS4_F_HALL_NOT_FOUND:
822         printf("Hall not found.");
```

File Documentation

```

823         break;
824     case EPOS4_F_HALL_ANGLE:
825         printf("Hall angle detection error.");
826         break;
827     case EPOS4_F_SSI_SENSOR:
828         printf("SSI sensor.");
829         break;
830     case EPOS4_F_SSI_FRAME:
831         printf("SSI frame.");
832         break;
833     case EPOS4_F_MISSING_MAIN_SENSOR:
834         printf("Missing main sensor.");
835         break;
836     case EPOS4_F_MISSING_COMM_SENSOR:
837         printf("Missing comm sensor.");
838         break;
839     case EPOS4_F_MAIN_SENSOR_DIR:
840         printf("Main sensor direction.");
841         break;
842     case EPOS4_F_CAN_OVERRUN_OBJ_LOST:
843         printf("CAN overrun (object lost).");
844         break;
845     case EPOS4_F_CAN_OVERRUN:
846         printf("CAN overrun.");
847         break;
848     case EPOS4_F_CAN_PASSIVE_MODE:
849         printf("CAN passive mode.");
850         break;
851     case EPOS4_F_CAN_HEARTBEAT:
852         printf("CAN heartbeat error.");
853         break;
854     case EPOS4_F_CAN_PDO_COB_ID_COLLISION:
855         printf("CAN PDO COB-ID collision.");
856         break;
857     case EPOS4_F_ETHERCAT_COMM:
858         printf("EtherCAT communication error.");
859         break;
860     case EPOS4_F_ETHERCAT_INIT:
861         printf("EtherCAT initialization error.");
862         break;
863     case EPOS4_F_ETHERCAT_RX_QUEUE:
864         printf("EtherCAT Rx queue overflow error.");
865         break;
866     case EPOS4_F_ETHERCAT_COMM_INTERNAL:
867         printf("EtherCAT communication (internal) error.");
868         break;
869     case EPOS4_F_ETHERCAT_CYCLE_TIME:
870         printf("EtherCAT communication cycle time error.");
871         break;
872     case EPOS4_F_CAN_BUS_OFF:
873         printf("CAN bus turned off.");
874         break;
875     case EPOS4_F_CAN_RX_QUEUE:
876         printf("CAN Rx queue overflow.");
877         break;
878     case EPOS4_F_CAN_TX_QUEUE:
879         printf("CAN Tx queue overflow.");
880         break;
881     case EPOS4_F_CAN_PDO_LENGTH:
882         printf("CAN PDO length error.");
883         break;
884     case EPOS4_F_RPDO_TIMEOUT:
885         printf("RPDO timeout.");
886         break;
887     case EPOS4_F_ETHERCAT_PDO_COMM:
888         printf("EtherCAT PDO communication error.");
889         break;
890     case EPOS4_F_ETHERCAT_SDO_COMM:
891         printf("EtherCAT SDO communication error.");
892         break;

```

```

893     case EPOS4_F_FOLLOWING:
894         printf("Following error.");
895         break;
896     case EPOS4_F_NEG_LIMIT_SWITCH:
897         printf("Negative limit switch error.");
898         break;
899     case EPOS4_F_POS_LIMIT_SWITCH:
900         printf("Positive limit switch error.");
901         break;
902     case EPOS4_F_SOFTWARE_POSITION_LIMIT:
903         printf("Software position limit error.");
904         break;
905     case EPOS4_F_STO:
906         printf("STO error.");
907         break;
908     case EPOS4_F_SYSTEM_OVERLOADED:
909         printf("System overloaded.");
910         break;
911     case EPOS4_F_WATCHDOG:
912         printf("Watchdog error.");
913         break;
914     case EPOS4_F_SYSTEM_PEAK_OVERLOADED:
915         printf("System peak overloaded.");
916         break;
917     case EPOS4_F_CONTROLLER_GAIN:
918         printf("Controller gain error.");
919         break;
920     case EPOS4_F_AUTO_TUNING_ID:
921         printf("Auto tuning identification error.");
922         break;
923     case EPOS4_F_AUTO_TUNING_CURRENT_LIMIT:
924         printf("Auto tuning current limit error.");
925         break;
926     case EPOS4_F_AUTO_TUNING_ID_CURRENT:
927         printf("Auto tuning identification current error.");
928         break;
929     case EPOS4_F_AUTO_TUNING_DATA_SAMPLING:
930         printf("Auto tuning data sampling error.");
931         break;
932     case EPOS4_F_AUTO_TUNING_SAMPLE_MISMATCH:
933         printf("Auto tuning sample mismatch error.");
934         break;
935     case EPOS4_F_AUTO_TUNING_PARAM:
936         printf("Auto tuning parameter error.");
937         break;
938     case EPOS4_F_AUTO_TUNING_AMPLITUDE_MISMATCH:
939         printf("Auto tuning amplitude mismatch error.");
940         break;
941     case EPOS4_F_AUTO_TUNING_TIMEOUT:
942         printf("Auto tuning timeout error.");
943         break;
944     case EPOS4_F_AUTO_TUNING_STANDSTILL:
945         printf("Auto tuning standstill error.");
946         break;
947     case EPOS4_F_AUTO_TUNING_TORQUE_INVALID:
948         printf("Auto tuning torque invalid error.");
949         break;
950     case EPOS4_F_AUTO_TUNING_MAX_SPEED:
951         printf("Auto tuning max system speed error.");
952         break;
953     case EPOS4_F_AUTO_TUNING_MOTOR_CONNECTION:
954         printf("Auto tuning motor connection error.");
955         break;
956     case EPOS4_F_AUTO_TUNING_SENSOR_SIGNAL:
957         printf("Auto tuning sensor signal error.");
958         break;
959     default:
960         if (errorCode >= 0x6180 && errorCode <= 0x61F0) //0x6180 through 0x61F0 is internal
software error
961         {

```

File Documentation

```
962         printf("Internal software error.");
963     }
964     else
965     {
966         printf("Unknown error code:  0x%04X\n", errorCode);
967     }
968     break;
969 }
970 }
```

```
5.2.2.3 sdkEpos4_SetupCanBusModule() long sdkEpos4_SetupCanBusModule (
    long axis,
    long busId,
    long pdoNumber,
    long operationMode )
```

Setup the Can bus module for an Epos4.

This function sets up the Can bus module for an Epos4. It can be defined with which operation mode is used. Currently there are no differences in the different operation modes regarding the Bus Module setup.

Parameters

<i>axis</i>	Axis module number
<i>busId</i>	Bus ID of the connected slave
<i>pdoNumber</i>	Used PDO number
<i>operationMode</i>	Definition of the operation mode 0x08: Cyclic synchronous position (csp) mode 0x09: Cyclic synchronous velocity (csv) mode 0x0A: Cyclic synchronous torque (cst) mode

Note

The value passed from "BUSMOD_RXMAP_POVALUE3" is the current torque. The value is given in per thousand of "Motor rated torque" and is signed.

Returns

- value: Always 1 in this function
- value > 0 Process successful
- value = 0 Process is active
- value < 0 Error

Examples

[CAN_1Ax_EPOS4-Test_csp.mc](#), [CAN_1Ax_EPOS4-Test_cst.mc](#), [CAN_1Ax_EPOS4-Test_csv.mc](#), and [CAN_2Ax_EPOS4-Test_csp.mc](#).

Definition at line 313 of file SDK_Amplifier_Epos4.mc.

```

314 {
315     long busmod, signedFlag;
316     busmod = axis;
317     signedFlag = 0x10000000;
318
319     print("sdkEpos4_SetupCanBusModule: ", busmod);
320
321     BUSMOD_PARAM(busmod, BUSMOD_MODE) = BUSMOD_MODE_DEACTIVATE;           // Deactivate (objects
will be deleted)
322     BUSMOD_PARAM(busmod, BUSMOD_BUSTYPE) = BUSMOD_BUSTYPE_CAN;           // Can bus
323     BUSMOD_PARAM(busmod, BUSMOD_MODE) = BUSMOD_MODE_ACTIVATE;           // Create bus module
324
325     BUSMOD_PARAM(busmod, BUSMOD_BUSNO) = busId / 100000;                 // Bus number (0 = master,
1 = slave CAN bus)
326     BUSMOD_PARAM(busmod, BUSMOD_ID) = busId % 1000;                     // CAN Id for this bus
module
327     BUSMOD_PARAM(busmod, BUSMOD_SYNC) = 1;                               // Sync active
328     BUSMOD_PARAM(busmod, BUSMOD_GUARDTIME) = 0;                         // no guarding
329
330     BUSMOD_PARAM(busmod, BUSMOD_PISRC_INPUT1) = VIRTAMP_PROCESS_SRCINDEX(axis, PO_VIRTAMP_CMDWORD);
// Select the input value sources object (send to bus): CMD Word
331
332     if (operationMode == EPOS4_OP_CSP) // cycle synchronous position (csp) mode
333     {
334         BUSMOD_PARAM(busmod, BUSMOD_PISRC_INPUT2) = VIRTAMP_PROCESS_SRCINDEX(axis, PO_VIRTAMP_REFPOS);
// Select the input value sources object (send to bus): position setpoint
335         print("...: cycle synchronous position (csp) mode");
336     }
337     else if (operationMode == EPOS4_OP_CSV) // cycle synchronous velocity (csv) mode
338     {
339         BUSMOD_PARAM(busmod, BUSMOD_PISRC_INPUT2) = VIRTAMP_PROCESS_SRCINDEX(axis, PO_VIRTAMP_REFVEL);
// Select the input value sources object (send to bus): velocity setvel
340         print("...: cycle synchronous velocity (csv) mode");
341     }
342     else if (operationMode == EPOS4_OP_CST) // cycle synchronous torque (cst) mode
343     {
344         BUSMOD_PARAM(busmod, BUSMOD_PISRC_INPUT2) = AXE_PROCESS_SRCINDEX(axis, REG_USERREFCUR); //
Select the input value sources object (send to bus): target torque
345         print("...: cycle synchronous torque (cst) mode");
346     }
347     else // Error
348     {
349         print("...: not supported operation mode");
350         return(-1);
351     }
352
353     BUSMOD_PARAM(busmod, BUSMOD_TXMAP_INPUT1) = pdoNumber*0x01000000 + 2*0x00010000 + 0; // pdo ;
length in bytes; bytes offset control word
354     BUSMOD_PARAM(busmod, BUSMOD_TXMAP_INPUT2) = pdoNumber*0x01000000 + 4*0x00010000 + 2; // pdo ;
length in bytes; bytes offset target position/ velocity
355
356     BUSMOD_PARAM(busmod, BUSMOD_RXMAP_POVALUE1) = pdoNumber*0x01000000 + 2*0x00010000 + 0; // pdo ;
length in bytes; bytes offset status word
357     BUSMOD_PARAM(busmod, BUSMOD_RXMAP_POVALUE2) = pdoNumber*0x01000000 + 4*0x00010000 + 2; // pdo ;
length in bytes; bytes offset position actual value
358     BUSMOD_PARAM(busmod, BUSMOD_RXMAP_POVALUE3) = signedFlag + pdoNumber*0x01000000 + 2*0x00010000 +
6; // pdo ; length in bytes; bytes offset torque actual value
359
360     BUSMOD_PARAM(busmod, BUSMOD_MODE) = BUSMOD_MODE_ACTIVATE_NOSTOP;     // Start
bus module
361
362     return(1);
363 }

```

5.2.2.4 sdkEpos4_SetupCanSdoParam() long sdkEpos4_SetupCanSdoParam (

File Documentation

```

    long busId,
    long pdonumber,
    long axisPolarity,
    long operationMode )

```

Setup the Sdo parameter for an Epos4.

This function sets up the Sdo parameter for an Epos4. It can be defined with which operation mode is used. The CANSYNCTIMER must be set before calling this function.

Parameters

<i>busId</i>	Bus ID of the connected slave
<i>pdoNumber</i>	Used PDO number
<i>axisPolarity</i>	Definition of the polarity, 0 : Normal polarity, 1 : Inverse polarity
<i>operationMode</i>	Definition of the operation mode 0x08 : Cyclic synchronous position (csp) mode 0x09 : Cyclic synchronous velocity (csv) mode 0x0A : Cyclic synchronous torque (cst) mode

Returns

- value: Process value
- value > 0 Process successful
- value = 0 Process is active
- value < 0 Error

Examples

[CAN_1Ax_EPOS4-Test_csp.mc](#), [CAN_1Ax_EPOS4-Test_cst.mc](#), [CAN_1Ax_EPOS4-Test_csv.mc](#), and [CAN_2Ax_EPOS4-Test_csp.mc](#).

Definition at line 497 of file SDK_Amplifier_Epos4.mc.

```

499 {
500     long nodeid, last_value;
501     nodeid = busId % 100000;
502
503     print("sdkEpos4_SetupCanSdoParam:  ",nodeid);
504
505     if (axisPolarity==1){ // set axis polarity
506         SdoWrite( busId, EPOS4_AXIS_CONFIGURATION, 4, 1);
507         print("...:  change axis direction");
508     }
509     else{
510         SdoWrite( busId, EPOS4_AXIS_CONFIGURATION, 4, 0);
511     }
512
513     //interpolation time periode value
514     SdoWrite(busId, EPOS4_INTERPOLATION_TIME_PERIOD, 1, GLB_PARAM(CANSYNCTIMER));
515     print("...:  set interpolation time periode:  ", GLB_PARAM(CANSYNCTIMER), " ms");
516
517     // reset Transmitttype
518     SdoWrite(busId, EPOS4_TRANSMIT_PDO_1_PARAMETER, 2, 255); // TxPDO1 Transmitttype
519     SdoWrite(busId, EPOS4_TRANSMIT_PDO_2_PARAMETER, 2, 255); // TxPDO2 Transmitttype
520     SdoWrite(busId, EPOS4_TRANSMIT_PDO_3_PARAMETER, 2, 255); // TxPDO3 Transmitttype
521     SdoWrite(busId, EPOS4_TRANSMIT_PDO_4_PARAMETER, 2, 255); // TxPDO4 Transmitttype

```

```
522
523 // the pdos have to be disabled for configuring
524 last_value = (SdoRead(busId, EPOS4_TRANSMIT_PDO_1_PARAMETER, 1));
525 SdoWrite(busId, EPOS4_TRANSMIT_PDO_1_PARAMETER, 1, (last_value | 0x80000000)); // disable pdo 1
526 last_value = (SdoRead(busId, EPOS4_TRANSMIT_PDO_2_PARAMETER, 1));
527 SdoWrite(busId, EPOS4_TRANSMIT_PDO_2_PARAMETER, 1, (last_value | 0x80000000)); // disable pdo 2
528 last_value = (SdoRead(busId, EPOS4_TRANSMIT_PDO_3_PARAMETER, 1));
529 SdoWrite(busId, EPOS4_TRANSMIT_PDO_3_PARAMETER, 1, (last_value | 0x80000000)); // disable pdo 3
530 last_value = (SdoRead(busId, EPOS4_TRANSMIT_PDO_4_PARAMETER, 1));
531 SdoWrite(busId, EPOS4_TRANSMIT_PDO_4_PARAMETER, 1, (last_value | 0x80000000)); // disable pdo 4
532
533 //the pdos have to be disabled for configuring
534 last_value = (SdoRead(busId, EPOS4_RECEIVE_PDO_1_PARAMETER, 1));
535 SdoWrite(busId, EPOS4_RECEIVE_PDO_1_PARAMETER, 1, (last_value | 0x80000000)); // disable pdo 1
536 last_value = (SdoRead(busId, EPOS4_RECEIVE_PDO_2_PARAMETER, 1));
537 SdoWrite(busId, EPOS4_RECEIVE_PDO_2_PARAMETER, 1, (last_value | 0x80000000)); // disable pdo 2
538 last_value = (SdoRead(busId, EPOS4_RECEIVE_PDO_3_PARAMETER, 1));
539 SdoWrite(busId, EPOS4_RECEIVE_PDO_3_PARAMETER, 1, (last_value | 0x80000000)); // disable pdo 3
540 last_value = (SdoRead(busId, EPOS4_RECEIVE_PDO_4_PARAMETER, 1));
541 SdoWrite(busId, EPOS4_RECEIVE_PDO_4_PARAMETER, 1, (last_value | 0x80000000)); // disable pdo 4
542
543 // Now we setup the correct PDO transmission for Tx and Rx
544 SdoWrite(busId, (EPOS4_TRANSMIT_PDO_1_PARAMETER + pdonumber-1), 2, 1); // TxPDO
Transmitttype SYNC
545 SdoWrite(busId, (EPOS4_RECEIVE_PDO_1_PARAMETER + pdonumber-1), 2, 1); // RxPDO
Transmitttype SYNC
546
547 // config RX PDO
548 SdoWrite(busId, (EPOS4_RECEIVE_PDO_1_MAPPING + pdonumber-1), 0x00, 0); // disable
549 SdoWrite(busId, (EPOS4_RECEIVE_PDO_1_MAPPING + pdonumber-1), 0x01, 0x60400010); // controlword
550
551 if (operationMode == EPOS4_OP_CSP) { // cycle
synchronous position (csp) mode
552 SdoWrite(busId, (EPOS4_RECEIVE_PDO_1_MAPPING + pdonumber-1), 2, 0x607A0020); // download
pdo 0x1600 entry: position set point
553 SdoWrite(busId, EPOS4_MODES_OF_OPERATION, 0, EPOS4_OP_CSP); // mode of
operation CSP
554 print("...: cycle synchronous position (csp) mode");
555 }
556 else if (operationMode == EPOS4_OP_CSV ) {
557 SdoWrite(busId, (EPOS4_RECEIVE_PDO_1_MAPPING + pdonumber-1), 2, 0x60FF0020); // download
pdo 0x1600 entry: target velocity 32bit
558 SdoWrite(busId, EPOS4_MODES_OF_OPERATION, 0, EPOS4_OP_CSV); // mode of
operation CSV
559 print("...: cycle synchronous velocity (csv) mode");
560 }
561 else if (operationMode == EPOS4_OP_CST) {
562 SdoWrite(busId, (EPOS4_RECEIVE_PDO_1_MAPPING + pdonumber-1), 2, 0x60710010); // download
pdo 0x1600 entry: Target torque 32bit
563 SdoWrite(busId, EPOS4_MODES_OF_OPERATION, 0, EPOS4_OP_CST); // mode of
operation CST
564 print("...: cycle synchronous torque (cst) mode");
565 }
566 else { // Error
567 print("...: not supported operation mode");
568 return(-1);
569 }
570 SdoWrite(busId, (EPOS4_RECEIVE_PDO_1_MAPPING + pdonumber - 1), 0x00, 2); // enable
571
572 // config TX PDO
573 SdoWrite(busId, (EPOS4_TRANSMIT_PDO_1_MAPPING + pdonumber - 1), 0x00, 0); // disable
574 SdoWrite(busId, (EPOS4_TRANSMIT_PDO_1_MAPPING + pdonumber - 1), 0x01, 0x60410010); // statusword
575 SdoWrite(busId, (EPOS4_TRANSMIT_PDO_1_MAPPING + pdonumber - 1), 0x02, 0x60640020); // actpos
576 SdoWrite(busId, (EPOS4_TRANSMIT_PDO_1_MAPPING + pdonumber - 1), 0x03, 0x30D20110); // Torque
actual value averaged
577 SdoWrite(busId, (EPOS4_TRANSMIT_PDO_1_MAPPING + pdonumber - 1), 0x00, 3 ); // enable
578
579 // fill in the wanted pdo
580 SdoWrite(busId, (EPOS4_TRANSMIT_PDO_1_PARAMETER + pdonumber - 1), 1, (0x80000180 + (pdonumber -
1)*0x100 + nodeid) );// fill in pdo
```

File Documentation

```

581     SdoWrite(busId, (EPOS4_RECEIVE_PDO_1_PARAMETER + pdonumber - 1), 1, (0x80000200 + (pdonumber -
582     1)*0x100 + nodeid) );// fill in pdo
583     // enable
584     SdoWrite(busId, (EPOS4_TRANSMIT_PDO_1_PARAMETER + pdonumber - 1), 1, (0x00000180 + (pdonumber -
585     1)*0x100 + nodeid) );// enable pdo
586     SdoWrite(busId, (EPOS4_RECEIVE_PDO_1_PARAMETER + pdonumber - 1), 1, (0x00000200 + (pdonumber -
587     1)*0x100 + nodeid) );// enable pdo
588 }

```

5.2.2.5 sdkEpos4_SetupCanVirtAmp() long sdkEpos4_SetupCanVirtAmp (

```

    long axis,
    long maxRpm,
    long operationMode )

```

Setup the virtual amplifier for an Epos4 with Can bus.

This function sets up the virtual amplifier with Can bus for an Epos4. It can be defined with which operation mode is used.

Parameters

<i>axis</i>	Axis module number
<i>maxRpm</i>	Definition of the maximum motor speed [rpm]
<i>operationMode</i>	Definition of the operation mode 0x08: Cyclic synchronous position (csp) mode 0x09: Cyclic synchronous velocity (csv) mode 0x0A: Cyclic synchronous torque (cst) mode

Note

The value passed from "PO_BUSMOD_VALUE3" to "VIRTAMP_PISRC_CURRENT" is the current torque. The value is given in per thousand of "Motor rated torque".

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[CAN_1Ax_EPOS4-Test_csp.mc](#), [CAN_1Ax_EPOS4-Test_cst.mc](#), [CAN_1Ax_EPOS4-Test_csv.mc](#), and [CAN_2Ax_EPOS4-Test_csp.mc](#).

Definition at line 383 of file SDK_Amplifier_Epos4.mc.

```

384 {
385     print("sdkEpos4_SetupCanVirtAmp: ",axis );
386
387     // virtual amplifiers have a fixed connection to axes number, axes 0 uses amplifier 0.
388     VIRTAMP_PARAM(axis,VIRTAMP_PISRC_CMDWORD)      = AXE_PROCESS_SRCINDEX(axis,REG_CNTRLWORD);
389     VIRTAMP_PARAM(axis,VIRTAMP_PISRC_REFPOS)        = AXE_PROCESS_SRCINDEX(axis,REG_COMPOS);
390     VIRTAMP_PARAM(axis,VIRTAMP_PISRC_REFVEL)        = AXE_PROCESS_SRCINDEX(axis,REG_REFERENCE);
391     VIRTAMP_PARAM(axis,VIRTAMP_PISRC_REFACC)        = AXE_PROCESS_SRCINDEX(axis,PID_FFACCPART);
392     VIRTAMP_PARAM(axis,VIRTAMP_PISRC_STATUS)        = BUSMOD_PROCESS_SRCINDEX(axis,PO_BUSMOD_VALUE1);
393     VIRTAMP_PARAM(axis,VIRTAMP_PISRC_CURRENT)       = BUSMOD_PROCESS_SRCINDEX(axis, PO_BUSMOD_VALUE3);
394
395     VIRTAMP_PARAM(axis,VIRTAMP_CNTRLW_PWROFF)       = 0x06;
396     VIRTAMP_PARAM(axis,VIRTAMP_CNTRLW_PWRONDIS)     = 0x06;
397     VIRTAMP_PARAM(axis,VIRTAMP_CNTRLW_PWRONENP)     = 0x0F;
398     VIRTAMP_PARAM(axis,VIRTAMP_CNTRLW_PWRONENN)     = 0x0F;
399     VIRTAMP_PARAM(axis,VIRTAMP_CNTRLW_QUICKSTOP)    = 0x02;
400     VIRTAMP_PARAM(axis,VIRTAMP_CNTRLW_RESET)        = 0x80;
401
402     VIRTAMP_PARAM(axis,VIRTAMP_STOPDELAY)           = 0x00;
403     VIRTAMP_PARAM(axis,VIRTAMP_ERROR_BITMASK)       = 0x08;
404     VIRTAMP_PARAM(axis,VIRTAMP_ERROR_POLARITY)       = 1;
405
406     if (operationMode == EPOS4_OP_CSP) // cycle synchronous position (csp) mode
407     {
408         VIRTAMP_PARAM(axis,VIRTAMP_REF100PERC)      = 0; // not used in csp
409         VIRTAMP_PARAM(axis,VIRTAMP_REFLIMIT)        = 0; // not used in csp
410         print("...: cycle synchronous position (csp) mode");
411     }
412     else if (operationMode == EPOS4_OP_CSV) // cycle synchronous velocity (csv) mode
413     {
414         VIRTAMP_PARAM(axis,VIRTAMP_REF100PERC)      = maxRpm;
415         VIRTAMP_PARAM(axis,VIRTAMP_REFLIMIT)        = 2* maxRpm;
416         print("...: cycle synchronous velocity (csv) mode");
417     }
418     else if (operationMode == EPOS4_OP_CST) // cycle synchronous torque (cst) mode
419     {
420         VIRTAMP_PARAM(axis,VIRTAMP_REF100PERC)      = 0; // not used in cst
421         VIRTAMP_PARAM(axis,VIRTAMP_REFLIMIT)        = 0; // not used in cst
422         print("...: cycle synchronous torque (cst) mode");
423     }
424     else // Error
425     {
426         print("...: not supported operation mode");
427         return(-1);
428     }
429
430     VIRTAMP_PARAM(axis,VIRTAMP_MODE) = VIRTAMP_MODE_ENABLE; // has to be the last one because it
activates all
431
432     return(1);
433 }

```

5.2.2.6 sdkEpos4_SetupCanVirtCntin() long sdkEpos4_SetupCanVirtCntin (

long axis,

long operationMode)

Setup the virtual counter input for an Epos4 with Can bus.

This function sets up the virtual counter input for an Epos4. It can be defined with which operation mode is used.

<i>axis</i>	Axis module number
<i>operationMode</i>	Definition of the operation mode 0x08 : Cyclic synchronous position (csp) mode 0x09 : Cyclic synchronous velocity (csv) mode 0x0A : Cyclic synchronous torque (cst) mode

Returns

value: Process value
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[CAN_1Ax_EPOS4-Test_csp.mc](#), [CAN_1Ax_EPOS4-Test_cst.mc](#), [CAN_1Ax_EPOS4-Test_csv.mc](#),
and [CAN_2Ax_EPOS4-Test_csp.mc](#).

Definition at line 450 of file SDK_Amplifier_Epos4.mc.

```
451 {
452     print("sdkEpos4_SetupCanVirtCntin: ",axis);
453
454     VIRTCONTIN_PARAM(axis,VIRTCONTIN_PISRC_COUNTER) = BUSMOD_PROCESS_SRCINDEX(axis, PO_BUSMOD_VALUE2);
455
456     if (operationMode == EPOS4_OP_CSP) // cycle synchronous position (csp) mode
457     {
458         VIRTCONTIN_PARAM(axis,VIRTCONTIN_MODE) = VIRTCONTIN_MODE_ABSOLUTE_DIRECT; // Source is
absolute and is taken as it is
459         print("...: cycle synchronous position (csp) mode");
460     }
461     else if (operationMode == EPOS4_OP_CSV) // cycle synchronous velocity (csv) mode
462     {
463         VIRTCONTIN_PARAM(axis,VIRTCONTIN_MODE) = VIRTCONTIN_MODE_ABSOLUTE; // Source is a
position value and difference to last value is added
464         print("...: cycle synchronous velocity (csv) mode");
465     }
466     else if (operationMode == EPOS4_OP_CST) // cycle synchronous torque (cst) mode
467     {
468         VIRTCONTIN_PARAM(axis,VIRTCONTIN_MODE) = VIRTCONTIN_MODE_ABSOLUTE_DIRECT; // Source is
absolute and is taken as it is
469         print("...: cycle synchronous torque (cst) mode");
470     }
471     else // Error
472     {
473         print("...: not supported operation mode");
474         return(-1);
475     }
476
477     return(1);
478 }
```

5.2.2.7 sdkEpos4_SetupECatBusModule()

long sdkEpos4_SetupECatBusModule (

long axis,

long busId,

long pdoNumber,

long operationMode)

Setup the ECAT bus module for an Epos4.

This function sets up the ECAT bus module for an Epos4. It can be defined with which operation mode is used. Currently there are no differences in the different operation modes regarding the Bus Module setup.

Parameters

<i>axis</i>	Axis module number
<i>busId</i>	Bus ID of the connected slave
<i>pdoNumber</i>	Used PDO number
<i>operationMode</i>	Definition of the operation mode 1: Cyclic synchronous position (csp) mode 2: Cyclic synchronous velocity (csv) mode

Returns

- value: Process value
- value > 0 Process successful
- value = 0 Process is active
- value < 0 Error

Examples

[DeltaRobot_No_SM_EPOS4_ECAT.mc](#), [ECAT_1Ax_EPOS4-Test_cst.mc](#), and [ECAT_3Ax_Epos4-Test_cs](#)

Definition at line 37 of file SDK_Amplifier_Epos4.mc.

38 {

39 long busmod, slaveNo;

40

41 busmod = axis;

42 slaveNo = (busId-1000000);

43

44 print("sdkEpos4_SetupECatBusModule: ",busmod);

45

46 BUSMOD_PARAM(busmod, BUSMOD_MODE) = BUSMOD_MODE_DEACTIVATE; // Deactivate (objects will be

deleted)

47 BUSMOD_PARAM(busmod, BUSMOD_BUSTYPE) = BUSMOD_BUSTYPE_ECATE_M; // EtherCAT Master

48 BUSMOD_PARAM(busmod, BUSMOD_ID) = slaveNo; // Select the nodeid of the

connected slave

49

50 BUSMOD_PARAM(busmod, BUSMOD_PISRC_INPUT1) = VIRTAMP_PROCESS_SRCINDEX(axis,PO_VIRTAMP_CMDWORD);

// Select the input value sources object (send to bus): CMD Word

51

52 if (operationMode==EPOS4_OP_CSP) // cycle synchronous position (csp) mode

53 {

54 BUSMOD_PARAM(busmod, BUSMOD_PISRC_INPUT2) = VIRTAMP_PROCESS_SRCINDEX(axis,PO_VIRTAMP_REFPOS);

// Select the input value sources object (send to bus): position setpoint

55 print("...: cycle synchronous position (csp) mode");

56 }

57 else if (operationMode==EPOS4_OP_CSV) // cycle synchronous velocity (csv) mode

File Documentation

```

58     {
59         BUSMOD_PARAM(busmod, BUSMOD_PISRC_INPUT2) = VIRTAMP_PROCESS_SRCINDEX(axis, PO_VIRTAMP_REFVEL);
// Select the input value sources object (send to bus): velocity setvel
60         print("...: cycle synchronous velocity (csv) mode");
61     }
62     else if (operationMode == EPOS4_OP_CST) // cycle synchronous torque (cst) mode
63     {
64         BUSMOD_PARAM(busmod, BUSMOD_PISRC_INPUT2) = AXE_PROCESS_SRCINDEX(axis, REG_USERREFCUR); //
Select the input value sources object (send to bus): target torque
65         print("...: cycle synchronous torque (cst) mode");
66     }
67     else // Error
68     {
69         print("...: not supported operation mode");
70         return(-1);
71     }
72
73     BUSMOD_PARAM(busmod, BUSMOD_TXMAP_INPUT1) = pdoNumber*0x01000000 + 2*0x00010000 + 0; // pdo ;
length in bytes; bytes offset control word
74     BUSMOD_PARAM(busmod, BUSMOD_TXMAP_INPUT2) = pdoNumber*0x01000000 + 4*0x00010000 + 2; // pdo ;
length in bytes; bytes offset target position/ velocity
75
76     BUSMOD_PARAM(busmod, BUSMOD_RXMAP_POVALUE1) = pdoNumber*0x01000000 + 2*0x00010000 + 0; // pdo ;
length in bytes; bytes offset status word
77     BUSMOD_PARAM(busmod, BUSMOD_RXMAP_POVALUE2) = pdoNumber*0x01000000 + 4*0x00010000 + 2; // pdo ;
length in bytes; bytes offset position actual value
78     BUSMOD_PARAM(busmod, BUSMOD_RXMAP_POVALUE3) = pdoNumber*0x01000000 + 4*0x00010000 + 6; // pdo ;
length in bytes; bytes offset current actual value
79
80     BUSMOD_PARAM(busmod, BUSMOD_MODE) = BUSMOD_MODE_ACTIVATE_NOSTOP; // Start bus
module
81
82     return(1);
83 }

```

5.2.2.8 sdkEpos4_SetupECatSdoParam() long sdkEpos4_SetupECatSdoParam (

long busId,

long pdoNumber,

long axisPolarity,

long operationMode)

Setup the Sdo parameter for an Epos4 with EtherCat.

This function sets up the Sdo parameter with EtherCat for an Epos4. It can be defined with which operation mode is used.

Parameters

<i>busId</i>	Bus ID of the connected slave
<i>pdoNumber</i>	Used PDO number
<i>axisPolarity</i>	Definition of the polarity, 0 : Normal polarity, 1 : Inverse polarity
<i>operationMode</i>	Definition of the operation mode 0x08 : Cyclic synchronous position (csp) mode 0x09 : Cyclic synchronous velocity (csv) mode 0x0A : Cyclic synchronous torque (cst) mode

Returns

value: Process value
 value > 0 Process successful
 value = 0 Process is active
 value < 0 Error

Examples

[DeltaRobot_No_SM_EPOS4_ECAT.mc](#), [ECAT_1Ax_EPOS4-Test_cst.mc](#), and [ECAT_3Ax_Epos4-Test_cs](#)

Definition at line 209 of file SDK_Amplifier_Epos4.mc.

```

210 {
211     long sm_TX = EPOS4_SYNC_MANAGER_2_PDO_ASSIGNMENT;
212     long sm_RX = EPOS4_SYNC_MANAGER_3_PDO_ASSIGNMENT;
213
214     print("sdkEpos4_SetupECatSdoParam Ax: ",busId," pdoNumber ",pdoNumber);
215
216     SdoWriten( busId, sm_TX, 0x00, 1, 0x00);           // disable entry
217     SdoWriten( busId, sm_RX, 0x00, 1, 0x00);           // disable entry
218
219     SdoWriten( busId, EPOS4_TRANSMIT_PDO_1_MAPPING, 0, 1, 0);           // clear pdo 0x1a00
220     entry: SdoWriten( busId, EPOS4_TRANSMIT_PDO_1_MAPPING, 1, 4, 0x60410010);           // download pdo 0x1A00
221     entry: SdoWriten( busId, EPOS4_TRANSMIT_PDO_1_MAPPING, 2, 4, 0x60640020);           // download pdo 0x1A00
222     entry: SdoWriten( busId, EPOS4_TRANSMIT_PDO_1_MAPPING, 3, 4, 0x30D10120);           // download pdo 0x1A00
223     entry: SdoWriten( busId, EPOS4_TRANSMIT_PDO_1_MAPPING, 0, 1, 3);           // download pdo 0x1A00
224     entry: number of entries
225
226     SdoWriten( busId, EPOS4_TRANSMIT_PDO_2_MAPPING, 0, 1, 0);           // clear pdo 0x1a01
227     entry: SdoWriten( busId, EPOS4_TRANSMIT_PDO_3_MAPPING, 0, 1, 0);           // clear pdo 0x1a02
228     entry: SdoWriten( busId, EPOS4_TRANSMIT_PDO_4_MAPPING, 0, 1, 0);           // clear pdo 0x1a03
229     entry:
230
231     SdoWriten( busId, EPOS4_RECEIVE_PDO_1_MAPPING, 0, 1, 0);           // clear pdo 0x1600
232     entry: SdoWriten( busId, EPOS4_RECEIVE_PDO_1_MAPPING, 1, 4, 0x60400010);           // download pdo 0x1600
233     entry: controlword
234
235     if (operationMode==EPOS4_OP_CSP)           // cycle synchronous position (csp) mode
236     {
237         SdoWriten( busId, EPOS4_RECEIVE_PDO_1_MAPPING, 2, 4, 0x607A0020);           // download pdo 0x1600
238     entry: position set point
239     }
240     else if (operationMode==EPOS4_OP_CSV)           // cycle synchronous velocity (csv) mode
241     {
242         SdoWriten( busId, EPOS4_RECEIVE_PDO_1_MAPPING, 2, 4, 0x60FF0020);           // download pdo 0x1600
243     entry: target velocity 32bit
244     }
245     else if (operationMode == EPOS4_OP_CST) {
246         SdoWriten(busId, (EPOS4_RECEIVE_PDO_1_MAPPING), 2, 4, 0x60710010);           // download pdo 0x1600
247     entry: Target torque 32bit
248     }
249
250     SdoWriten( busId, EPOS4_RECEIVE_PDO_1_MAPPING, 0, 1, 2);           // download pdo 0x1600
251     entry: number of entries
252
253     SdoWriten( busId, EPOS4_RECEIVE_PDO_2_MAPPING, 0, 1, 0);           // clear pdo 0x1601
254     entry: SdoWriten( busId, EPOS4_RECEIVE_PDO_3_MAPPING, 0, 1, 0);           // clear pdo 0x1602
255     entry: SdoWriten( busId, EPOS4_RECEIVE_PDO_4_MAPPING, 0, 1, 0);           // clear pdo 0x1603
256     entry:

```

File Documentation

```

249
250
251     SdoWriten( busId, sm_TX, 1, 2, 0x1600);           // download pdo 0x1C1@b
252     2:01 index
253     SdoWriten( busId, sm_TX, 0, 1, 1);               // download pdo 0x1C12
254     count
255     SdoWriten( busId, sm_RX, 1, 2, 0x1A00 );         // download pdo 0x1C13:01
256     index
257     SdoWriten( busId, sm_RX, 0, 1, 1);               // download pdo 0x1C13
258     count
259
260     SdoWriten( busId, EPOS4_INTERPOLATION_TIME_PERIOD, 1, 0, 1); //interpolation time
261     periode value
262
263     if (axisPolarity==1) // set axis polarity
264     {
265         SdoWriten( busId, EPOS4_AXIS_CONFIGURATION, 4, 4, 1);
266         print("...:  change axis direction");
267     }
268     else
269     {
270         SdoWriten( busId, EPOS4_AXIS_CONFIGURATION, 4, 4, 0);
271     }
272
273     if (operationMode==EPOS4_OP_CSP)
274     {
275         SdoWriten( busId, EPOS4_MODES_OF_OPERATION, 0, 1, EPOS4_OP_CSP);           // cycle
276         synchronous position (csp) mode
277         print("...:  cycle synchronous position (csp) mode");
278     }
279     else if (operationMode==EPOS4_OP_CSV)
280     {
281         SdoWriten( busId, EPOS4_MODES_OF_OPERATION, 0, 1, EPOS4_OP_CSV);           // cycle
282         synchronous velocity (csv) mode
283         print("...:  cycle synchronous velocity (csv) mode");
284     }
285     else if (operationMode == EPOS4_OP_CST) {
286         SdoWriten(busId, EPOS4_MODES_OF_OPERATION, 0, 1, EPOS4_OP_CST);           // cycle
287         synchronous torque (cst) mode
288         print("...:  cycle synchronous torque (cst) mode");
289     }
290     else // Error
291     {
292         print("...:  not supported operation mode");
293         return(-1);
294     }
295
296     return(1);
297 }

```

5.2.2.9 sdkEpos4_SetupECatVirtAmp() long sdkEpos4_SetupECatVirtAmp (

```

    long axis,
    long maxRpm,
    long operationMode )

```

Setup the virtual amplifier for an Epos4 with EtherCat.

This function sets up the virtual amplifier with EtherCat for an Epos4. It can be defined with which operation mode is used.

Parameters

<i>axis</i>	Axis module number
<i>maxRpm</i>	Definition of the maximum motor speed [rpm]
<i>operationMode</i>	Definition of the operation mode 0x08: Cyclic synchronous position (csp) mode 0x09: Cyclic synchronous velocity (csv) mode 0x0A: Cyclic synchronous torque (cst) mode

Returns

value: Process value
 value > 0 Process successful
 value = 0 Process is active
 value < 0 Error

Examples

[DeltaRobot_No_SM_EPOS4_ECAT.mc](#), [ECAT_1Ax_EPOS4-Test_cst.mc](#), and [ECAT_3Ax_Epos4-Test_cs](#)

Definition at line 101 of file SDK_Amplifier_Epos4.mc.

```

102 {
103     print("sdkEpos4_SetupECatVirtAmp: ",axis );
104
105     // virtual amplifiers have a fixed connection to axes number, axes 0 uses amplifier 0.
106     VIRTAMP_PARAM(axis,VIRTAMP_PISRC_CMDWORD)      = AXE_PROCESS_SRCINDEX(axis,REG_CNTRLWORD);
107     VIRTAMP_PARAM(axis,VIRTAMP_PISRC_REFPOS)        = AXE_PROCESS_SRCINDEX(axis,REG_COMPOS);
108     VIRTAMP_PARAM(axis,VIRTAMP_PISRC_REFVEL)        = AXE_PROCESS_SRCINDEX(axis,REG_REFERENCE);
109     VIRTAMP_PARAM(axis,VIRTAMP_PISRC_REFACC)        = AXE_PROCESS_SRCINDEX(axis,PID_FFACCPART);
110     VIRTAMP_PARAM(axis,VIRTAMP_PISRC_STATUS)        = BUSMOD_PROCESS_SRCINDEX(axis,PO_BUSMOD_VALUE1);
111     VIRTAMP_PARAM(axis,VIRTAMP_PISRC_CURRENT)       = BUSMOD_PROCESS_SRCINDEX(axis, PO_BUSMOD_VALUE3);
112
113     VIRTAMP_PARAM(axis,VIRTAMP_CNTRLW_PWROFF)       = 0x06;
114     VIRTAMP_PARAM(axis,VIRTAMP_CNTRLW_PWRONDIS)     = 0x06;
115     VIRTAMP_PARAM(axis,VIRTAMP_CNTRLW_PWRONENP)     = 0x0F;
116     VIRTAMP_PARAM(axis,VIRTAMP_CNTRLW_PWRONENN)     = 0x0F;
117     VIRTAMP_PARAM(axis,VIRTAMP_CNTRLW_QUICKSTOP)    = 0x02;
118     VIRTAMP_PARAM(axis,VIRTAMP_CNTRLW_RESET)        = 0x80;
119
120     VIRTAMP_PARAM(axis,VIRTAMP_STOPDELAY)           = 0x0;
121     VIRTAMP_PARAM(axis,VIRTAMP_ERROR_BITMASK)       = 0x0008;
122     VIRTAMP_PARAM(axis,VIRTAMP_ERROR_POLARITY)      = 1;
123
124     if (operationMode==EPOS4_OP_CSP)                // cycle synchronous position (csp) mode
125     {
126         VIRTAMP_PARAM(axis,VIRTAMP_REF100PERC)      = 0; // not used in csp
127         VIRTAMP_PARAM(axis,VIRTAMP_REFLIMIT)        = 0; // not used in csp
128         print("...: cycle synchronous position (csp) mode");
129     }
130     else if (operationMode==EPOS4_OP_CSV)            // cycle synchronous velocity (csv) mode
131     {
132         VIRTAMP_PARAM(axis,VIRTAMP_REF100PERC)      = maxRpm;
133         VIRTAMP_PARAM(axis,VIRTAMP_REFLIMIT)        = 2* maxRpm;
134         print("...: cycle synchronous velocity (csv) mode");
135     }
136     else if (operationMode == EPOS4_OP_CST)          // cycle synchronous torque (cst) mode
137     {
138         VIRTAMP_PARAM(axis,VIRTAMP_REF100PERC)      = 0; // not used in cst
139         VIRTAMP_PARAM(axis,VIRTAMP_REFLIMIT)        = 0; // not used in cst
140         print("...: cycle synchronous torque (cst) mode");
141     }

```

File Documentation

```

142     else        // Error
143     {
144         print("...:  not supported operation mode");
145         return(-1);
146     }
147
148     VIRTAMP_PARAM(axis,VIRTAMP_MODE) = VIRTAMP_MODE_ENABLE;        // has to be the last one because it
    activates all
149
150     return(1);
151 }
```

5.2.2.10 sdkEpos4_SetupECatVirtCntin() long sdkEpos4_SetupECatVirtCntin (

long axis,

long operationMode)

Setup the virtual counter input for an Epos4 with EtherCat.

This function sets up the virtual counter input with EtherCat for an Epos4. It can be defined with which operation mode is used.

Parameters

<i>axis</i>	Axis module number
<i>operationMode</i>	Definition of the operation mode 1: Cyclic synchronous position (csp) mode 2: Cyclic synchronous velocity (csv) mode

Returns

value: Process value
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[DeltaRobot_No_SM_EPOS4_ECAT.mc](#), [ECAT_1Ax_EPOS4-Test_cst.mc](#), and [ECAT_3Ax_Epos4-Test_csp.mc](#).

Definition at line 168 of file SDK_Amplifier_Epos4.mc.

```

169 {
170     print("sdkEpos4_SetupECatVirtCntin:  ",axis);
171
172     VIRTCOUNTIN_PARAM(axis,VIRTCTIN_PISRC_COUNTER) = BUSMOD_PROCESS_SRCINDEX(axis, PO_BUSMOD_VALUE2);
173
174     if (operationMode==EPOS4_OP_CSP)        // cycle synchronous position (csp) mode
175     {
176         VIRTCOUNTIN_PARAM(axis,VIRTCTIN_MODE) = VIRTCTIN_MODE_ABSOLUTE_DIRECT;    // Source is
    absolute and is taken as it is
177         print("...:  cycle synchronous position (csp) mode");
178     }
179     else if (operationMode==EPOS4_OP_CSV)    // cycle synchronous velocity (csv) mode
180     {
```

```

181     VIRTCONTIN_PARAM(axis,VIRTCONTIN_MODE) = VIRTCONTIN_MODE_ABSOLUTE;
    position value and difference to last value is added
182     print("...: cycle synchronous velocity (csv) mode");
183 }
184 else // Error
185 {
186     print("...: not supported operation mode");
187     return(-1);
188 }
189
190 return(1);
191 }

```

5.3 SDK_Amplifier_MACS.mc File Reference

Functions for the integrated amplifier setup.

```

#include <SysDef.mh>
#include "SDK_Amplifier_MACS.mh"

```

Functions

- long [sdkSetupAmpDcMotor](#) (long axis, long controlMode, long polePairs, long maxCur, long encQc, long maxRpm)

Sets the amplifier parameters for a DC motor.

- long [sdkSetupAmpBlDc120Motor](#) (long axis, long controlMode, long polePairs, long maxCur, long encQc, long maxRpm)

Sets the amplifier parameters for a BLDC 120° motor

Deprecated: Use [sdkSetupAmpBlDcMotor\(\)](#) and HWAMP_HALL_ALIGNMENT.

- long [sdkSetupAmpBlDcMotor](#) (long axis, long hallAlignent, long controlMode, long polePairs, long maxCur, long encQc, long maxRpm)

Sets the amplifier parameters for a BLDC motor.

- long [sdkSetupAmpStepMotor_CL](#) (long axis, long controlMode, long steps, long maxCur, long encQc, long maxRpm)

Sets the amplifier parameters for a stepper motor (closed loop)

- long [sdkSetupAmpStepMotor_OL](#) (long axis, long steps, long maxCur, long maxRpm)

Sets the amplifier parameters for a stepper motor (closed loop)

- long [sdkSetupAmpPmsmMotor](#) (long axis, long controlMode, long polePairs, long maxCur, long encQc, long maxRpm)

Sets the amplifier parameters for a brushless, PMSM commuted motor (no Hall sensors are used)

File Documentation

- long [sdkSetupAmpHallPmsmMotor](#) (long axis, long controlMode, long polePairs, long maxCur, long encQc, long maxRpm, long elPol)

Sets the amplifier parameters for a brushless, PMSM commuted motor (Hall sensors are used)

- long [sdkSetupCurrentPIControl](#) (long axis, long curkprop, long curkint, long curkilim)

Set parameters for PI current control loop.

- long [sdkSetupVelocityPIControl](#) (long axis, long velkprop, long velkint, long velkilim)

Set parameters for PI velocity control loop.

- long [sdkSetupVirtualI2T](#) (long axis, long nominalCur, long thermalTime)

Function to generate a virtual I2T protection.

5.3.1 Detailed Description

Functions for the integrated amplifier setup.

Revision

243

5.3.2 Function Documentation

5.3.2.1 [sdkSetupAmpBldc120Motor\(\)](#) long [sdkSetupAmpBldc120Motor](#) (

```

    long axis,
    long controlMode,
    long polePairs,
    long maxCur,
    long encQc,
    long maxRpm )

```

Sets the amplifier parameters for a BLDC 120° motor

Deprecated: Use [sdkSetupAmpBldcMotor\(\)](#) and HWAMP_HALL_ALIGNMENT.

The function sets the amplifier parameters for a BLDC 120° motor. Additionally the controller mode can be selected. Depending on the mode, the controllers must then be parameterized.

The following functions are available: [sdkSetupPositionPIDControl\(\)](#) / [sdkSetupPositionPIDControlExt\(\)](#), [sdkSetupCurrentPIControl\(\)](#), [sdkSetupVelocityPIControl\(\)](#)

Parameters

<i>axis</i>	Axis module number
<i>controlMode</i>	Define control typ (default 3) 0 HWAMP_MODE_POS_VEL_CUR: Pos -> Vel -> Cur -> PWM 1 HWAMP_MODE_POS_VEL: Pos -> Vel -> PWM 2 HWAMP_MODE_POS_CUR: Pos -> Cur -> PWM 3 HWAMP_MODE_POS: Pos -> PWM
<i>polePairs</i>	Number of pole pairs (default 1)
<i>maxCur</i>	Maximal current allowed in mA (default 2000)
<i>encQc</i>	Resolution of the encoder for position feed back in increments qc (default 2000)
<i>maxRpm</i>	Maximum velocity in RPM, Use for scale the parameters for the velocity controller (default 1500)

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Definition at line 85 of file SDK_Amplifier_MACS.mc.

```
86 {  
87     HWAMP_PARAM(axis, HWAMP_COMMTYPE) = HWAMP_COMMTYPE_BLDC_120;    // Set motor type  
88     HWAMP_PARAM(axis, HWAMP_MODE)     = controlMode;                // Set controller principle  
89     HWAMP_PARAM(axis, HWAMP_POLES)     = polePairs;                  // Number of pole pairs  
90     HWAMP_PARAM(axis, HWAMP_MAXCUR)    = maxCur;                    // Max current in mA  
91     HWAMP_PARAM(axis, HWAMP_ENCRES)    = encQc;                      // Given in qc  
92     HWAMP_PARAM(axis, HWAMP_MAXRPM)    = maxRpm;                     // Given in RPM  
93  
94     return(1);  
95 }
```

5.3.2.2 sdkSetupAmpBldcMotor() long sdkSetupAmpBldcMotor (
 long axis,
 long hallAligment,
 long controlMode,
 long polePairs,
 long maxCur,
 long encQc,
 long maxRpm)

Sets the amplifier parameters for a BLDC motor.

The function sets the amplifier parameters for a BLDC motor. Additionally the controller mode can be selected. Depending on the mode, the controllers must then be parameterized.

The following functions are available: [sdkSetupPositionPIDControl\(\)](#) / [sdkSetupPositionPIDControlExt\(\)](#), [sdkSetupCurrentPIControl\(\)](#), [sdkSetupVelocityPIControl\(\)](#)

<i>axis</i>	Axis module number
<i>hallAlignment</i>	Hall alignment. See SDO Dictionary Index: 4000, SubIndex: 88 (default 5)
<i>controlMode</i>	Define control typ (default 3) 0 HWAMP_MODE_POS_VEL_CUR: Pos -> Vel -> Cur -> PWM 1 HWAMP_MODE_POS_VEL: Pos -> Vel -> PWM 2 HWAMP_MODE_POS_CUR: Pos -> Cur -> PWM 3 HWAMP_MODE_POS: Pos -> PWM
<i>polePairs</i>	Number of pole pairs (default 1)
<i>maxCur</i>	Maximal current allowed in mA (default 2000)
<i>encQc</i>	Resolution of the encoder for position feed back in increments qc (default 2000)
<i>maxRpm</i>	Maximum velocity in RPM, Use for scale the parameters for the velocity controller (default 1500)

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[Maxon_EC45_flat_1ax_BC_Enc.mc](#), and [MotorCommissioning_MaxonECi30.mc](#).

Definition at line 122 of file SDK_Amplifier_MACS.mc.

```
123 {  
124     HWAMP_PARAM(axis, HWAMP_COMMTYPE) = HWAMP_COMMTYPE_BLDC;           // Set motor type  
125     HWAMP_PARAM(axis, HWAMP_HALL_ALIGNMENT) = hallAlignment;           // set hall alignment  
126     HWAMP_PARAM(axis, HWAMP_MODE) = controlMode;                       // Set controller priciple  
127     HWAMP_PARAM(axis, HWAMP_POLES) = polePairs;                        // Number of pole pairs  
128     HWAMP_PARAM(axis, HWAMP_MAXCUR) = maxCur;                         // Max current in mA  
129     HWAMP_PARAM(axis, HWAMP_ENCRES) = encQc;                           // Given in qc  
130     HWAMP_PARAM(axis, HWAMP_MAXRPM) = maxRpm;                          // Given in RPM  
131  
132     return(1);  
133 }
```

5.3.2.3 sdkSetupAmpDcMotor() long sdkSetupAmpDcMotor (
 long *axis*,
 long *controlMode*,
 long *polePairs*,
 long *maxCur*,
 long *encQc*,
 long *maxRpm*)

Sets the amplifier parameters for a DC motor.

The function sets the amplifier parameters for a DC motor. Additionally the controller mode can be selected. Depending on the mode, the controllers must then be parameterized.

The following functions are available: [sdkSetupPositionPIDControl\(\)](#) / [sdkSetupPositionPIDControlExt\(\)](#), [sdkSetupCurrentPIControl\(\)](#), [sdkSetupVelocityPIControl\(\)](#)

<i>axis</i>	Axis module number
<i>controlMode</i>	Define control typ (default 3) 0 HWAMP_MODE_POS_VEL_CUR: Pos -> Vel -> Cur -> PWM 1 HWAMP_MODE_POS_VEL: Pos -> Vel -> PWM 2 HWAMP_MODE_POS_CUR: Pos -> Cur -> PWM 3 HWAMP_MODE_POS: Pos -> PWM
<i>polePairs</i>	Number of pole pairs (default 1)
<i>maxCur</i>	Maximal current allowed in mA (default 2000)
<i>encQc</i>	Resolution of the encoder for position feed back in increments qc (default 2000)
<i>maxRpm</i>	Maximum velocity in RPM, Use for scale the parameters for the velocity controller (default 1500)

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[Maxon_RE_40_1ax_Inc.mc](#), and [Maxon_RE_40_1ax_OL.mc](#).

Definition at line 48 of file SDK_Amplifier_MACS.mc.

```

49 {
50     HWAMP_PARAM(axis, HWAMP_COMMTYPE) = HWAMP_COMMTYPE_DC;           // Set motor type
51     HWAMP_PARAM(axis, HWAMP_MODE)     = controlMode;                 // Set controller priciple
52     HWAMP_PARAM(axis, HWAMP_POLES)    = polePairs;                   // Number of pole pairs
53     HWAMP_PARAM(axis, HWAMP_MAXCUR)   = maxCur;                     // Max current in mA
54     HWAMP_PARAM(axis, HWAMP_ENCRESES) = encQc;                       // Given in qc
55     HWAMP_PARAM(axis, HWAMP_MAXRPM)   = maxRpm;                       // Given in RPM
56
57     return(1);
58 }
```

5.3.2.4 sdkSetupAmpHallPmsmMotor()

```

long sdkSetupAmpHallPmsmMotor (
    long axis,
    long controlMode,
    long polePairs,
    long maxCur,
    long encQc,
    long maxRpm,
    long elPol )
```

Sets the amplifier parameters for a brushless, PMSM commuted motor (Hall sensors are used)

The function sets the amplifier parameters for a brushless, PMSM commuted motor. Additionally the controller mode can be selected. Depending on the mode, the controllers must then be parameterized.

The following functions are available: [sdkSetupPositionPIDControl\(\)](#) / [sdkSetupPositionPIDControlExt\(\)](#), [sdkSetupCurrentPIControl\(\)](#), [sdkSetupVelocityPIControl\(\)](#)

Parameters

<i>axis</i>	Axis module number
<i>controlMode</i>	Define control typ (default 3) 0 HWAMP_MODE_POS_VEL_CUR: Pos -> Vel -> Cur -> PWM 1 HWAMP_MODE_POS_VEL: Pos -> Vel -> PWM 2 HWAMP_MODE_POS_CUR: Pos -> Cur -> PWM 3 HWAMP_MODE_POS: Pos -> PWM
<i>polePairs</i>	Number of pole pairs (default 1)
<i>maxCur</i>	Maximal current allowed in mA (default 2000)
<i>encQc</i>	Resolution of the encoder for position feed back in increments qc (default 2000)
<i>maxRpm</i>	Maximum velocity in RPM, Use for scale the parameters for the velocity controller (default 1500)
<i>elPol</i>	Encoder polarity vs. electrical polarity (default -1) 1 HWAMP_ELPOL_REGULAR: Electrical polarity is equal encoder's polarity -1 HWAMP_ELPOL_INVERS: Electrical polarity is inverse to encoder's polarity

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[Maxon_EC45_flat_1ax_SC_Hall_Inc.mc](#), [Maxon_ECi30_2ax_SC_Hall_Inc.mc](#), [Maxon_ECi52_1ax_SC_SS](#)
and [MotorCommissioning_MaxonECi30.mc](#).

Definition at line 268 of file SDK_Amplifier_MACS.mc.

```

269 {
270     HWAMP_PARAM(axis, HWAMP_COMMTYPE) = HWAMP_COMMTYPE_HALL_PMSM; // Set motor type
271     HWAMP_PARAM(axis, HWAMP_ELPOL)    = elPol; // encoder polarity vs.
    electrical polarity:
272     HWAMP_PARAM(axis, HWAMP_MODE)      = controlMode; // Set controller priciple
273     HWAMP_PARAM(axis, HWAMP_POLES)     = polePairs; // Number of pole pairs
274     HWAMP_PARAM(axis, HWAMP_MAXCUR)    = maxCur; // Max current in mA
275     HWAMP_PARAM(axis, HWAMP_ENCRESES)  = encQc; // Given in qc
276     HWAMP_PARAM(axis, HWAMP_MAXRPM)    = maxRpm; // Given in RPM
277
278     return(1);
279 }
```

5.3.2.5 sdkSetupAmpPmsmMotor() long sdkSetupAmpPmsmMotor (

long axis,

long controlMode,

long polePairs,

long maxCur,

```
long encQc,  
long maxRpm )
```

Sets the amplifier parameters for a brushless, PMSM commuted motor (no Hall sensors are used)

The function sets the amplifier parameters for a brushless, PMSM commuted motor. Additionally the controller mode can be selected. Depending on the mode, the controllers must then be parameterized.

The following functions are available: [sdkMotorAlignment\(\)](#) [sdkSetupPositionPIDControl\(\)](#) / [sdkSetupPositionPIDControl\(\)](#) [sdkSetupCurrentPIControl\(\)](#), [sdkSetupVelocityPIControl\(\)](#)

Parameters

<i>axis</i>	Axis module number
<i>controlMode</i>	Define control typ (default 3) 0 HWAMP_MODE_POS_VEL_CUR: Pos -> Vel -> Cur -> PWM 1 HWAMP_MODE_POS_VEL: Pos -> Vel -> PWM 2 HWAMP_MODE_POS_CUR: Pos -> Cur -> PWM 3 HWAMP_MODE_POS: Pos -> PWM
<i>polePairs</i>	Number of pole pairs (default 1)
<i>maxCur</i>	Maximal current allowed in mA (default 2000)
<i>encQc</i>	Resolution of the encoder for position feed back in increments qc (default 2000)
<i>maxRpm</i>	Maximum velocity in RPM, Use for scale the parameters for the velocity controller (default 1500)

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[Maxon_ECi40_1ax_SC_OL_Inc.mc](#), and [Maxon_ECi40_3ax_SC_OL_Inc.mc](#).

Definition at line 230 of file SDK_Amplifier_MACS.mc.

```
231 {  
232     HWAMP_PARAM(axis, HWAMP_COMMTYPE) = HWAMP_COMMTYPE_PMSM;           // Set motor type  
233     HWAMP_PARAM(axis, HWAMP_MODE)     = controlMode;                  // Set controller principle  
234     HWAMP_PARAM(axis, HWAMP_POLES)    = polePairs;                    // Number of pole pairs  
235     HWAMP_PARAM(axis, HWAMP_MAXCUR)   = maxCur;                      // Max current in mA  
236     HWAMP_PARAM(axis, HWAMP_ENCRES)   = encQc;                        // Given in qc  
237     HWAMP_PARAM(axis, HWAMP_MAXRPM)   = maxRpm;                      // Given in RPM  
238  
239     return(1);  
240 }
```

5.3.2.6 sdkSetupAmpStepMotor_CL() `long sdkSetupAmpStepMotor_CL (`
`long axis,`
`long controlMode,`
`long steps,`
`long maxCur,`
`long encQc,`
`long maxRpm)`

Sets the amplifier parameters for a stepper motor (closed loop)

The function sets the amplifier parameters for a stepper motor in closed loop. Additionally the controller mode can be selected. Depending on the mode, the controllers must then be parameterized.

The following functions are available: [sdkSetupPositionPIDControl\(\)](#) / [sdkSetupPositionPIDControlExt\(\)](#), [sdkSetupCurrentPIControl\(\)](#), [sdkSetupVelocityPIControl\(\)](#)

Parameters

<i>axis</i>	Axis module number
<i>controlMode</i>	Define control typ (default 3) 0 HWAMP_MODE_POS_VEL_CUR: Pos -> Vel -> Cur -> PWM 1 HWAMP_MODE_POS_VEL: Pos -> Vel -> PWM 2 HWAMP_MODE_POS_CUR: Pos -> Cur -> PWM 3 HWAMP_MODE_POS: Pos -> PWM
<i>steps</i>	Steps per revolution
<i>maxCur</i>	Maximal current allowed in mA (default 2000)
<i>encQc</i>	Resolution of the encoder for position feed back in increments qc (default 2000)
<i>maxRpm</i>	Maximum velocity in RPM, Use for scale the parameters for the velocity controller (default 1500)

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[Stepper_XY_CL.mc.](#)

Definition at line 159 of file SDK_Amplifier_MACS.mc.

```

160 {
161     HWAMP_PARAM(axis, HWAMP_COMMTYPE) = HWAMP_COMMTYPE_STEP;           // Set motor type
162     HWAMP_PARAM(axis, HWAMP_MODE)     = controlMode;                   // Set controller principle
163     HWAMP_PARAM(axis, HWAMP_POLES)    = steps/4;                        // For 2-phase stepper: Poles =
    Steps per revolution / 4
164     HWAMP_PARAM(axis, HWAMP_MAXCUR)   = maxCur;                       // Max current in mA
165     HWAMP_PARAM(axis, HWAMP_ENCRES)   = encQc;                         // Given in qc
166     HWAMP_PARAM(axis, HWAMP_MAXRPM)   = maxRpm;                        // Given in RPM
167
168     return(1);
169 }
```

5.3.2.7 sdkSetupAmpStepMotor_OL() long sdkSetupAmpStepMotor_OL (

```

    long axis,
    long steps,
    long maxCur,
    long maxRpm )

```

Sets the amplifier parameters for a stepper motor (closed loop)

The function sets the amplifier parameters for a stepper motor in closed loop. Additionally the controller mode can be selected. Depending on the mode, the controllers must then be parameterized.

The following functions are available: [sdkSetupPositionPIDControl\(\)](#) / [sdkSetupPositionPIDControlExt\(\)](#), [sdkSetupCurrentPIControl\(\)](#), [sdkSetupVelocityPIControl\(\)](#)

Parameters

<i>axis</i>	Axis module number
<i>controlMode</i>	Define control typ (default 3) 0 HWAMP_MODE_POS_VEL_CUR: Pos -> Vel -> Cur -> PWM 1 HWAMP_MODE_POS_VEL: Pos -> Vel -> PWM 2 HWAMP_MODE_POS_CUR: Pos -> Cur -> PWM 3 HWAMP_MODE_POS: Pos -> PWM
<i>steps</i>	Steps per revolution
<i>maxCur</i>	Maximal current allowed in mA (default 2000)
<i>maxRpm</i>	Maximum velocity in RPM, Use for scale the parameters for the velocity controller (default 1500)

Returns

value: Always 1 in this function
 value > 0 Process successful
 value = 0 Process is active
 value < 0 Error

Examples

[Stepper_XY_OL.mc.](#)

Definition at line 193 of file SDK_Amplifier_MACS.mc.

```

194 {
195     HWAMP_PARAM(axis, HWAMP_COMMTYPE) = HWAMP_COMMTYPE_STEP;           // Set motor type
196     HWAMP_PARAM(axis, HWAMP_MODE)     = HWAMP_MODE_PFG_CUR_PWM;         // Set controller principle
197     HWAMP_PARAM(axis, HWAMP_POLES)     = steps/4;                       // For 2-phase stepper: Poles =
    Steps per revolution / 4
198     HWAMP_PARAM(axis, HWAMP_MAXCUR)    = maxCur;                       // Max current in mA
199     HWAMP_PARAM(axis, HWAMP_ENCRES)    = HWAMP_PARAM(axis, HWAMP_POLES) * HWAMP_ENCRES_MICROSTEP_RES; //
    Given in qc
200     HWAMP_PARAM(axis, HWAMP_MAXRPM)    = maxRpm;                         // Given in RPM
201
202     return(1);
203 }

```

5.3.2.8 sdkSetupCurrentPIControl() long sdkSetupCurrentPIControl (

long axis,

long curkprop,

long curkint,

long curkilim)

Set parameters for PI current control loop.

The function sets the factors of the current PI controller. More information are available in the Help of ApossIDE.

Parameters

<i>axis</i>	Axis module number
<i>curkprop</i>	Proportional factor of current controller (default 200)
<i>curkint</i>	Integral factor of current controller (default 100)
<i>curkilim</i>	Integral limit of current controller (default 1000)

Returns

- value: Always 1 in this function
- value > 0 Process successful
- value = 0 Process is active
- value < 0 Error

Examples

[Maxon_EC45_flat_1ax_BC_Enc.mc](#), [Maxon_EC45_flat_1ax_BC_Hall.mc](#), [Maxon_EC45_flat_1ax_SC_Hall.mc](#), [Maxon_ECi30_2ax_SC_Hall_Inc.mc](#), [Maxon_ECi40_1ax_SC_OL_Inc.mc](#), [Maxon_ECi40_3ax_SC_OL_Inc.mc](#), [Maxon_ECi52_1ax_SC_SSI.mc](#), [Maxon_RE_40_1ax_Inc.mc](#), [Maxon_RE_40_1ax_OL.mc](#), [MotorCommissiioning.mc](#), [Stepper_XY_CL.mc](#), and [Stepper_XY_OL.mc](#).

Definition at line 293 of file SDK_Amplifier_MACS.mc.

```
294 {
295     // parameters for current controller
296     HWAMP_PARAM((axis), HWAMP_CURKPROP) = curkprop; // current-controller proportional factor:
+/-32767 (CURKPROP)
297     HWAMP_PARAM((axis), HWAMP_CURKINT) = curkint; // current-controller integral factor:
+/-32767 (CURKINT)
298     HWAMP_PARAM((axis), HWAMP_CURKILIM) = curkilim; // current-controller Integral limit:
0 ... 32767 (CURKILIM)
299
300     return(1);
301 }
```

5.3.2.9 sdkSetupVelocityPIControl() long sdkSetupVelocityPIControl (

long axis,

long velkprop,

File Documentation

```
long velkint,  
long velkilim )
```

Set parameters for PI velocity control loop.

The function sets the factors of the velocity PI controller. More information are available in the Help of ApossIDE.

Parameters

<i>axis</i>	Axis module number
<i>velkprop</i>	Proportional factor of velocity controller (default 200)
<i>velkint</i>	Integral factor of velocity controller (default 5)
<i>velkilim</i>	Integral limit of velocity controller (default 1000)

Returns

- value: Always 1 in this function
- value > 0 Process successful
- value = 0 Process is active
- value < 0 Error

Examples

[Maxon_EC45_flat_1ax_BC_Enc.mc](#), [Maxon_EC45_flat_1ax_BC_Hall.mc](#), [Maxon_EC45_flat_1ax_SC_Hall_Inc.mc](#), [Maxon_ECi30_2ax_SC_Hall_Inc.mc](#), [Maxon_ECi40_1ax_SC_OL_Inc.mc](#), [Maxon_ECi40_3ax_SC_OL_Inc.mc](#), [Maxon_ECi52_1ax_SC_SSI.mc](#), [Maxon_RE_40_1ax_Inc.mc](#), [Maxon_RE_40_1ax_OL.mc](#), and [MotorCommissioning_MaxonECi30.mc](#).

Definition at line 315 of file SDK_Amplifier_MACS.mc.

```
316 {  
317     // parameters for velocity controller  
318     HWAMP_PARAM(axis, HWAMP_VELKPROP) = velkprop;    // velocity-controller proportional factor:  
+/-32767 (VELKPROP)  
319     HWAMP_PARAM(axis, HWAMP_VELKINT) = velkint;      // velocity-controller integral factor:  
+/-32767 (VELKINT)  
320     HWAMP_PARAM(axis, HWAMP_VELKILIM) = velkilim;    // velocity-controller Integral limit:  
0 ... 32767 (VELKILIM)  
321  
322     return(1);  
323 }
```

5.3.2.10 sdkSetupVirtualI2T() long sdkSetupVirtualI2T (
long axis,
long nominalCur,
long thermalTime)

Function to generate a virtual I2T protection.

This function activates a virtual I2T protection. More information about this can be found in the ApossIDE help.

<i>axis</i>	Axis module number
<i>nominalCur</i>	Nominal current (max. continuous load current) in mA
<i>thermalTime</i>	Therm. Winding time constant in ms

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[Maxon_EC45_flat_1ax_BC_Enc.mc](#), [Maxon_EC45_flat_1ax_BC_Hall.mc](#), [Maxon_EC45_flat_1ax_SC_Hall.mc](#), [Maxon_ECi30_2ax_SC_Hall_Inc.mc](#), [Maxon_ECi52_1ax_SC_SSI.mc](#), [Maxon_RE_40_1ax_Inc.mc](#), [Maxon_RE_40_1ax_OL.mc](#), [MotorCommissioning_MaxonECi30.mc](#), [Stepper_XY_CL.mc](#), and [Stepper_XY_OL.mc](#).

Definition at line 336 of file SDK_Amplifier_MACS.mc.

```
337 {
338     // parameters for I2T protection
339     VIRTAMP_PARAM(axis,VIRTAMP_I2TLIMIT)      = (nominalCur*nominalCur/1000); // I*I*1000
340     VIRTAMP_PARAM(axis,VIRTAMP_I2TTIME) = thermalTime;
341
342     return(1);
343 }
```

5.4 SDK_Amplifier_MiniMACS6_DS402_Slave.mc File Reference

This file provides all the function used to setup a MiniMACS6 DS402 slave.

```
#include <SysDef.mh>
#include "SDK_Amplifier_MiniMACS6_DS402_Slave.mh"
```

Functions

- long [sdkMiniMACS6_SetupCanBusModule](#) (long axis, long busId, long pdoNumber, long operationMode)
Setup the Can bus module for an MiniMACS6 DS402 slave.
- long [sdkMiniMACS6_SetupCanVirtAmp](#) (long axis, long maxRpm, long operationMode)
Setup the virtual amplifier for an MiniMACS6 DS402 slave with Can bus.

File Documentation

- long [sdkMiniMACS6_SetupCanVirtCntin](#) (long axis, long operationMode)

Setup the virtual counter input for an MiniMACS6 DS402 slave with Can bus.

- long [sdkMiniMACS6_SetupCanSdoParam](#) (long busId, long pdonumber, long slaveAxisNo, long operationMode)

Setup the Sdo parameter for an MiniMACS6 DS402 slave.

- long [sdkMiniMACS6_AxisHomingStart](#) (long axis, long busId, long operationMode, long &homingState)

State machine function for performing a homing on an MiniMACS6 DS402 slave.

5.4.1 Detailed Description

This file provides all the function used to setup a MiniMACS6 DS402 slave.

The MiniMACS DS402 Slave can be operated in the different modes csp, csv and cst The amplifiers can be controlled via CAN bus. The motor, encoder and controller parameters are set directly on the MiniMACS6 DS402 Application.

Revision

225

5.4.2 Function Documentation

5.4.2.1 [sdkMiniMACS6_AxisHomingStart\(\)](#) long [sdkMiniMACS6_AxisHomingStart](#) (

```
    long axis,
    long busId,
    long operationMode,
    long & homingState )
```

State machine function for performing a homing on an MiniMACS6 DS402 slave.

This function sets the MiniMACS6 DS402 slave in homing mode and starts the homing defined on the MiniMACS6 DS402 slave. Before calling this function, all homing parameters must be configured on the MiniMACS6 DS402 slave. The function is not blocking. The return parameter must be checked.

Parameters

<i>axis</i>	Axis module number
<i>busId</i>	Bus ID of the connected slave
<i>operationMode</i>	Operation mod which should be set after homing.
<i>homingState</i>	Call-by-reference variable for the iteration of the different homing states. Must be initialized with 0.

Returns

value: Process value
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[CAN_4Ax_MiniMACS6_csp.mc](#), [CAN_4Ax_MiniMACS6_cst.mc](#), and [CAN_4Ax_MiniMACS6_csv.mc](#).

Definition at line 318 of file SDK_Amplifier_MiniMACS6_DS402_Slave.mc.

```
320 {
321     long time, displayMode;
322     long axOffset = 0x800;
323     time= Time()%1000;
324
325     switch (homingState) {
326     case 0:
327         print("sdkMiniMACS6_AxisHomingStart:  ",axis);
328         homingState = 1;
329         break;
330     case 1:
331         // x6060 Change operation mode to "6: Homing mode."
332         SdoWriten( busId, MINIMACS6_MODES_OF_OPERATION+axOffset*axis, 0, 1, MINIMACS6_OP_HMM);
333         print("...: DS402 Homing AxisNo:  ",axis," - Set mode of OP: 0x06");
334
335         Delay(1);
336         print("...: DS402 Homing AxisNo:  ",axis," - Enable Drive");
337         AxisControl(axis, ON);
338
339         homingState = 2;
340
341     case 2:
342         displayMode = SdoRead(busId, MINIMACS6_MODES_OF_OPERATION_DISPLAY+axOffset*axis, 0);
343         if(displayMode== MINIMACS6_OP_HMM)
344         {
345             print("...: DS402 Homing AxisNo:  ",axis," - Display mode of OP: ",displayMode);
346             homingState = 3;
347         }
348         else if(time==0)
349         {
350             printf("...: DS402 Homing AxisNo:  %ld - Waiting for OP mode display:
351 %lx\n",axis, VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS));
352             // This section must be called in a 1 ms loop because of the print function.
353             Delay(1);
354         }
355         break;
356     case 3:
357         // Bit 10 Target reached - Homing procedure is interrupted or not started
358         if(VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS).i[10]==1)
359         {
360             printf("...: DS402 Homing AxisNo:  %ld - Homing start signal Status:
361 %lx\n",axis,VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS));
362             VIRTAMP_PARAM(axis, VIRTAMP_CNTRLW_PWRONENP) = 0x1F;
363             VIRTAMP_PARAM(axis, VIRTAMP_CNTRLW_PWRONENN) = 0x1F;
364             homingState = 4;
365         }
366         // Bit 13 Homingerror - Homing error occurred
367         else if(VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS).i[13]==1)
368         {
369             printf("...: DS402 Homing AxisNo:  %ld - Homing Error - Status:      %lx\n",axis,
370 VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS));
371             return(-1);
372         }
373         else if(time==0)
```

File Documentation

```

372         {
373             printf("...: DS402 Homing AxisNo:  %ld - Waiting for ready bit:      %lx\n",axis,
VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS));
374             // This section must be called in a 1 ms loop because of the print function.
375             Delay(1);
376         }
377         break;
378
379     case 4:
380         // Homing procedure is completed successfully
381         if (VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS).i[12]==1)
382         {
383             printf("...: DS402 Homing AxisNo:  %ld - Homing done - Status:      %lx\n",axis,
VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS));
384
385             VIRTAMP_PARAM(axis, VIRTAMP_CNTRLW_PWRONENP) = 0x0F;
386             VIRTAMP_PARAM(axis, VIRTAMP_CNTRLW_PWRONENN) = 0x0F;
387
388             print("...: DS402 Homing AxisNo:  ",axis," - Disable Drive");
389             AxisControl(axis, OFF);
390
391             SdoWriten(busId, MINIMACS6_MODES_OF_OPERATION+axOffset*axis, 0, 1, operationMode
);
            // 0x6060 Change operation mode to "8:  CSP mode."
392
393             homingState = 5;
394             break;
395         }
396         // Bit 13 Homingerror - Homing error occurred
397         else if (VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS).i[13]==1)
398         {
399             printf("...: DS402 Homing AxisNo:  %ld - Homing Error - Status:      %lx\n",axis,
VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS));
400             return(-1);
401         }
402         // Homing procedure is in progress
403         else if (time==0)
404         {
405             printf("...: DS402 Homing AxisNo:  %ld - Homing in progress - Status:
%lx\n",axis, VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS));
406             // This section must be called in a 1 ms loop because of the print function.
407             Delay(1);
408         }
409         break;
410
411     case 5:
412         displayMode = SdoRead(busId, MINIMACS6_MODES_OF_OPERATION_DISPLAY+axOffset*axis, 0);
413         if (displayMode== operationMode)
414         {
415             print("...: DS402 Homing AxisNo:  ",axis," - Display mode of OP: ",displayMode);
416             homingState = 6;
417         }
418         else if (time==0)
419         {
420             printf("...: DS402 Homing AxisNo:  %ld - Waiting for OP mode display:
%lx\n",axis, VIRTAMP_PROCESS(axis,PO_VIRTAMP_STATUS));
421             // This section must be called in a 1 ms loop because of the print function.
422             Delay(1);
423         }
424     case 6:
425         print("...: DS402 Homing AxisNo:  ",axis," - Finished");
426         homingState = 7;
427         return(1);
428     case 7:
429         return(1);
430     default :
431         print("...: DS402 Homing AxisNo:  ",axis," - Incorrect hominState init value: ",
homingState);
432         return(-1);
433     }
434

```


File Documentation

```

44
45     BUSMOD_PARAM(busmod, BUSMOD_MODE) = BUSMOD_MODE_DEACTIVATE;           // Deactivate (objects will
be deleted)
46     BUSMOD_PARAM(busmod, BUSMOD_BUSTYPE) = BUSMOD_BUSTYPE_CAN;           // Can bus
47
48
49     BUSMOD_PARAM(busmod, BUSMOD_BUSNO) = busId / 100000;                 // Bus number (0 = master,
1 = slave CAN bus)
50     BUSMOD_PARAM(busmod, BUSMOD_ID) = busId % 1000;                     // CAN Id for this bus
module
51     BUSMOD_PARAM(busmod, BUSMOD_SYNC) = 1;                               // Sync active
52     BUSMOD_PARAM(busmod, BUSMOD_GUARDTIME) = 0;                         // no guarding
53     BUSMOD_PARAM(busmod, BUSMOD_MODE) = BUSMOD_MODE_ACTIVATE;           // Create bus module
54
55     BUSMOD_PARAM(busmod, BUSMOD_PISRC_INPUT1) = VIRTAMP_PROCESS_SRCINDEX(axis, PO_VIRTAMP_CMDWORD);
// Select the input value sources object (send to bus): CMD Word
56
57     if (operationMode == MINIMACS6_OP_CSP) // cycle synchronous position (csp) mode
58     {
59         BUSMOD_PARAM(busmod, BUSMOD_PISRC_INPUT2) = VIRTAMP_PROCESS_SRCINDEX(axis, PO_VIRTAMP_REFPOS);
// Select the input value sources object (send to bus): position setpoint
60         print("...: cycle synchronous position (csp) mode");
61     }
62     else if (operationMode == MINIMACS6_OP_CSV) // cycle synchronous velocity (csv) mode
63     {
64         BUSMOD_PARAM(busmod, BUSMOD_PISRC_INPUT2) = VIRTAMP_PROCESS_SRCINDEX(axis, PO_VIRTAMP_REFVEL);
// Select the input value sources object (send to bus): velocity setvel
65         print("...: cycle synchronous velocity (csv) mode");
66     }
67     else if (operationMode == MINIMACS6_OP_CST) // cycle synchronous torque (cst) mode
68     {
69         BUSMOD_PARAM(busmod, BUSMOD_PISRC_INPUT2) = AXE_PROCESS_SRCINDEX(axis, REG_USERREFCUR); //
Select the input value sources object (send to bus): target torque
70         print("...: cycle synchronous torque (cst) mode");
71     }
72     else // Error
73     {
74         print("...: not supported operation mode");
75         return(-1);
76     }
77
78     BUSMOD_PARAM(busmod, BUSMOD_TXMAP_INPUT1) = pdoNumber*0x01000000 + 2*0x00010000 + 0; // pdo ;
length in bytes; bytes offset control word
79
80     if (operationMode != MINIMACS6_OP_CST)
81         BUSMOD_PARAM(busmod, BUSMOD_TXMAP_INPUT2) = pdoNumber*0x01000000 + 4*0x00010000 + 2; // pdo
; length in bytes; bytes offset target position/ velocity
82     else // cycle synchronous torque (cst) mode
83         BUSMOD_PARAM(busmod, BUSMOD_TXMAP_INPUT2) = pdoNumber*0x01000000 + 2*0x00010000 + 2; // pdo
; length in bytes; bytes offset target torque
84
85     BUSMOD_PARAM(busmod, BUSMOD_RXMAP_POVALUE1) = pdoNumber*0x01000000 + 2*0x00010000 + 0; // pdo ;
length in bytes; bytes offset status word
86     BUSMOD_PARAM(busmod, BUSMOD_RXMAP_POVALUE2) = pdoNumber*0x01000000 + 4*0x00010000 + 2; // pdo ;
length in bytes; bytes offset position actual value
87     BUSMOD_PARAM(busmod, BUSMOD_RXMAP_POVALUE3) = signedFlag + pdoNumber*0x01000000 + 2*0x00010000 +
6; // pdo ; length in bytes; bytes offset torque actual value
88
89     BUSMOD_PARAM(busmod, BUSMOD_MODE) = BUSMOD_MODE_ACTIVATE_NOSTOP;     // Start
bus module
90
91     return(1);
92 }

```

5.4.2.3 sdkMiniMACS6_SetupCanSdoParam() long sdkMiniMACS6_SetupCanSdoParam (

long busId,

```
long pdonumber,  
long slaveAxisNo,  
long operationMode )
```

Setup the Sdo parameter for an MiniMACS6 DS402 slave.

This function sets up the Sdo parameter for an MiniMACS6 DS402 slave. It can be defined with which operation mode is used. Each axis is controlled via its own PDO. A maximum of 4 axes can be controlled.

Parameters

<i>busId</i>	Bus ID of the connected slave
<i>pdoNumber</i>	Used PDO number
<i>slaveAxisNo</i>	Axis module number of the slave (0-5)
<i>operationMode</i>	Definition of the operation mode 0x08: Cyclic synchronous position (csp) mode 0x09: Cyclic synchronous velocity (csv) mode n 0x0A: Cyclic synchronous torque (cst) mode

Returns

```
value: Process value  
value > 0 Process successful  
value = 0 Process is active  
value < 0 Error
```

Examples

[CAN_4Ax_MiniMACS6_csp.mc](#), [CAN_4Ax_MiniMACS6_cst.mc](#), and [CAN_4Ax_MiniMACS6_csv.mc](#).

Definition at line 230 of file SDK_Amplifier_MiniMACS6_DS402_Slave.mc.

```
232 {  
233     long nodeid, last_value;  
234     long axOffset = 0x8000000;  
235     nodeid = busId % 100000;  
236  
237     print("sdkMiniMACS6_SetupCanSdoParam:  ",nodeid);  
238  
239     // Set ms loop sync source to CAN PDO, This feature will sync the controllers loop to a bus  
    signal.  
240     SdoWrite(busId, 0x2203, 0x13, 1);  
241  
242     // reset Transmitttype  
243     SdoWrite(busId, MINIMACS6_TRANSMIT_PDO_1_PARAMETER+ pdonumber-1, 2, 255);    // TxPDO1  
    Transmitttype  
244  
245     // the pdos have to be disabled for configuring  
246     last_value = (SdoRead(busId, MINIMACS6_TRANSMIT_PDO_1_PARAMETER+ pdonumber-1, 1));  
247     SdoWrite(busId, MINIMACS6_TRANSMIT_PDO_1_PARAMETER+ pdonumber-1, 1, (last_value | 0x80000000)); //  
    disable pdo x  
248  
249     //the pdos have to be disabled for configuring  
250     last_value = (SdoRead(busId, MINIMACS6_RECEIVE_PDO_1_PARAMETER+pdonumber-1, 1));  
251     SdoWrite(busId, MINIMACS6_RECEIVE_PDO_1_PARAMETER+ pdonumber-1, 1, (last_value | 0x80000000)); //  
    disable pdo 1  
252  
253     // Now we setup the correct PDO transmission for Tx and Rx
```

File Documentation

```

254     SdoWrite(busId, (MINIMACS6_TRANSMIT_PDO_1_PARAMETER + pdonumber-1), 2, 1);           // TxPDO
Transmitttype SYNC
255     SdoWrite(busId, (MINIMACS6_RECEIVE_PDO_1_PARAMETER + pdonumber-1), 2, 1);           // RxPDO
Transmitttype SYNC
256
257     // config RX PDO
258     SdoWrite(busId, (MINIMACS6_RECEIVE_PDO_1_MAPPING + pdonumber-1), 0x00, 0);           // disable
259     SdoWrite(busId, (MINIMACS6_RECEIVE_PDO_1_MAPPING + pdonumber-1), 0x01,
0x60400010+axOffset*slaveAxisNo); // controlword
260
261     if (operationMode == MINIMACS6_OP_CSP) {
262         SdoWrite(busId, (MINIMACS6_RECEIVE_PDO_1_MAPPING + pdonumber-1), 2,
0x607A0020+axOffset*slaveAxisNo); // download pdo 0x1600 entry: position set point
263         SdoWrite(busId, MINIMACS6_MODES_OF_OPERATION, 0, MINIMACS6_OP_CSP);
// mode of operation CSP
264         print("...: cycle synchronous position (csp) mode");
265     }
266     else if (operationMode == MINIMACS6_OP_CSV ) {
267         SdoWrite(busId, (MINIMACS6_RECEIVE_PDO_1_MAPPING + pdonumber-1), 2,
0x60FF0020+axOffset*slaveAxisNo); // download pdo 0x1600 entry: target velocity 32bit
268         SdoWrite(busId, MINIMACS6_MODES_OF_OPERATION, 0, MINIMACS6_OP_CSV);
// mode of operation CSV
269         print("...: cycle synchronous velocity (csv) mode");
270     }
271     else if (operationMode == MINIMACS6_OP_CST) {
272         SdoWrite(busId, (MINIMACS6_RECEIVE_PDO_1_MAPPING + pdonumber-1), 2,
0x60710010+axOffset*slaveAxisNo); // download pdo 0x1600 entry: Target torque 16bit
273         SdoWrite(busId, MINIMACS6_MODES_OF_OPERATION, 0, MINIMACS6_OP_CST);
// mode of operation CST
274         print("...: cycle synchronous torque (cst) mode");
275     }
276     else { // Error
277         print("...: not supported operation mode");
278         return(-1);
279     }
280
281     SdoWrite(busId, (MINIMACS6_RECEIVE_PDO_1_MAPPING + pdonumber - 1), 0x00, 2);           // enable
282
283     // config TX PDO
284     SdoWrite(busId, (MINIMACS6_TRANSMIT_PDO_1_MAPPING + pdonumber - 1), 0x00, 0);           // disable
285     SdoWrite(busId, (MINIMACS6_TRANSMIT_PDO_1_MAPPING + pdonumber - 1), 0x01,
0x60410010+axOffset*slaveAxisNo); // statusword
286     SdoWrite(busId, (MINIMACS6_TRANSMIT_PDO_1_MAPPING + pdonumber - 1), 0x02,
0x60640020+axOffset*slaveAxisNo); // actpos
287     SdoWrite(busId, (MINIMACS6_TRANSMIT_PDO_1_MAPPING + pdonumber - 1), 0x03,
0x60770010+axOffset*slaveAxisNo); // Torque actual
288     SdoWrite(busId, (MINIMACS6_TRANSMIT_PDO_1_MAPPING + pdonumber - 1), 0x00, 3 );           // enable
289
290     // fill in the wanted pdo
291     SdoWrite(busId, (MINIMACS6_TRANSMIT_PDO_1_PARAMETER + pdonumber - 1), 1, (0x80000180 + (pdonumber
- 1)*0x100 + nodeid) );// fill in pdo
292     SdoWrite(busId, (MINIMACS6_RECEIVE_PDO_1_PARAMETER + pdonumber - 1), 1, (0x80000200 + (pdonumber -
1)*0x100 + nodeid) );// fill in pdo
293
294     // enable
295     SdoWrite(busId, (MINIMACS6_TRANSMIT_PDO_1_PARAMETER + pdonumber - 1), 1, (0x00000180 + (pdonumber
- 1)*0x100 + nodeid) );// enable pdo
296     SdoWrite(busId, (MINIMACS6_RECEIVE_PDO_1_PARAMETER + pdonumber - 1), 1, (0x00000200 + (pdonumber -
1)*0x100 + nodeid) );// enable pdo
297
298     return(1);
299 }

```

5.4.2.4 sdkMiniMACS6_SetupCanVirtAmp() long sdkMiniMACS6_SetupCanVirtAmp (

long axis,

```
long maxRpm,
long operationMode )
```

Setup the virtual amplifier for an MiniMACS6 DS402 slave with Can bus.

This function sets up the virtual amplifier with Can bus for an MiniMACS6 DS402 slave. It can be defined with which operation mode is used.

Parameters

<i>axis</i>	Axis module number
<i>maxRpm</i>	Definition of the maximum motor speed [rpm]
<i>operationMode</i>	Definition of the operation mode 0x08: Cyclic synchronous position (csp) mode 0x09: Cyclic synchronous velocity (csv) mode n 0x0A: Cyclic synchronous torque (cst) mode

Note

The value passed from "PO_BUSMOD_VALUE3" to "VIRTAMP_PISRC_CURRENT" is the current torque. The value is given in per thousand of "Motor rated torque".

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[CAN_4Ax_MiniMACS6_csp.mc](#), [CAN_4Ax_MiniMACS6_cst.mc](#), and [CAN_4Ax_MiniMACS6_csv.mc](#).

Definition at line 112 of file SDK_Amplifier_MiniMACS6_DS402_Slave.mc.

```
113 {
114     print("sdkMiniMACS6_SetupCanVirtAmp: ",axis );
115
116     // virtual amplifiers have a fixed connection to axes number, axes 0 uses amplifier 0.
117     VIRTAMP_PARAM(axis,VIRTAMP_PISRC_CMDWORD)      = AXE_PROCESS_SRCINDEX(axis,REG_CNTRLWORD);
118     VIRTAMP_PARAM(axis,VIRTAMP_PISRC_REFPOS)        = AXE_PROCESS_SRCINDEX(axis,REG_COMPOS);
119     VIRTAMP_PARAM(axis,VIRTAMP_PISRC_REFVEL)        = AXE_PROCESS_SRCINDEX(axis,REG_REFERENCE);
120     VIRTAMP_PARAM(axis,VIRTAMP_PISRC_REFACC)        = AXE_PROCESS_SRCINDEX(axis,PID_FFACCPART);
121     VIRTAMP_PARAM(axis,VIRTAMP_PISRC_STATUS)        = BUSMOD_PROCESS_SRCINDEX(axis,PO_BUSMOD_VALUE1);
122     //statusword
123     VIRTAMP_PARAM(axis,VIRTAMP_PISRC_CURRENT)       = BUSMOD_PROCESS_SRCINDEX(axis, PO_BUSMOD_VALUE3);
124     //torque
125
126     VIRTAMP_PARAM(axis,VIRTAMP_CNTRLW_PWROFF)       = 0x06;
127     VIRTAMP_PARAM(axis,VIRTAMP_CNTRLW_PWRONDIS)     = 0x06;
128     VIRTAMP_PARAM(axis,VIRTAMP_CNTRLW_PWRONENP)     = 0x0F;
129     VIRTAMP_PARAM(axis,VIRTAMP_CNTRLW_PWRONENN)     = 0x0F;
130     VIRTAMP_PARAM(axis,VIRTAMP_CNTRLW_QUICKSTOP)    = 0x02;
131     VIRTAMP_PARAM(axis,VIRTAMP_CNTRLW_RESET)        = 0x80;
132
133     VIRTAMP_PARAM(axis,VIRTAMP_STOPDELAY)           = 0x00;
134     VIRTAMP_PARAM(axis,VIRTAMP_ERROR_BITMASK)       = 0x08;
```

File Documentation

```

133     VIRTAMP_PARAM(axis,VIRTAMP_ERROR_POLARITY)    = 1;
134
135     // Ready bit: Ready to switch on, Switched on, Operation enabled, Voltage enabled
136     VIRTAMP_PARAM(axis,VIRTAMP_READY_BITMASK)     = 0x17;
137     VIRTAMP_PARAM(axis,VIRTAMP_READY_POLARITY)    = 1;
138
139     if (operationMode == MINIMACS6_OP_CSP) // cycle synchronous position (csp) mode
140     {
141         VIRTAMP_PARAM(axis,VIRTAMP_REF100PERC)     = 0; // not used in csp
142         VIRTAMP_PARAM(axis,VIRTAMP_REFLIMIT)       = 0; // not used in csp
143         print("...: cycle synchronous position (csp) mode");
144     }
145     else if (operationMode == MINIMACS6_OP_CSV) // cycle synchronous velocity (csv) mode
146     {
147         VIRTAMP_PARAM(axis,VIRTAMP_REF100PERC)     = maxRpm;
148         VIRTAMP_PARAM(axis,VIRTAMP_REFLIMIT)       = 2* maxRpm;
149         print("...: cycle synchronous velocity (csv) mode");
150     }
151     else if (operationMode == MINIMACS6_OP_CST) // cycle synchronous torque (cst) mode
152     {
153         VIRTAMP_PARAM(axis,VIRTAMP_REF100PERC)     = 0; // not used in cst
154         VIRTAMP_PARAM(axis,VIRTAMP_REFLIMIT)       = 0; // not used in cst
155         print("...: cycle synchronous torque (cst) mode");
156     }
157     else // Error
158     {
159         print("...: not supported operation mode");
160         return(-1);
161     }
162
163     VIRTAMP_PARAM(axis,VIRTAMP_MODE) = VIRTAMP_MODE_ENABLE; // has to be the last one because it
    activates all
164
165     return(1);
166 }

```

5.4.2.5 sdkMiniMACS6_SetupCanVirtCntin() long sdkMiniMACS6_SetupCanVirtCntin (

long axis,

long operationMode)

Setup the virtual counter input for an MiniMACS6 DS402 slave with Can bus.

This function sets up the virtual counter input for an MiniMACS6 DS402 slave. It can be defined with which operation mode is used.

Parameters

<i>axis</i>	Axis module number
<i>operationMode</i>	Definition of the operation mode 0x08: Cyclic synchronous position (csp) mode 0x09: Cyclic synchronous velocity (csv) mode n 0x0A: Cyclic synchronous torque (cst) mode

Returns

value: Process value
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[CAN_4Ax_MiniMACS6_csp.mc](#), [CAN_4Ax_MiniMACS6_cst.mc](#), and [CAN_4Ax_MiniMACS6_csv.mc](#).

Definition at line 183 of file SDK_Amplifier_MiniMACS6_DS402_Slave.mc.

```
184 {
185     print("sdkMiniMACS6_SetupCanVirtCntin: ",axis);
186
187     VIRTCONTIN_PARAM(axis,VIRTCONTIN_PISRC_COUNTER) = BUSMOD_PROCESS_SRCINDEX(axis, PO_BUSMOD_VALUE2);
188
189     if (operationMode == MINIMACS6_OP_CSP) // cycle synchronous position (csp) mode
190     {
191         VIRTCONTIN_PARAM(axis,VIRTCONTIN_MODE) = VIRTCONTIN_MODE_ABSOLUTE_DIRECT; // Source is
absolute and is taken as it is
192         print("...: cycle synchronous position (csp) mode");
193     }
194     else if (operationMode == MINIMACS6_OP_CSV) // cycle synchronous velocity (csv) mode
195     {
196         VIRTCONTIN_PARAM(axis,VIRTCONTIN_MODE) = VIRTCONTIN_MODE_ABSOLUTE; // Source is a
position value and difference to last value is added
197         print("...: cycle synchronous velocity (csv) mode");
198     }
199     else if (operationMode == MINIMACS6_OP_CST) // cycle synchronous torque (cst) mode
200     {
201         VIRTCONTIN_PARAM(axis,VIRTCONTIN_MODE) = VIRTCONTIN_MODE_ABSOLUTE_DIRECT; // Source is
absolute and is taken as it is
202         print("...: cycle synchronous torque (cst) mode");
203     }
204     else // Error
205     {
206         print("...: not supported operation mode");
207         return(-1);
208     }
209
210     return(1);
211 }
```

5.5 SDK_Amplifier_MotorAlignment.mc File Reference

Functions for aligning a motor.

```
#include <SysDef.mh>
#include "SDK_Amplifier_MotorAlignment.mh"
```

Functions

- long [sdkMotorAlignment](#) (long axis, long mode, long maxCurrent, long alignCurrent)

File Documentation

The purpose of this function is to find the rotor position of a brushless motor in relation to the encoder position feedback.

- long `sdkMotorMultiAlignment` (long axis, long axisNumber, long mode, long maxCurrent, long align↔Current)

The purpose of this function is to find the rotor position of a brushless motor in relation to the encoder position feedback.

- long `sdkSetMotorAlignmentOffset` (long axis, long elPol, long poseOffset)

This function sets the offset of the electric field in relation to an absolute encoder.

5.5.1 Detailed Description

Functions for aligning a motor.

Revision

192

5.5.2 Function Documentation

```
5.5.2.1 sdkMotorAlignment() long sdkMotorAlignment (
    long axis,
    long mode,
    long maxCurrent,
    long alignCurrent )
```

The purpose of this function is to find the rotor position of a brushless motor in relation to the encoder position feedback.

The function will start a program in the amplifier, which will apply a magnetic field to the rotor. The rotor will then align to this field. The field will then be varied a bit back and forth to commit the measurement. To use this function the maximum current of the motor must also be specified.

Note

In order for this procedure to work, the motor must not be braked or be in a limit position. The rotor must be able to rotate freely for about half a revolution.

Parameters

<i>axis</i>	Axis module number
<i>mode</i>	Mode of alignment 1: For brushless motors 3: For stepper motors
<i>maxCurrent</i>	Maximum current of the motor
<i>alignCurrent</i>	Current for the alignment function

Returns

value: Command Reference, MotorAlignStatus()
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[Maxon_ECi40_1ax_SC_OL_Inc.mc](#), [Maxon_ECi52_1ax_SC_SSI.mc](#), and [Stepper_XY_CL.mc](#).

Definition at line 34 of file SDK_Amplifier_MotorAlignment.mc.

```

35 {
36     long retVal;
37
38     HWAMP_PARAM((axis), HWAMP_MAXCUR) = alignCurrent;      // Set the maximum current for aligning the
motor
39
40     print("AxisNo ", axis, " Start position detection");
41     MotorAlignStart(axis, mode);                            // Start motor alignment
42
43     while((retVal = MotorAlignStatus(axis)) == 0)           // Wait until the alignment is successful
or failed
44     {
45         print("Aligning... ", retVal);
46         Delay(500);
47     }
48     if(retVal < 0)
49     {
50         print("!!!! Rotor position detection FAILED !!!! retVal: ", retVal);
51     }
52     else if (retVal > 0)
53     {
54         print("Alignment completed with ", retVal);
55     }
56     HWAMP_PARAM((axis), HWAMP_MAXCUR) = maxCurrent;        // Set the maximum current for the motor
57
58     return(retVal);
59 }
```

5.5.2.2 sdkMotorMultiAlignment() long sdkMotorMultiAlignment (

long axis,

long axisNumber,

long mode,

long maxCurrent,

long alignCurrent)

The purpose of this function is to find the rotor position of a brushless motor in relation to the encoder position feedback.

The function is the same as function [sdkMotorAlignment\(\)](#) - but several motors can be started in simultaneous. The same current is set for each axis.

Note

In order for this procedure to work, the motor must not be braked or be in a limit position. The rotor must be able to rotate freely for about half a revolution.

<i>axis</i>	Axis module number (lowest axis)
<i>axisNumber</i>	Number axes. These must be lined up and increased by one for each axis
<i>mode</i>	Mode of alignment 1: For brushless motors 3: For stepper motors
<i>maxCurrent</i>	Maximum current of the motor
<i>alignCurrent</i>	Current for the alignment function

Returns

value: Command Reference, MotorAlingStatus()
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[Maxon_ECi40_3ax_SC_OL_Inc.mc.](#)

Definition at line 79 of file SDK_Amplifier_MotorAlignment.mc.

```
80 {
81     long retVal[6]={0,0,0,0,0,0},retValCheck[6]={0,0,0,0,0,0}, retValSum=0, i;
82
83     axisNumber=axis+axisNumber;
84
85     for (i=axis;i<axisNumber;i++)
86     {
87         AxisControl(i,OFF);
88         HWAMP_PARAM(i, HWAMP_MAXCUR) = alignCurrent;    // Set the maximum current for aligning the
motor
89         print("AxisNo ", i, " Start position detection");
90         MotorAlignStart(i, mode);
91         Delay(250);
92     }
93     print("AxisNumber ", axisNumber - axis);
94
95     while(!(retValSum==(axisNumber-axis)))
96     {
97         retValSum=0;
98         for (i=axis;i<axisNumber;i++)
99         {
100             retVal[i] = MotorAlignStatus(i);
101             printf("Aligning Axis:  %ld, Status:  %ld", i, retVal[i]);
102             print("");
103
104             if(retVal[i] < 0)
105             {
106                 print("!!!! Rotor position detection FAILED !!!!  retVal:  ", retVal[i]);
107                 print("");
108                 return(-1);
109             }
110             else if (retVal[i] > 0)
111             {
112                 printf("Aligning Axis:  %ld completed with:  %ld", i, retVal[i]);
113                 print("");
114                 retValCheck[i]=1;
115             }
116             retValSum=retValCheck[i]+retValSum;
```

```
117         }
118         Delay(1000);
119     }
120     for (i=axis;i<=axisNumber;i++)
121     {
122         HWAMP_PARAM((i), HWAMP_MAXCUR) = maxCurrent;           // Set the maximum current for the motor
123     }
124
125     return(1);
126 }
```

5.5.2.3 sdkSetMotorAlignmentOffset() long sdkSetMotorAlignmentOffset (

 long axis,

 long elPol,

 long poselOffset)

This function sets the offset of the electric field in relation to an absolute encoder.

If an absolute encoder is used and the offset between rotor/ encoder is already known, the classic motor alignment can be dispensed with. The offset and the polarity can be determined with the function [sdkMotorAlignment\(\)](#) during commissioning.

Parameters

<i>axis</i>	Axis module number
<i>elPol</i>	Encoder polarity vs. electrical polarity (default -1) 1 HWAMP_ELPOL_REGULAR: Electrical polarity is equal encoder's polarity -1 HWAMP_ELPOL_INVERS: Electrical polarity is inverse to encoder's polarity
<i>poselOffset</i>	Sets the offsets for the aligned absolute encoders.

Returns

- value: Always 1 in this function
- value > 0 Process successful
- value = 0 Process is active
- value < 0 Error

Examples

[Maxon_ECi52_1ax_SC_SSI.mc.](#)

Definition at line 143 of file SDK_Amplifier_MotorAlignment.mc.

```
144 {
145     HWAMP_PARAM(axis,HWAMP_ELPOL)           = elPol;
146     HWAMP_PARAM(axis, HWAMP_POSEL_OFFSET)    = poselOffset;
147
148     return(1);
149 }
```

Functions for commissioning a motor.

```
#include <SysDef.mh>
#include "SDK_Amplifier_MotorCommissioning.mh"
```

Functions

- long [sdkMotorCommCurrentStep](#) (long axis, long current, long time, long recordEnable)

Current step for setting the current controller.

- long [sdkMotorCommVelStep](#) (long axis, long velocity, long time, long recordEnable)

Velocity step for setting the velocity controller.

- long [sdkMotorCommPositionRamp](#) (long axis, long distance, long controlMode, long recordEnable)

Position ramp for setting the position controller.

- long [sdkMotorCommHallSignalCheck](#) (long axis)

Checking the hall signals.

- long [sdkMotorCommEncoderUserUnitCheck](#) (long axis)

Checking the configured encoders and the user units.

5.6.1 Detailed Description

Functions for commissioning a motor.

Revision

241

5.6.2 Function Documentation

```
5.6.2.1 sdkMotorCommCurrentStep() long sdkMotorCommCurrentStep (
    long axis,
    long current,
    long time,
    long recordEnable )
```

Current step for setting the current controller.

This function can be used to do a current step and record it. The controller parameters can then be adjusted until the desired behaviour of the current controller is achieved. The axis must be correctly parameterised before calling the function. With the help of this function, the attached example and the utility monitor file `ut_Control_Loop_Parameter_Slider.zbm` the controller can be adjusted. More information about the controller setting can be found in the ApossIDE Help.

Note

To adjust the current controller, the motor does not need to move. A step which only takes several ms is enough.

Parameters

<i>axis</i>	Axis module number
<i>current</i>	Current for the step in mA
<i>time</i>	Duration of the current step in ms
<i>recordEnable</i>	Definition record oscilloscope (Only available on MACS6) 0 Record Disable 1 Record Enable

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[MotorCommissioning_MaxonECi30.mc](#).

Definition at line 33 of file `SDK_Amplifier_MotorCommissioning.mc`.

```
34 {
35     if (recordEnable==1)    // Settings for the recording
36     {
37         RecordIndex(    HWAMP_PROCESS_INDEX(axis,PO_HWAMP_CURRENT),
38                         AXE_PROCESS_INDEX(axis,REG_USERREFCUR));
39         RecordTimeBase(RECORD_LOOP_CUR);
40         RecordDest(DYNMEM);
41         RecordStart(0);
42     }
43
44     // Set the controller loop
45     HWAMP_PARAM(axis, HWAMP_MODE)    = HWAMP_MODE_POS_CUR;
46 }
```

File Documentation

```

47 // Parameters for position controller -> Make sure that the position controller is not running
48 AXE_PARAM(axis, KPROP) = 0; // position-controller proportional factor
49 AXE_PARAM(axis, KINT) = 0; // position-controller integral factor
50 AXE_PARAM(axis, KDER) = 0; // position-controller differencial factor
51 AXE_PARAM(axis, FFVEL) = 0; // position-controller Velocity Feed forward
52 AXE_PARAM(axis, KFFACC) = 0; // position-controller Acceleration Feed Forward
53 AXE_PARAM(axis, KFFDEC) = 0; // position-controller Deceleration Feed Forward
54 AXE_PARAM(axis, POSERR) = 0x7FFFFFFF; // very high to avoid errors
55
56 AxisControl(axis, USERREFCUR); // Turn motor control on with a user defined reference current
57 print("Start current step");
58 Delay(50);
59
60 AXE_PROCESS(axis, REG_USERREFCUR)=current; //Current Step to "current" mA
61 Delay(time); //Wait for "time" ms
62 AXE_PROCESS(axis, REG_USERREFCUR)=0; //Set current to 0 mA
63
64 Delay(50);
65 print("Stop current step");
66 RecordStop(0,0);
67
68 return(1);
69 }

```

5.6.2.2 sdkMotorCommEncoderUserUnitCheck() long sdkMotorCommEncoderUserUnitCheck (long axis)

Checking the configured encoders and the user units.

With this function the number of quadcounts, counts per turn and user units can be compared. The encoder must first be correctly configured. The values are printed in the APosSIDE.

Parameters

axis	Axis module number
------	--------------------

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[MotorCommissioning_MaxonECi30.mc.](#)

Definition at line 218 of file SDK_Amplifier_MotorCommissioning.mc.

```

219 {
220 // Turn off the motor
221 AxisControl(axis, OFF);
222 print("Counts per turn [cpt]: ", AXE_PROCESS(axis, REG_ACTPOS)/4);
223 print("Quadcounts [qc] : ", AXE_PROCESS(axis, REG_ACTPOS));
224 print("User units [uu] : ", Apos(axis));
225 print("");

```

5.6.2.3 sdkMotorCommHallSignalCheck()

```
long sdkMotorCommHallSignalCheck (
    long axis )
```

With this function, the individual hall states and the position can be read out. The states are printed in the ApossiDE.

<i>axis</i>	Axis module number
-------------	--------------------

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

MotorCommissioning MaxonECi30.mc.

```

196 {
197     // Turn off the motor and set a hall commtype
198     AxisControl(axis, OFF);
199     HWAMP_PARAM(axis,HWAMP_COMMTYPE)=HWAMP_COMMTYPE_BLDC;
200
201     print("HALL Bit 0:  ", HWHALL_PROCESS(axis,PO_HWHALL_STATUS).i[0]);
202     print("HALL Bit 1:  ", HWHALL_PROCESS(axis,PO_HWHALL_STATUS).i[1]);
203     print("HALL Bit 2:  ", HWHALL_PROCESS(axis,PO_HWHALL_STATUS).i[2]);
204     print("HALL Position: ", HWHALL_PROCESS(axis,PO_HWHALL_POS));
205
206     return(1);
207 }

```

Position ramp for setting the position controller.

File Documentation

This function can be used to do a position ramp and record it. The controller parameters can then be adjusted until the desired behaviour of the position controller is achieved. The axis must be correctly parameterised before calling the function. With the help of this function, the attached example and the utility monitor file `ut_Control_Loop_Parameter_Slider.zbm` the controller can be adjusted. More information about the controller setting can be found in the ApossIDE Help.

Parameters

<i>axis</i>	Axis module number
<i>distance</i>	Relative distance in user units between start and end position
<i>controlMode</i>	Define control typ (default 3) 0 HWAMP_MODE_POS_VEL_CUR: Pos -> Vel -> Cur -> PWM 1 HWAMP_MODE_POS_VEL: Pos -> Vel -> PWM 2 HWAMP_MODE_POS_CUR: Pos -> Cur -> PWM 3 HWAMP_MODE_POS: Pos -> PWM
<i>recordEnable</i>	Definition record oscilloscope (Only available on MACS6) 0 Record Disable 1 Record Enable

Returns

value: Always 1 in this function
 value > 0 Process successful
 value = 0 Process is active
 value < 0 Error

Examples

[MotorCommissioning_MaxonECi30.mc](#).

Definition at line 154 of file `SDK_Amplifier_MotorCommissioning.mc`.

```

155 {
156     if (recordEnable==1)    // Settings for the recording
157     {
158         RecordIndex(    AXE_PROCESS_INDEX(axis,REG_COMPOS),
159                         AXE_PROCESS_INDEX(axis,REG_ACTPOS),
160                         AXE_PROCESS_INDEX(axis,REG_TRACKERR));
161         RecordDest(DYNMEM);
162         RecordStart(0);
163     }
164
165     // Set controller loop
166     HWAMP_PARAM(axis, HWAMP_MODE)    = controlMode;
167     AXE_PARAM(axis, POSERR)          = 0x7FFFFFFF;    // very high to avoid errors
168
169     AxisControl(axis, ON);            // Turn the motor on
170     print("Start position ramp");
171     Delay(50);
172
173     AxisPosRelStart(axis, distance);  // Start relative movement
174     AxisWaitReached(axis);           // Wait until target ist reached
175
176     Delay(50);
177     print("Stop position ramp");
178     RecordStop(0,0);
179
180     return(1);
181 }
```

```
5.6.2.5 sdkMotorCommVelStep() long sdkMotorCommVelStep (
    long axis,
    long velocity,
    long time,
    long recordEnable )
```

Velocity step for setting the velocity controller.

This function can be used to do a velocity step and record it. The controller parameters can then be adjusted until the desired behaviour of the current controller is achieved. The axis must be correctly parameterised before calling the function. With the help of this function, the attached example and the utility monitor file `ut_Control_Loop_Parameter_Slider.zbm` the controller can be adjusted. More information about the controller setting can be found in the ApossIDE Help.

Note

To adjust the velocity controller, the motor does need to move. The speed must be achievable in the selected time.

Parameters

<i>axis</i>	Axis module number
<i>velocity</i>	Velocity for the step, scaling is —0x4000 to 0x4000 for -100 to 100% of VELMAX
<i>time</i>	Duration of the velocity step in ms
<i>recordEnable</i>	Definition record oscilloscope (Only available on MACS6) 0 Record Disable 1 Record Enable

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[MotorCommissioning_MaxonECi30.mc](#).

Definition at line 91 of file SDK_Amplifier_MotorCommissioning.mc.

```
92 {
93     // Settings for the recording
94     if (recordEnable==1)
95     {
96         RecordIndex(    HWAMP_PROCESS_INDEX(axis,PO_HWAMP_VELPI_ACTUAL),
97                         HWAMP_PROCESS_INDEX(axis,PO_HWAMP_VELPI_REF));
98         RecordTimeBase(RECORD_LOOP_VEL);
99         RecordDest(DYNMEM);
100         RecordStart(0);
101     }
102
103     // Set the controller loop
104     HWAMP_PARAM(axis, HWAMP_MODE)      = HWAMP_MODE_POS_VEL_CUR;
```

File Documentation

```

105
106 // Parameters for position controller -> Make sure that the position controller is not running
107 AXE_PARAM(axis, KPROP) = 0; // position-controller proportional factor
108 AXE_PARAM(axis, KINT) = 0; // position-controller integral factor
109 AXE_PARAM(axis, KDER) = 0; // position-controller differential factor
110 AXE_PARAM(axis, FFVEL) = 0; // position-controller Velocity Feed forward
111 AXE_PARAM(axis, KFFACC) = 0; // position-controller Acceleration Feed Forward
112 AXE_PARAM(axis, KFFDEC) = 0; // position-controller Deceleration Feed Forward
113 AXE_PARAM(axis, POSERR) = 0x7FFFFFFF; // very high to avoid errors
114
115
116 AxisControl(axis, USERREFVEL); // Turn motor control on with a user defined reference current
117 print("Start velocity step");
118 Delay(50);
119
120 AXE_PROCESS(axis, REG_USERREFVEL)=velocity; // Velocity step to "velocity"
121 Delay(time); // Wait for "time" ms
122 AXE_PROCESS(axis, REG_USERREFVEL)=0; // Set velocity to 0
123
124 Delay(50);
125 print("Stop velocity step");
126 RecordStop(0,0);
127
128 return(1);
129 }

```

5.7 SDK_ApossC.mc File Reference

File to include the entire SDK.

```

#include "Amplifier\SDK_Amplifier_MACS.mc"
#include "Amplifier\SDK_Amplifier_MotorAlignment.mc"
#include "Amplifier\SDK_Amplifier_MotorCommissioning.mc"
#include "Amplifier\SDK_Amplifier_DS402_StateMachine.mc"
#include "Amplifier\SDK_Amplifier_Epos4.mc"
#include "Amplifier\SDK_Amplifier_MiniMACS6_DS402_Slave.mc"
#include "Axis\SDK_Axis_Setup.mc"
#include "BusSystem\SDK_Bussystem_EtherCat.mc"
#include "Communication\SDK_Communication_Ethernet.mc"
#include "Encoder\SDK_Encoder_Setup.mc"
#include "Information\SDK_Information_General.mc"
#include "Error\SDK_Error_Description.mc"
#include "Kinematics\SDK_Kinematics_Setup.mc"
#include "Miscellaneous\SDK_Miscellaneous_Recording.mc"
#include "Miscellaneous\SDK_Miscellaneous_IO.mc"
#include "Motion\SDK_Motion_Movement.mc"
#include "VirtualModule\SDK_VirtualModule_AxisSetup.mc"
#include "VirtualModule\SDK_VirtualModule_MasterSetup.mc"

```

5.7.1 Detailed Description

File to include the entire SDK.

This file can be included to include the entire SDK. The folder "SDK" must be on the same level as the used program. In order to keep the memory requirements of the SDK as low as possible, the compiler must be set to include only referenced files. The procedure for this is shown in detail in the section [HowToUse](#).

5.8 SDK_Axis_Setup.mc File Reference

Functions for the axis setup.

```
#include <SysDef.mh>
#include "SDK_Axis_Setup.mh"
```

Functions

- long [sdkSetupPositionPIDControl](#) (long axis, long kprop, long kint, long kder)

Set parameters for PID position control loop.

- long [sdkSetupPositionPIDControlExt](#) (long axis, long kprop, long kint, long kder, long kilim, long kilimtime, long bandwidth, long ffvel, long kffacc, long kffdec)

Set extended parameters for PID position control loop.

- long [sdkSetupAxisUserUnits](#) (long axis, long posencrev, long posencqc, long posfact_z, long posfact_n, long feedrev, long feeddist)

Setup to convert the resolution of the machine in user units.

- long [sdkSetupAxisMovementParam](#) (long axis, long velres, long maxRpm, long ramptype, long rampmin, long jerkmin, long poserr)

Defines parameters which are required for the calculation of ramps and velocities.

- long [sdkSetupAxisJerkLimited](#) (long axis, long jerkmin1, long jerkmin2, long jerkmin3, long jerkmin4)

Definition of the jerk limited extended parameter.

- long [sdkSetupAxisDirection](#) (long axis, long posdrct)

Definition of the rotation direction of the axis.

- long [skdSetupAxisSoftwareLimit](#) (long axis, long negLimitUu, long posLimitUu)

Setting up a software axis limitation in user units.

- long [skdEnableAxisSoftwareLimit](#) (long axis, long active)

Activation of the axis software limits.

5.8.1 Detailed Description

Functions for the axis setup.

Revision

271

5.8.2.1 sdkSetupAxisDirection() `long sdkSetupAxisDirection (`
`long axis,`
`long posdrct)`

Definition of the rotation direction of the axis.

The application defines what is a forward and backward movement of the motor. With the help of this function this definition can be adjusted.

Parameters

<i>axis</i>	Axis module number
<i>posdrct</i>	Parameter to set the rotation direction (default 0) 1 Normal direction -1 Invert direction

Returns

value: Process value
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[Maxon_EC45_flat_1ax_BC_Enc.mc](#), [Maxon_EC45_flat_1ax_BC_Hall.mc](#), [Maxon_EC45_flat_1ax_SC_Hall_Inc.](#),
[Maxon_ECi30_2ax_SC_Hall_Inc.mc](#), [Maxon_ECi40_1ax_SC_OL_Inc.mc](#), [Maxon_ECi40_3ax_SC_OL_Inc.mc](#),
[Maxon_ECi52_1ax_SC_SSI.mc](#), [Maxon_RE_40_1ax_Inc.mc](#), [Maxon_RE_40_1ax_OL.mc](#), [MotorCommissioning](#)
[Stepper_XY_CL.mc](#), and [Stepper_XY_OL.mc](#).

Definition at line 186 of file SDK_Axis_Setup.mc.

```
187 {
188     // Set rotation direction
189     if (posdrct==1 || posdrct==-1 )
190     {
191         VIRTAMP_PARAM(axis,VIRTAMP_INVERT)=posdrct;
192         VIRCOUNTIN_PARAM(axis,VIRCNTIN_UUFACT_UNITNO)=posdrct;
193     }
194     else    // Error
195     {
196         print("SDK Error:  not supported rotation direction");
197         return (-1);
198     }
199
200     return (1);
201 }
```

5.8.2.2 sdkSetupAxisJerkLimited() long sdkSetupAxisJerkLimited (

long axis,

long jerkmin1,

long jerkmin2,

long jerkmin3,

long jerkmin4)

Definition of the jerk limited extended parameter.

Function to set the extended jerk limited parameters. This topic is described in the ApossIDE Help on the page "Limited Jerk". If the parameters JERKMIN2 - 4 are set to the value 0, the jerk limit is taken from the parameter JERKMIN.

Parameters

<i>axis</i>	Axis module number
<i>jerkmin1</i>	Defines the minimum time [ms] required before reaching the maximum acceleration (default 100)
<i>jerkmin2</i>	Defines the minimum time [ms] required to ramp the acceleration down from maximum acceleration to 0 (default 0)
<i>jerkmin3</i>	Defines the minimum time [ms] required to ramp the deceleration up from 0 to maximum deceleration (default 0)
<i>jerkmin4</i>	Defines the minimum time [ms] required to ramp the deceleration down from maximum deceleration to 0 (default 0)

Returns

value: Always 1 in this function

value > 0 Process successful

value = 0 Process is active

value < 0 Error

Definition at line 162 of file SDK_Axis_Setup.mc.

```
163 {
164     AXE_PARAM(axis, RAMPTYPE) = RAMPTYPE_JERKLIMITED;      // Ramp type: Jerk limited
165     AXE_PARAM(axis, JERKMIN)  = jerkmin1;
166     AXE_PARAM(axis, JERKMIN2) = jerkmin2;
167     AXE_PARAM(axis, JERKMIN3) = jerkmin3;
168     AXE_PARAM(axis, JERKMIN4) = jerkmin4;
169
170     return(1);
171 }
```

5.8.2.3 sdkSetupAxisMovementParam() long sdkSetupAxisMovementParam (

long axis,

long velres,

long maxRpm,

long ramptype,

File Documentation

```
long rampmin,
long jerkmin,
long poserr )
```

Defines parameters which are required for the calculation of ramps and velocities.

The function sets all basic parameters which are needed for the calculation of the ramps and velocities.

Parameters

<i>axis</i>	Axis module number
<i>velres</i>	Velocity resolution, Scaling used for the velocity and acceleration/deceleration commands (default 100)
<i>maxRpm</i>	Defines the rated speed of the drive. This value is listed in RPM (default 1000)
<i>ramptype</i>	Defines the ramptype (default 0) 0 RAMPTYPE_TRAPEZ: Trapezoid ramp 2 RAMPTYPE_JERKLIMITED: Jerk limited ramp
<i>rampmin</i>	Maximum acceleration (default 1000)
<i>jerkmin</i>	Minimum time (ms) required before reaching the maximum acceleration (default 100) just usefull for ramptype "RAMPTYPE_JERKLIMITED"
<i>poserr</i>	The Maximum Tolerated Position Error POSERR defines the tolerance allowed between the current actual position and the calculated command position (default 20000)

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[CAN_1Ax_EPOS4-Test_csp.mc](#), [CAN_1Ax_EPOS4-Test_cst.mc](#), [CAN_1Ax_EPOS4-Test_csv.mc](#),
[CAN_2Ax_EPOS4-Test_csp.mc](#), [CAN_4Ax_MiniMACS6_csp.mc](#), [CAN_4Ax_MiniMACS6_cst.mc](#),
[CAN_4Ax_MiniMACS6_csv.mc](#), [ECAT_1Ax_EPOS4-Test_cst.mc](#), [ECAT_3Ax_Epos4-Test_csp.mc](#),
[Maxon_EC45_flat_1ax_BC_Enc.mc](#), [Maxon_EC45_flat_1ax_BC_Hall.mc](#), [Maxon_EC45_flat_1ax_SC_Hall_Inc.](#)
[Maxon_ECi30_2ax_SC_Hall_Inc.mc](#), [Maxon_ECi40_1ax_SC_OL_Inc.mc](#), [Maxon_ECi40_3ax_SC_OL_Inc.mc](#),
[Maxon_ECi52_1ax_SC_SSI.mc](#), [Maxon_RE_40_1ax_Inc.mc](#), [Maxon_RE_40_1ax_OL.mc](#), [MotorCommissioning](#)
[Stepper_XY_CL.mc](#), and [Stepper_XY_OL.mc](#).

Definition at line 136 of file SDK_Axis_Setup.mc.

```
137 {
138     AXE_PARAM(axis, VELRES)      = velres;    // Velocity resolution
139     AXE_PARAM(axis, VELMAX)      = maxRpm;    // Rated speed of drive
140     AXE_PARAM(axis, RAMPTYPE)    = ramptype;  // Ramp type
141     AXE_PARAM(axis, RAMPMIN)     = rampmin;   // Maximum acceleration
142     AXE_PARAM(axis, JERKMIN)     = jerkmin;   // Minimum time required before reaching the maximum
acceleration
143     AXE_PARAM(axis, POSERR)      = poserr;    // Maximum tolerated position error
144
145     return(1);
146 }
```

```
5.8.2.4 sdkSetupAxisUserUnits() long sdkSetupAxisUserUnits (
    long axis,
    long posencrev,
    long posencqc,
    long posfact_z,
    long posfact_n,
    long feedrev,
    long feeddist )
```

Setup to convert the resolution of the machine in user units.

The basic resolution of the entire machine is 1 encoder quadcount. Using the above formula, this can be converted to the resolution of the machine in user units. This is the highest precision to which an application may specify positions. For example, if 1 uu equals 10 qc, then the command "AxisPosRelStart(0,1);" will move the motor foward by 10 qc; it will not be possible for the application program to move the motor by only 5 qc. Conversely, if the motor moves forward by only 1 qc, then the Apos command is likely to return the same actual position value even though the motor has moved to a new position; the application program will be unable to differentiate between these two motor positions. More information are available in the Help of ApossIDE: "How to Set Up Resolution and Gear Parameters".

Parameters

<i>axis</i>	Axis module number
<i>posencrev</i>	Number of revolutions of the motor
<i>posencqc</i>	Number of quadcounts in POSENCREV revolutions
<i>posfact↔_z</i>	Number of revolutions of the input shaft
<i>posfact↔_n</i>	Number of revolutions of the output shaft in POSFACT_Z revolutions of the input shaft
<i>feedrev</i>	Number of revolutions of the gear box output shaft
<i>feeddist</i>	Distance travelled (in user units) in FEEDREV revolutions of the gear box output shaft

Returns

- value: Always 1 in this function
- value > 0 Process successful
- value = 0 Process is active
- value < 0 Error

Examples

[CAN_1Ax_EPOS4-Test_csp.mc](#), [CAN_1Ax_EPOS4-Test_cst.mc](#), [CAN_1Ax_EPOS4-Test_csv.mc](#), [CAN_2Ax_EPOS4-Test_csp.mc](#), [CAN_4Ax_MiniMACS6_csp.mc](#), [CAN_4Ax_MiniMACS6_cst.mc](#), [CAN_4Ax_MiniMACS6_csv.mc](#), [ECAT_1Ax_EPOS4-Test_cst.mc](#), [ECAT_3Ax_Epos4-Test_csp.mc](#), [Maxon_EC45_flat_1ax_BC_Enc.mc](#), [Maxon_EC45_flat_1ax_BC_Hall.mc](#), [Maxon_EC45_flat_1ax_SC_Hal](#), [Maxon_ECi30_2ax_SC_Hall_Inc.mc](#), [Maxon_ECi40_1ax_SC_OL_Inc.mc](#), [Maxon_ECi40_3ax_SC_OL_Inc](#), [Maxon_ECi52_1ax_SC_SSI.mc](#), [Maxon_RE_40_1ax_Inc.mc](#), [MotorCommissioning_MaxonECi30.mc](#), [Stepper_XY_CL.mc](#), and [Stepper_XY_OL.mc](#).

File Documentation

Definition at line 104 of file SDK_Axis_Setup.mc.

```

105 {
106     AXE_PARAM(axis,POSENCREV) = posencrev;           // Number of revolutions of the motor
107     AXE_PARAM(axis,POSENCQC)  = posencqc;           // Number of quadcounts in POSENCREV revolutions
108     AXE_PARAM(axis,POSFACT_Z) = posfact_z;          // Number of revolutions of the input shaft
109     AXE_PARAM(axis,POSFACT_N) = posfact_n;          // Number of revolutions of the output shaft in
    POSFACT_Z revolutions of the input shaft
110     AXE_PARAM(axis,FEEDREV)   = feedrev;            // Number of revolutions of the gear box output shaft
111     AXE_PARAM(axis,FEEDDIST)  = feeddists;          // Distance travelled (in user units) in FEEDREV
    revolutions of the gear box output shaft
112
113     return(1);
114 }
```

5.8.2.5 sdkSetupPositionPIDControl() long sdkSetupPositionPIDControl (

```

    long axis,
    long kprop,
    long kint,
    long kder )
```

Set parameters for PID position control loop.

The function sets the basic factors of the position PID controller. With the function [sdkSetupPositionPIDControlExt\(\)](#) the extended factors of the position controller can be set. More information are available in the Help of ApossIDE in the topic "Control Loop Design".

Parameters

<i>axis</i>	Axis module number
<i>kprop</i>	Proportional value for PID position control loop (default 30)
<i>kint</i>	Integral value for PID position control loop (default 0)
<i>kder</i>	Derivative value for PID position control loop (default 0)

Returns

value: Always 1 in this function
 value > 0 Process successful
 value = 0 Process is active
 value < 0 Error

Definition at line 36 of file SDK_Axis_Setup.mc.

```

37 {
38     // Parameters for position controller
39     AXE_PARAM(axis, KPROP)      = kprop;           // position-controller proportional factor:      1
    ... 65000
40     AXE_PARAM(axis, KINT)       = kint;            // position-controller integral factor:
    0 ... 65000
41     AXE_PARAM(axis, KDER)       = kder;            // position-controller differencial factor:      0
    ... 65000
42
43     return(1);
44 }
```

5.8.2.6 sdkSetupPositionPIDControlExt()

long sdkSetupPositionPIDControlExt (

```
long axis,
long kprop,
long kint,
long kder,
long kilim,
long kilimtime,
long bandwidth,
long ffvel,
long kffacc,
long kffdec )
```

Set extended parameters for PID position control loop.

The function sets the extended factors of the position PID controller. With the function [sdkSetupPositionPIDControlExt\(\)](#) the basic factors of the position controller can be set. More information are available in the Help of ApossC IDE in the topic "Control Loop Design".

Parameters

<i>axis</i>	Axis module number
<i>kprop</i>	Proportional value for PID position control loop (default 30)
<i>kint</i>	Integral value for PID position control loop (default 0)
<i>kder</i>	Derivative value for PID position control loop (default 0)
<i>kilim</i>	Limit value for the integral sum of the PID position control loop (default 1000)
<i>kilimtime</i>	Time used to increase or decrease the integral limit (default 0)
<i>bandwidth</i>	Bandwidth within which the PID filter is active. 1000 equals to 100% velocity setpoint (default 1000)
<i>ffvel</i>	Velocity Feed forward (default 0)
<i>kffacc</i>	Acceleration Feed forward (default 0)
<i>kffdec</i>	Deceleration Feed Forward (default 0)

Returns

- value: Always 1 in this function
- value > 0 Process successful
- value = 0 Process is active
- value < 0 Error

Examples

[CAN_1Ax_EPOS4-Test_csp.mc](#), [CAN_1Ax_EPOS4-Test_cst.mc](#), [CAN_1Ax_EPOS4-Test_csv.mc](#), [CAN_2Ax_EPOS4-Test_csp.mc](#), [CAN_4Ax_MiniMACS6_csp.mc](#), [CAN_4Ax_MiniMACS6_cst.mc](#), [CAN_4Ax_MiniMACS6_csv.mc](#), [ECAT_1Ax_EPOS4-Test_cst.mc](#), [ECAT_3Ax_Epos4-Test_csp.mc](#), [Maxon_EC45_flat_1ax_BC_Enc.mc](#), [Maxon_EC45_flat_1ax_BC_Hall.mc](#), [Maxon_EC45_flat_1ax_SC_Hall.mc](#), [Maxon_ECi30_2ax_SC_Hall_Inc.mc](#), [Maxon_ECi40_1ax_SC_OL_Inc.mc](#), [Maxon_ECi40_3ax_SC_OL_Inc.mc](#), [Maxon_ECi52_1ax_SC_SSI.mc](#), [Maxon_RE_40_1ax_Inc.mc](#), [Maxon_RE_40_1ax_OL.mc](#), [MotorCommissioning.mc](#), [Stepper_XY_CL.mc](#), and [Stepper_XY_OL.mc](#).

File Documentation

Definition at line 65 of file SDK Axis Setup.mc.

```

66 {
67     // Parameters for position controller
68     AXE_PARAM(axis, KPROP)      = kprop;      // position-controller proportional factor:
69     1 ... 65000
70     AXE_PARAM(axis, KINT)       = kint;        // position-controller integral factor:
71     0 ... 65000
72     AXE_PARAM(axis, KDER)       = kder;        // position-controller differencial factor:
73     0 ... 65000
74     AXE_PARAM(axis, KILIM)      = kilim;       // position-controller limits the integral sum:
75     0 ... 65000
76     AXE_PARAM(axis, KILIMTIME)  = kilimtime;   // position-controller Time used to for integral limit:
77     -10000...10000
78     AXE_PARAM(axis, BANDWIDTH)  = bandwidth;   // position-controller Bandwidth of the PID filter:
79     0 ... 2000
80     AXE_PARAM(axis, FFVEL)      = ffvel;       // position-controller Velocity Feed forward:
81     0...100000
82     AXE_PARAM(axis, KFFACC)      = kffacc;      // position-controller Acceleration Feed Forward:
83     0 ... 8000
84     AXE_PARAM(axis, KFFDEC)      = kffdec;      // position-controller Deceleration Feed Forward:
85     0 ... 8000
86
87     return(1);
88 }

```

5.8.2.7 skdEnableAxisSoftwareLimit()

```
long skdEnableAxisSoftwareLimit (
    long axis,
    long active )
```

Activation of the axis software limits.

This function enables or disables the software limits set with the xy function [skdSetupAxisSoftwareLimit\(\)](#).

Parameters

<i>axis</i>	Axis module number
<i>active</i>	Parameter to enable / disable the software limit 0 Disable software limit 1 Enable software limit

Returns

```
value: Process value
value > 0 Process successful
value = 0 Process is active
value < 0 Error
```

Definition at line 238 of file SDK Axis Setup.mc.

```

239 {
240     if(active == 0)
241     {
242         AXE_PARAM(axis, SWPOSIMACT) = active;
243         AXE_PARAM(axis, SWNEGLIMACT) = active;

```

```
244     print("Disable software limit axis ", axis);
245 }
246 else
247 {
248     AXE_PARAM(axis,SWPOSLIMACT)      =    1;
249     AXE_PARAM(axis,SWNEGLIMACT)      =    1;
250     print("Enable software limit axis ", axis);
251 }
252
253 return(1);
254 }
```

5.8.2.8 skdSetupAxisSoftwareLimit() long skdSetupAxisSoftwareLimit (

 long axis,

 long negLimitUu,

 long posLimitUu)

Setting up a software axis limitation in user units.

For additional safety a software axis limit can be set. This function is for setting the limits and must be activated with the function [skdEnableAxisSoftwareLimit\(\)](#). The limit is set in user units.

Parameters

<i>axis</i>	Axis module number
<i>negLimitUu</i>	Negative axis limit [Uu]
<i>posLimitUu</i>	Positive axis limit [Uu]

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Definition at line 215 of file SDK_Axis_Setup.mc.

```
216 {
217     AXE_PARAM(axis,NEGLIMIT)          =    negLimitUu *
((double) (AXE_PARAM(axis,FEEDREV) *AXE_PARAM(axis,POSFAC_Z) *AXE_PARAM(axis,POSENCQC)) / ((double) AXE_PARAM(ax
218     AXE_PARAM(axis,POSLIMIT)          =    posLimitUu *
((double) (AXE_PARAM(axis,FEEDREV) *AXE_PARAM(axis,POSFAC_Z) *AXE_PARAM(axis,POSENCQC)) / ((double) AXE_PARAM(ax
219     print("Set negative software limit to ", AXE_PARAM(axis,NEGLIMIT), " qc");
220     print("Set positive software limit to ", AXE_PARAM(axis,POSLIMIT), " qc");
221     return(1);
222 }
```

5.9 SDK_Bussystem_EtherCat.mc File Reference

Functions to work with an EtherCat Master/ slave.

File Documentation

```
#include <SysDef.mh>
#include "SDK_Bussystem_EtherCat.mh"
```

Functions

- long [sdkEtherCATMasterInitialize](#) (void)

Initialization of an EtherCAT master.

- long [sdkEtherCATMasterDoMapping](#) (void)

Mapping of EtherCAT Master IO Image.

- long [sdkEtherCATSetupDC](#) (long slaveNo, long cycleTime_ms, long offset_us)

Sets the DC cycle for an individual slave.

- long [sdkEtherCATMasterStart](#) (void)

Start an EtherCAT Master.

5.9.1 Detailed Description

Functions to work with an EtherCat Master/ slave.

Revision

275

5.9.2 Function Documentation

5.9.2.1 [sdkEtherCATMasterDoMapping\(\)](#) long [sdkEtherCATMasterDoMapping](#) (
void)

Mapping of EtherCAT Master IO Image.

Scans through the slave's PDO setup and generates mapping. The slaves PDO setup has to be done by SDO commands before this function is called. The slaves are SAFEOP state after this function.

Parameters

-	
---	--

Returns

value: Number of slaves found on bus
value > 0 Process successful
value < 0 Error

Examples

[ECAT_1Ax_EPOS4-Test_cst.mc](#), and [ECAT_3Ax_Epos4-Test_csp.mc](#).

Definition at line 46 of file SDK_Bssystem_EtherCat.mc.

```
47 {  
48     print("map input/output");  
49     return ECatMasterCommand(0x1000, 2);    // Map Input and Output buffers (go to safeop)  
50 }
```

5.9.2.2 sdkEtherCATMasterInitialize() long sdkEtherCATMasterInitialize (void)

Initialization of an EtherCAT master.

Initializes an EtherCAT master and scans the EtherCat bus for slaves. The slaves are now in PREOP state.

Parameters

-	
---	--

Returns

value: Number of slaves found on bus
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[DeltaRobot_No_SM_EPOS4_ECAT.mc](#), [ECAT_1Ax_EPOS4-Test_cst.mc](#), and [ECAT_3Ax_Epos4-Test_cs](#)

Definition at line 23 of file SDK_Bssystem_EtherCat.mc.

```
24 {  
25     long slaveCount = 0;  
26  
27     ECatMasterCommand(0x1000, 0);    // disable Master  
28     ECatMasterCommand(0x1000, 1);    // start Master  
29     ECatMasterInfo(0x1000, 0, slaveCount);    // slave count
```

File Documentation

```
30
31     print(" ECAT salve count:  ",slaveCount);
32     return(slaveCount);
33 }
```

5.9.2.3 sdkEtherCATMasterStart() `long sdkEtherCATMasterStart (void)`

Start an EtherCAT Master.

If DC is configured, this function will block until the slaves are synchronized to the master. This can take some seconds. After that the system waits until all slaves are in the OP state.

Parameters

-	
---	--

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[DeltaRobot_No_SM_EPOS4_ECAT.mc](#), [ECAT_1Ax_EPOS4-Test_cst.mc](#), and [ECAT_3Ax_Epos4-Test_csp.mc](#).

Definition at line 86 of file SDK_Bussystem_EtherCat.mc.

```
87 {
88     long offsetValue;
89     long dcActive = 0;
90
91     Delay(2);    //Wait for ECatMasterInfo(0x1000, 12) value to be updated the first time
92
93     ECatMasterInfo(0, 30, dcActive);    //is DC configured?
94     print("DC Active:  ", dcActive);
95
96     if(dcActive)
97     {
98         print("wait for DC to lock");
99         offsetValue = 100001;
100         while(abs(offsetValue) > 1000)
101         {
102             ECatMasterInfo(0x1000, 12, offsetValue);
103             print("Offset time [ns]:  ",offsetValue);
104             Delay(300);
105         }
106     }
107
108     print("wait for op");
109     ECatMasterCommand(0x1000, 3);    // request & wait OP state for all slaves
110
111     return(1);
112 }
```

5.9.2.4 sdkEtherCATSetupDC() long sdkEtherCATSetupDC (

long *slaveNo*,

long *cycleTime_ms*,

long *offset_us*)

Sets the DC cycle for an individual slave.

DC0 clock output will be configured with the selected cycle time and offset. Typically, Cycle time is 1ms and offset 0. This function has to be called after sdkEtherCATMasterDoMapping.

Parameters

<i>slaveNo</i>	ID of the EhterCAT slave
<i>cycleTime_ms</i>	cycletime in milliseconds
<i>offset_us</i>	shift offset

Returns

value: Always 1 in this function

value > 0 Process successful

value = 0 Process is active

value < 0 Error

Examples

[DeltaRobot_No_SM_EPOS4_ECAT.mc](#), [ECAT_1Ax_EPOS4-Test_cst.mc](#), and [ECAT_3Ax_Epos4-Test_cs](#)

Definition at line 67 of file SDK_Bussystem_EtherCat.mc.

```
68 {
69     ECatMasterCommand(slaveNo, ( (cycleTime_ms<<8) + (offset_us<<16)));
70
71     return(1);
72 }
```

5.10 SDK_Communication_Ethernet.mc File Reference

Functions to work with an Ethernet as communication.

```
#include <SysDef.mh>
#include "SDK_Communication_Ethernet.mh"
```

Functions

- long [sdkEthernetSendString](#) (wchar sendArray[], long handle)
- long [sdkEthernetPrintConnectionStatus](#) (long status)
- long [sdkEthernetPrintGeneralResult](#) (long result)

Functions to work with an Ethernet as communication.

Revision

264

5.10.2 Function Documentation

5.10.2.1 sdkEthernetPrintConnectionStatus() `long sdkEthernetPrintConnectionStatus (long status)`

Examples

[EthernetSocket_OpenClient.mc](#), [EthernetSocket_OpenServer.mc](#), [EthernetUDPSocket_OpenClient.mc](#), and [EthernetUDPSocket_OpenServer.mc](#).

Definition at line 50 of file SDK_Communication_Ethernet.mc.

```
51 {  
52     switch(status)  
53     {  
54         case -2:         print("SOCK_STATUS_ERROR ");  
55                         break;  
56         case -1:         print("SOCK_STATUS_ERRORSENDING");  
57                         break;  
58         case 0:          print("SOCK_STATUS_INIT");  
59                         break;  
60         case 1:          print("SOCK_STATUS_WAITING");  
61                         break;  
62         case 2:          print("SOCK_STATUS_CONNECTING");  
63                         break;  
64         case 3:          print("SOCK_STATUS_READY");  
65                         break;  
66         case 4:          print("SOCK_STATUS_CLOSED");  
67                         break;  
68         default:         print("SOCK_UNKNOW_STATUS: ",status);  
69                         break;  
70     }  
71     return(1);  
72 }
```

5.10.2.2 sdkEthernetPrintGeneralResult() long sdkEthernetPrintGeneralResult (long result)

Examples

[EthernetSocket_OpenClient.mc](#), [EthernetSocket_OpenServer.mc](#), [EthernetUDPSocket_OpenClient.mc](#),
and [EthernetUDPSocket_OpenServer.mc](#).

Definition at line 89 of file SDK_Communication_Ethernet.mc.

```

90 {
91     printf("sdkEthernetPrintGeneralResult: ");
92     switch(result)
93     {
94         case 0:         print("No error, everything OK");
95                         break;
96         case -1:        print("Out of memory error");
97                         break;
98         case -2:        print("Buffer error");
99                         break;
100        case -3:        print("Timeout");
101                        break;
102        case -4:        print("Routing problem");
103                        break;
104        case -5:        print("Operation in progress");
105                        break;
106        case -6:        print("Illegal value");
107                        break;
108        case -7:        print("Operation would block");
109                        break;
110        case -8:        print("Address in use");
111                        break;
112        case -9:        print("Already connected");
113                        break;
114        case -10:       print("Connection aborted");
115                        break;
116        case -11:       print("Connection reset");
117                        break;
118        case -12:       print("Connection closed");
119                        break;
120        case -13:       print("Not connected");
121                        break;
122        case -14:       print("Illegal argument");
123                        break;
124        case -115:      print("Low-level netif error");
125                        break;
126        case -21:       print("Invalid handle has been passed");
127                        break;
128        case -22:       print("Limit of socket connection exceeded");
129                        break;
130        case -23:       print("The socket is not connected");
131                        break;
132        case -24:       print("Too much data has been attempted to send");
133                        break;
134        case -25:       print("Previous data not yet sent");
135                        break;
136        case -26:       print("Previously received data has not been read and new data has arrived");
137                        break;
138        case -27:       print("Received telegram is too long for the internal buffer");
139                        break;
140        case -28:       print("This controller does not support the socket feature");
141                        break;
142        case -29:       print("The requested protocol type is not supported");
143                        break;
144        default:        print("Unknown result: ",result);
145                        break;
146    }
147    return(1);
148 }
```

5.10.2.3 sdkEthernetSendString() long sdkEthernetSendString (

```

    wchar sendArray[],
    long handle )

```

Examples

[EthernetSocket_OpenClient.mc](#), and [EthernetSocket_OpenServer.mc](#).

Definition at line 25 of file SDK_Communication_Ethernet.mc.

```

26 {
27     long result;
28
29     print("sdkEthernetSendString: ", sendArray);
30     result = EthernetSendTelegram(handle, sendArray, arraylen(sendArray));
31
32     return(result);
33 }

```

5.11 SDK_Encoder_Setup.mc File Reference

Functions for the general amplifier setup.

```

#include <SysDef.mh>
#include "SDK_Encoder_Setup.mh"

```

Functions

- long [sdkSetupIncEncoder](#) (long axis, long encPort, long encRes, long latchType, long latchParam, long latchSlope)

Settings for an incremental encoder.

- long [sdkSetupSinCosEncoder](#) (long axis, long encPort, long encRes, long latchType, long latchParam, long latchSlope)

Settings for an SinCos encoder.

- long [sdkSetupAbsSSIEncoder](#) (long axis, long encPort, long encRes, long clockFreq, long fastUpdate, long datlen, long isBinary, long latchBitMask)

Settings for a SSI encoder.

- long [sdkSetupHallEncoderMACS5](#) (long axis, long hallPort, long hallAligment)

Settings for an hall encoder.

- long [sdkSetupHallEncoder](#) (long axis, long hallAligment, long hallDirection)

Settings for an hall encoder.

5.11.1 Detailed Description

Functions for the general amplifier setup.

Revision

255

5.11.2 Function Documentation

5.11.2.1 sdkSetupAbsSSIEncoder() `long sdkSetupAbsSSIEncoder (`
`long axis,`
`long encPort,`
`long encRes,`
`long clockFreq,`
`long fastUpdate,`
`long datlen,`
`long isBinary,`
`long latchBitMask)`

Settings for a SSI encoder.

With this function the settings can be made which are required for an SSI encoder. Default is the encoder input 0 of axis 0, the encoder input 1 of axis 1 etc. This function is used to reassign the modules. The encoder assignment can be set up variably.

Example for a latching with a singelturn absSSI encoder:

A latching is performed on an encoder with a resolution of 8192 with the actual value of 2192. At the next overflow (reaching 8192) this value is subtracted by 8192. The last marker value is -6000, after the next turn -14 192 and so on.

Parameters

<i>axis</i>	Axis module number
<i>encPort</i>	Encoder port number. Usually, module instance 0 is connected to X1 and so on. Please refer to product manual
<i>encRes</i>	Resolution of the encoder for position feed back in increments (quadcounts)
<i>clockFreq</i>	Clock frequency of the SSI encoder (Hz)
<i>fastUpdate</i>	<p>The fast update is only necessary, when the motor commutation is done with this encoder (PMSM mode)</p> <p>fastUpdate 0: The encoder is updated with the velocity controller update rate (8 kHz on MiniMACS6)</p> <p>fastUpdate 1: The encoder is updated with the current controller update rate (24 kHz on MiniMACS6)</p>

<i>datlen</i>	Databit length of endat position
<i>isBinary</i>	1 if data coding is binary otherwise it is gray coded.
<i>latchBitMask</i>	The result from VIRTLATCH_PISRC_LATCHVALID will be ANDed with this bitmask to decide if a latch should be accepted. In this function the virtual input module is set as source. This is linked to the hardware digital inputs by default.

Returns

value: Always 1 in this function
 value > 0 Process successful
 value = 0 Process is active
 value < 0 Error

Note

Examples

[Maxon_ECi52_1ax_SC_SSI.mc](#), and [SetupAbsSSIEncoder.mc](#).

Definition at line 179 of file SDK_Encoder_Setup.mc.

```

180 {
181   VIRTCONTIN_PARAM(axis, VIRTCONTIN_MODE) = VIRTCONTIN_MODE_ABSOLUTE_DIRECT_ENDLESS;
182   // Defines how the source value is handled
183   VIRTCONTIN_PARAM(axis, VIRTCONTIN_PISRC_COUNTER) =
184   HWCOUNTABS_PROCESS_SRCINDEX(encPort, PO_HWCNTABS_VALUE); // Setup virtual counter module
185   VIRTCONTIN_PARAM(axis, VIRTCONTIN_OVFL_VALUE) = encRes;
186   // Value where counter should produce an overflow
187
188   HWENC_PARAM(encPort, HWENCODER_DATLEN) = datlen;
189   HWENC_PARAM(encPort, HWENCODER_CLOCKFREQ) = clockFreq;
190
191   HWAMP_PARAM(axis, HWAMP_PISRC_ACTPOS) = HWCOUNTABS_PROCESS_SRCINDEX(encPort, PO_HWCNTABS_VALUE);
192   // Source input for actual position
193
194   HWENC_PARAM(encPort, HWENCODER_MODE) = HWENCODER_MODE_SSI_ACTIVE;
195   // Setup SSI Encoder active clock
196
197   if(fastUpdate)
198   // Setup fast update for commutation
199   {
200     HWENC_PARAM(encPort, HWENCODER_FAST_UPDATE) = 1;
201     //HWENC_PARAM(encPort, HWENCODER_FAST_UPDATE) = HWENCODER_FAST_UPDATE_ENABLE;
202   }
203   else
204   {
205     HWENC_PARAM(encPort, HWENCODER_FAST_UPDATE) = 0;
206     //HWENC_PARAM(encPort, HWENCODER_FAST_UPDATE) = HWENCODER_FAST_UPDATE_DISABLE;
207   }
208   if(isBinary)
209   // Setup data coding typ
210   {
211     HWCOUNTABS_PARAM(encPort, HWCNTABS_CODING) = HWCNTABS_CODING_NONE;
212   }
213   else
214   {
215     HWCOUNTABS_PARAM(encPort, HWCNTABS_CODING) = HWCNTABS_CODING_GREY;
216   }
217 }

```

```
209         //HWCOUNTABS_PARAM(encPort, HWCNTABS_CODING) = HWCNTABS_CODING_GRAY;
210     }
211
212     // Virtuallatch module mapping
213     VIRTSLATCH_PARAM(axis,VIRTSLATCH_MODE)=VIRTSLATCH_MODE_SOFTWARE;
214     // Set the source for the lathc module
215     VIRTSLATCH_PARAM(axis,VIRTSLATCH_PISRC_COUNTER)=
216     HWCOUNTABS_PROCESS_SRCINDEX(encPort,PO_HWCNTABS_VALUE);
217     VIRTSLATCH_PARAM(axis, VIRTSLATCH_PISRC_LATCHVALID) =
218     // Set the source for input trigger, could be an other source like user param
219     VIRTSLATCH_PROCESS_SRCINDEX(axis,PO_VIRTSLATCH_VALLONG);
220     // Bitmask for the latchvalid source
221     VIRTSLATCH_PARAM(axis, VIRTSLATCH_LATCHVALID_BITMASK) = latchBitMask;
222
223     return(1);
224 }
```

```
5.11.2.2 sdkSetupHallEncoder() long sdkSetupHallEncoder (
    long axis,
    long hallAlignment,
    long hallDirection )
```

Settings for an hall encoder.

With this function the settings can be made which are required for use hall encoder only.

Parameters

<i>axis</i>	Axis module number
<i>hallAlignment</i>	Hall alignment. See SDO Dictionary Index: 4000, SubIndex: 88
<i>hallDirection</i>	Direction of rotation of the hall signals hallDirection 1 : Normal hallDirection -1 : Inverse (For example Maxon)

Returns

- value: Process value
- value > 0 Process successful
- value = 0 Process is active
- value < 0 Error

Examples

```
Maxon_EC45_flat_1ax_BC_Hall.mc.
```

Definition at line 275 of file SDK_Encoder_Setup.mc.

```
276 {
277     VIRTSLATCH_PARAM(axis, VIRTSLATCH_MODE) = VIRTSLATCH_MODE_ABSOLUTE;
278     // Defines how the source value is handled
279     VIRTSLATCH_PARAM(axis, VIRTSLATCH_PISRC_COUNTER) = HWHALL_PROCESS_SRCINDEX(axis,PO_HWHALL_POS);
280     HWAMP_PARAM(axis, HWAMP_HALL_ALIGNMENT) = hallAlignment; // set hall alignment
```

File Documentation

```
281     HWHALL_PARAM(axis, HWHALL_MODE) = HWHALL_MODE_ENABLE;           // enable hall encoder
282
283     if(hallDirection== -1)
284         VIRTCONTIN_PARAM(axis, VIRTCONTIN_UUFACT_INCNO) = -1;
285
286     return(1);
287 }
```

```
5.11.2.3 sdkSetupHallEncoderMACS5() long sdkSetupHallEncoderMACS5 (
    long axis,
    long hallPort,
    long hallAligment )
```

Settings for an hall encoder.

With this function the settings can be made which are required for an hall encoder. This function only valid for a MACS5 controller

Parameters

<i>axis</i>	Axis module number
<i>hallPort</i>	Hall port number
<i>hallAligment</i>	Hall aligment. See SDO Dictionary Index: 4000, SubIndex: 88

Returns

- value: Process value
- value > 0 Process successful
- value = 0 Process is active
- value < 0 Error

Definition at line 237 of file SDK_Encoder_Setup.mc.

```
238 {
239     long hall_ModNo;
240
241     // Determine Hall Number, note that the Hall-Module number has a fix assignment to the axis number
242     if (axis == 0){ hall_ModNo = 0; }
243     else if (axis == 2){ hall_ModNo = 1; }
244     else if (axis == 3){ hall_ModNo = 2; }
245     else if (axis == 5){ hall_ModNo = 3; }
246     else { // (AxisNo == 1 || AxisNo == 4)
247         print("Axis No ", axis, " cannot be used as BLDC axis");
248         return(-1);
249     }
250
251     HWAMP_PARAM((axis), HWAMP_HALL_ALIGNMENT) = hallAligment;           // set hall alignment
252
253     // Parameterization Hall Sensors
254     HWENC_PARAM((hallPort), HWENCODER_MODE) = HWENCODER_MODE_HALL; // Set encoder mode
255     HWHALL_PARAM((hallPort), HWHALL_MODE) = HWHALL_MODE_ENABLE;       // enable hall decoder
256     HWHALL_PARAM((hall_ModNo), HWHALL_PISRC_ENCOUT) = (hallPort);      // use input number
257
258     return(1);
259 }
```

```
5.11.2.4 sdkSetupIncEncoder() long sdkSetupIncEncoder (
    long axis,
    long encPort,
    long encRes,
    long latchType,
    long latchParam,
    long latchSlope )
```

Settings for an incremental encoder.

With this function the settings can be made which are required for an incremental encoder. Default is the encoder input 0 of axis 0, the encoder input 1 of axis 1 etc. This function is used to reassign the modules. The encoder assignment can be set up variably.

Parameters

<i>axis</i>	Axis module number
<i>encPort</i>	Encoder port number. Usually, module instance 0 is connected to X1 and so on. Please refer to product manual
<i>encRes</i>	Resolution of the encoder for position feed back in increments (quadcounts)
<i>latchType</i>	Defines the latch type 0: Default latchTyp Encoder Z signal 1: A digital input is used as latch signal
<i>latchParam</i>	Parameter depending on the latch type latchType 0: Parameter can be set to 0 latchType 1: The preferred digital input is set (Digital input 1 - 64)
<i>latchSlope</i>	Defines the slope of the trigger signal (Default 1) 0 HWLATCH_SLOPE_CONTINUOUS: Continuous trigger signal 1 HWLATCH_SLOPE_RISING: Rising trigger signal 2 HWLATCH_SLOPE_FALLING: Falling trigger signal 3 HWLATCH_SLOPE_BOTH: Rising/ Falling trigger signal

Returns

- value: Process value
- value > 0 Process successful
- value = 0 Process is active
- value < 0 Error

Examples

```
Maxon_EC45_flat_1ax_BC_Enc.mc, Maxon_EC45_flat_1ax_SC_Hall_Inc.mc, Maxon_ECi30_2ax_SC_Ha
Maxon_ECi40_1ax_SC_OL_Inc.mc, Maxon_ECi40_3ax_SC_OL_Inc.mc, Maxon_RE_40_1ax_Inc.mc,
MotorCommissioning_MaxonECi30.mc, SetupIncEncoder.mc, SetupSignalGeneratorAxis.mc,
SetupSignalGeneratorOpenloop.mc, and Stepper_XY_CL.mc.
```

Definition at line 43 of file SDK_Encoder_Setup.mc.

44 {

File Documentation

```

45     VIRTCONTIN_PARAM(axis, VIRTCONTIN_MODE) = VIRTCONTIN_MODE_ABSOLUTE;
    // Defines how the source value is handled
46     VIRTCONTIN_PARAM(axis, VIRTCONTIN_PISRC_COUNTER) =
    HWCOUNTINC_PROCESS_SRCINDEX(encPort, PO_HWCNTINC_VALUE);          // Setup virtual counter module
47
48     HWENC_PARAM(encPort, HWENCODER_MODE) = HWENCODER_MODE_INCREMENTAL;
    // Setup encoder type
49     HWAMP_PARAM(encPort, HWAMP_ENCRESP) = encRes;
    // Resolution of the encoder
50     HWAMP_PARAM(axis, HWAMP_PISRC_ACTPOS) = HWCOUNTINC_PROCESS_SRCINDEX(encPort, PO_HWCNTINC_VALUE);
    // Source input for actual position
51
52     // Virtuallatch module mapping
53     VIRTLATCH_PARAM(axis, VIRTLATCH_MODE) =
54     VIRTLATCH_MODE_HARDWARE;
55     VIRTLATCH_PARAM(axis, VIRTLATCH_PISRC_COUNTER) =
56     VIRTCONTIN_PROCESS_SRCINDEX(axis, PO_VIRTCONTIN_VALUE);
57     VIRTLATCH_PARAM(axis, VIRTLATCH_PISRC_LATCHCNT) =
58     HWCOUNTINC_PROCESS_SRCINDEX(encPort, PO_HWCNTINC_VALUE);
59     VIRTLATCH_PARAM(axis, VIRTLATCH_PISRC_LATCH) =
60     HWLATCH_PROCESS_SRCINDEX(encPort, PO_HWLATCH_VALUE);
61     VIRTLATCH_PARAM(axis, VIRTLATCH_PISRC_LATCHSTAT) =
62     HWLATCH_PROCESS_SRCINDEX(encPort, PO_HWLATCH_FLAG);
63     VIRTLATCH_PARAM(axis, VIRTLATCH_PISRC_LATCHFIFO_AMOUNT) =
64     HWLATCH_PROCESS_SRCINDEX(encPort, PO_HWLATCH_AMOUNT_IN_FIFO);
65     VIRTLATCH_PARAM(axis, VIRTLATCH_PISRC_LATCHFIFO_READ) =
66     HWLATCH_PROCESS_SRCINDEX(encPort, PO_HWLATCH_FIFOREAD);
67
68     if (latchType == 0) // Default latchtype, encoder Z signal
69     {
70         HWLATCH_PARAM(encPort, HWLATCH_PISRC_TRIGGER) = HWLATCH_PISRC_TRIGGER_ENCZ;
71     }
72     else if (latchType == 1) // Digital input for latch signal
73     {
74         HWLATCH_PARAM(encPort, HWLATCH_PISRC_TRIGGER) = HWLATCH_PISRC_TRIGGER_DINP + latchParam-1;
75         HWLATCH_PARAM(encPort, HWLATCH_SLOPE) = latchSlope;
76     }
77     else // Error
78     {
79         print("Not supported latchType");
80         return(-1);
81     }
82     return(1);
83 }

```

5.11.2.5 sdkSetupSinCosEncoder() long sdkSetupSinCosEncoder (

```

    long axis,
    long encPort,
    long encRes,
    long latchType,
    long latchParam,
    long latchSlope )

```

Settings for an SinCos encoder.

With this function the settings can be made which are required for an SinCos encoder. Default is the encoder input 0 of axis 0, the encoder input 1 of axis 1 etc. This function is used to reassign the modules. The encoder assignment can be set up variably.

Parameters

<i>axis</i>	Axis module number
<i>encPort</i>	Encoder port number. Usually, module instance 0 is connected to X1 and so on. Please refer to product manual
<i>encRes</i>	Resolution of the encoder for position feed back in increments (quadcounts) Input must be multiplied by a factor of 256.
<i>latchType</i>	Defines the latch type 0 : Default latchTyp Encoder Z signal 1 : A digital input is used as latch signal
<i>latchParam</i>	Parameter depending on the latch type latchType 0 : Parameter can be set to 0 latchType 1 : The preferred digital input is set (Digital input 1 - 64)
<i>latchSlope</i>	Defines the slope of the trigger signal (Default 1) 0 HWLATCH_SLOPE_CONTINUOUS : Continuous trigger signal 1 HWLATCH_SLOPE_RISING : Rising trigger signal 2 HWLATCH_SLOPE_FALLING : Falling trigger signal 3 HWLATCH_SLOPE_BOTH : Rising/ Falling trigger signal

Returns

value: Process value
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[SetupSinCosEncoder.mc.](#)

Definition at line 110 of file SDK_Encoder_Setup.mc.

```

111 {
112     VIRTCONTIN_PARAM(axis, VIRTCONTIN_MODE) = VIRTCONTIN_MODE_ABSOLUTE;
113     // Defines how the source value is handled
114     VIRTCONTIN_PARAM(axis, VIRTCONTIN_PISRC_COUNTER) =
115     HWCOUNTINC_PROCESS_SRCINDEX(encPort,PO_HWCNTINC_VALUE);           // Setup virtual counter module
116
117     HWENC_PARAM(encPort, HWENCODER_MODE) = HWENCODER_MODE_SINCOS;
118     // Setup encoder type
119     HWAMP_PARAM(encPort, HWAMP_ENCRESP) = encRes;
120     // Resolution of the encoder
121     HWAMP_PARAM(axis, HWAMP_PISRC_ACTPOS) = HWCOUNTINC_PROCESS_SRCINDEX(encPort,PO_HWCNTINC_VALUE);
122     // Source input for actual position
123
124     // Virtuallatch module mapping
125     VIRTSLATCH_PARAM(axis,VIRTSLATCH_MODE)=
126     VIRTSLATCH_MODE_HARDWARE;
127     VIRTSLATCH_PARAM(axis,VIRTSLATCH_PISRC_COUNTER)=
128     VIRTCONTIN_PROCESS_SRCINDEX(encPort,PO_VIRTCONTIN_VALUE);
129     VIRTSLATCH_PARAM(axis,VIRTSLATCH_PISRC_LATCHCNT)=
130     HWCOUNTINC_PROCESS_SRCINDEX(encPort,PO_HWCNTINC_VALUE);
131     VIRTSLATCH_PARAM(axis,VIRTSLATCH_PISRC_LATCH)=
132     HWLATCH_PROCESS_SRCINDEX(encPort,PO_HWLATCH_VALUE);
133     VIRTSLATCH_PARAM(axis,VIRTSLATCH_PISRC_LATCHSTAT)=
134     HWLATCH_PROCESS_SRCINDEX(encPort,PO_HWLATCH_FLAG);

```

File Documentation

```

130     VIRT_LATCH_PARAM(axis, VIRT_LATCH_PISRC_LATCHFIFO_AMOUNT) =
131     HWLATCH_PROCESS_SRCINDEX(encPort, PO_HWLATCH_AMOUNT_IN_FIFO);
132     VIRT_LATCH_PARAM(axis, VIRT_LATCH_PISRC_LATCHFIFO_READ) =
133     HWLATCH_PROCESS_SRCINDEX(encPort, PO_HWLATCH_FIFOREAD);
134
135     if (latchType == 0) // Default latchtype, encoder Z signal
136     {
137         HWLATCH_PARAM(encPort, HWLATCH_PISRC_TRIGGER) = HWLATCH_PISRC_TRIGGER_ENCZ;
138     }
139     else if (latchType == 1) // Digital input for latch signal
140     {
141         HWLATCH_PARAM(encPort, HWLATCH_PISRC_TRIGGER) = HWLATCH_PISRC_TRIGGER_DINP + latchParam -1;
142         HWLATCH_PARAM(encPort, HWLATCH_SLOPE) = latchSlope;
143     }
144     else // Error
145     {
146         print("Not supported latchType");
147         return(-1);
148     }
149     return(1);
150 }
```

5.12 SDK_Error_Description.mc File Reference

```

#include <SysDef.mh>
#include "SDK_Error_Description.mh"
```

Macros

- #define [SDO_ERR_NO_DATA](#) 0x80000024

Functions

- void [sdkErrorPrint_ApossIdeErrorDescription](#) (long errorCode)

Prints a error description for ApossIDE errors.

- void [sdkErrorPrint_SdoErrorDescription](#) (long errorCode)

Prints a error description for SDO abort codes.

5.12.1 Macro Definition Documentation

5.12.1.1 SDO_ERR_NO_DATA #define SDO_ERR_NO_DATA 0x80000024

5.12.2 Function Documentation

5.12.2.1 sdkErrorPrint_ApossIdeErrorDescription() `void sdkErrorPrint_ApossIdeErrorDescription`
(
 `long errorCode`)

Prints a error description for ApossIDE errors.

This function prints a description for an error of ApossIDE

Parameters

<i>errorCode</i>	errorCode of ApossIDE
------------------	-----------------------

Examples

[CAN_1Ax_DS402_ProfilePositionMode-ppm-Test.mc](#), [CAN_1Ax_DS402_ProfileVelocityMode-pvm-Test.m](#),
[CAN_MultipleAx_DS402_ProfilePositionMode-ppm-Test.mc](#), and [ECAT_1Ax_EPOS4-Test_cst.mc](#).

Definition at line 25 of file SDK_Error_Description.mc.

```

26 {
27     switch(errorCode)
28     {
29         case F_CANMEM:
30             printf("There are no more CAN objects available (CANINI).");
31             break;
32         case F_NOAXES:
33             printf("Axis not in system.");
34             break;
35         case F_ERR:
36             printf("Error not cleared.");
37             break;
38         case F_NOMNU:
39             printf("Failed to move to HOME position.");
40             break;
41         case F_MNU:
42             printf("Home velocity 0.");
43             break;
44         case F_POSERR:
45             printf("Position error.");
46             break;
47         case F_NIO:
48             printf("Index pulse (encoder) not found.");
49             break;
50         case F_UNBEK:
51             printf("Unknown command.");
52             break;
53         case F_GRENZE:
54             printf("Software end limit activated.");
55             break;
56         case F_PARNUM:
57             printf("Illegal parameter number.");
58             break;
59         case F_NOCHZ:
60             printf("Too many nested loops.");

```

File Documentation

```

61         break;
62     case F_NUMFORMAT:
63         printf("INLONG command got an illegal string.");
64         break;
65     case F_CRCPAR:
66         printf("Parameters in memory are corrupted.");
67         break;
68     case F_CRCPRG:
69         printf("Programs in memory are corrupted.");
70         break;
71     case F_WDT:
72         printf("Reset by CPU.");
73         break;
74     case F_ABBRUCH:
75         printf("User abort.");
76         break;
77     case F_VLTCOM:
78         printf("FC communication error.");
79         break;
80     case F_SDOCHN:
81         printf("Number of SDO channels exceeded.");
82         break;
83     case F_FEATUREPROT:
84         printf("Feature protection error.");
85         break;
86     case F_ARMCOM:
87         printf("Communication with ARM lost.");
88         break;
89     case F_ENDSCHALT:
90         printf("Limit switch activated.");
91         break;
92     case F_ILGLSDO:
93         printf("SDO access error in SYSVAR or LINK command.");
94         break;
95     case F_NOPROG:
96         printf("Trial to execute a cleared or empty program.");
97         break;
98     case F_FPGA:
99         printf("Wrong or no FPGA firmware loaded.");
100        break;
101    case F_AMP:
102        printf("Amplifier error.");
103        break;
104    case F_ECAT_MASTER:
105        printf("EtherCAT Master Error.");
106        break;
107    case F_NOINTLEFT:
108        printf("Too many interrupt functions.");
109        break;
110    case F_UMIN:
111        printf("Minimum voltage of the amplifier has been undershot.");
112        break;
113    case F_STACK:
114        printf("Too many nested function/interruption calls.");
115        break;
116    case F_RETURN:
117        printf("Too many return() commands.");
118        break;
119    case F_MATHERR:
120        printf("A floating point function was called with an invalid argument.");
121        break;
122    case F_STATEMACHINE:
123        printf("State machine error.");
124        break;
125    case F_INTPTR:
126        printf("Interrupt happened, but interrupt address is no longer valid.");
127        break;
128    case F_MEEP:
129        printf("Error in verifying.");
130        break;

```

```

131     case F_PATHERR:
132         printf("Path control only: Path error.");
133         break;
134     case F_TIMEOUT:
135         printf("Path control only: Time out in V24 path control.");
136         break;
137     case F_PATHINT:
138         printf("Path control only.");
139         break;
140     case F_DIM:
141         printf("Too many DIM arrays defined.");
142         break;
143     case F_ARRBDS:
144         printf("Invalid array index.");
145         break;
146     case F_LTARRAY:
147         printf("Array number does not exist.");
148         break;
149     case F_NOARRAY:
150         printf("Array is empty.");
151         break;
152     case F_NOSPACE:
153         printf("No more memory space for the new array defined by DIM.");
154         break;
155     case F_ARRSIZERR:
156         printf("Array size does not correspond to the size of the existing array.");
157         break;
158     case F_TEMP:
159         printf("Maximum temperature of the amplifier has been exceeded.");
160         break;
161     case F_UMAX:
162         printf("Maximum voltage of the amplifier has been exceeded.");
163         break;
164     case F_TNDX:
165         printf("Timeout while waiting for index.");
166         break;
167     case F_CMDERR:
168         printf("Internal command error (illegal parameter or format or range).");
169         break;
170     case F_TIM:
171         printf("Too many TIME or PERIOD interrupts.");
172         break;
173     case F_NOVARMEM:
174         printf("Not enough memory for variables.");
175         break;
176     case F_CANGUARD:
177         printf("CAN guarding error.");
178         break;
179     case F_CANIO:
180         printf("CAN send or receive error.");
181         break;
182     case F_MEMLOCK:
183         printf("Memory locked.");
184         break;
185     case F_CURARR:
186         printf("Illegal curve array in SETCURVE.");
187         break;
188     case F_ENCERR:
189         printf("Encoder error.");
190         break;
191     case F_DYNSTACK:
192         printf("Stack overflow: Too many local variables or nested function calls.");
193         break;
194     case F_DYNMEM:
195         printf("Out of dynamic memory.");
196         break;
197     case F_OPALINDX:
198         printf("Too many test indices in data logging command.");
199         break;
200     case F_ILGLCODE:

```

File Documentation

```

201         printf("Code is too old for the current firmware.");
202         break;
203     case F_IMAX:
204         printf("Internal overcurrent detection of power stage.");
205         break;
206     case F_LIMIT_VIOLATION:
207         printf("Wrong direction after limit switch tripped and error reset.");
208         break;
209     case F_I2TLIMIT:
210         printf("I2T Limit exceeded.");
211         break;
212     case F_AMPCOM:
213         printf("Communication with amplifier interrupted.");
214         break;
215     case F_AMPPARAM:
216         printf("Illegal access to amplifier parameter.");
217         break;
218     case F_NOTSUPPORTED:
219         printf("Command is not supported.");
220         break;
221     case F_MEMDUMP:
222         printf("MemoryDump error.");
223         break;
224     case F_DIVIDEBY0:
225         printf("Divide by 0.");
226         break;
227     case F_ECAT_SLAVE:
228         printf("EtherCAT Slave Error.");
229         break;
230     case F_PROFGEN:
231         printf("Profile Generator Error.");
232         break;
233     case F_DIMTOOSHORT:
234         printf("DIM array is too short.");
235         break;
236     case F_INTERNALERROR:
237         printf("Internal firmware error.");
238         break;
239     case F_OPTIONBOOT:
240         printf("Option boot failure.");
241         break;
242     case F_COMMAND_USAGE:
243         printf("Wrong command usage.");
244         break;
245     case F_PROCESSDATAUSAGE:
246         printf("Illegal write access to process data.");
247         break;
248     case F_ARGTOOLONG:
249         printf("The size of string argument too long.");
250         break;
251     case F_AUTHFAILED:
252         printf("Authentication failed.");
253         break;
254     case F_AMP_NOTREADY:
255         printf("Amplifier not ready.");
256         break;
257     case F_LOGIC_UMIN:
258         printf("Logic Supply Voltage too low.");
259         break;
260     case F_ZFCUPDATE:
261         printf("ZFC file update failed.");
262         break;
263     case F_HWFAIL:
264         printf("Hardware Failure.");
265         break;
266     case F_AMP_INIT_ORDER:
267         printf("Amplifier's initialization order violated.");
268         break;
269     case F_HALL:
270         printf("Hall sensor problem.");

```

```

271         break;
272     case F_HALL_ANGLE:
273         printf("Hall angle detection error.");
274         break;
275     case F_STO:
276         printf("STO error.");
277         break;
278     default:
279         printf("Unknown error code (%d).", errorCode);
280         break;
281     }
282 }

```

5.12.2.2 sdkErrorPrint_SdoErrorDescription() void sdkErrorPrint_SdoErrorDescription (long errorCode)

Prints a error description for SDO abort codes.

This function prints a description for CiA 301 SDO abort codes

Parameters

<i>errorCode</i>	SDO abort code
------------------	----------------

Examples

[CAN_1Ax_DS402_ProfilePositionMode-ppm-Test.mc](#), [CAN_1Ax_DS402_ProfileVelocityMode-pvm-Test.mc](#), [CAN_1Ax_EPOS4-Test_csp.mc](#), [CAN_1Ax_EPOS4-Test_cst.mc](#), [CAN_1Ax_EPOS4-Test_csv.mc](#), [CAN_2Ax_EPOS4-Test_csp.mc](#), and [CAN_MultipleAx_DS402_ProfilePositionMode-ppm-Test.mc](#).

Definition at line 290 of file SDK_Error_Description.mc.

```

291 {
292     switch (errorCode)
293     {
294         case SDO_ERR_ACCESS_RO:
295             printf("Attempt to write a read only object.");
296             break;
297         case SDO_ERR_ACCESS_UNSUPPORTED:
298             printf("Unsupported access to an object.");
299             break;
300         case SDO_ERR_ACCESS_WO:
301             printf("Attempt to read a write only object.");
302             break;
303         case SDO_ERR_BLOCK_CRC:
304             printf("Invalid block CRC value (block mode only).");
305             break;
306         case SDO_ERR_BLOCK_SEQUENCE:
307             printf("Invalid block sequence number (block mode only).");
308             break;
309         case SDO_ERR_BLOCK_SIZE:
310             printf("Invalid block size (block mode only).");
311             break;
312         case SDO_ERR_COMMAND:
313             printf("Client/server command specifier not valid or unknown.");
314             break;
315         case SDO_ERR_DATA_STORE:

```

File Documentation

```

316         printf("Data cannot be transferred or stored to the application.");
317         break;
318     case SDO_ERR_DATA_STORE_LOCAL:
319         printf("Data cannot be transferred or stored to the application because of local
control.");
320         break;
321     case SDO_ERR_DATA_STORE_STATE:
322         printf("Data cannot be transferred or stored to the application because of the present
device state.");
323         break;
324     case SDO_ERR_DATATYPE:
325         printf("Data type does not match, length of service parameter does not match.");
326         break;
327     case SDO_ERR_DATATYPE_HIGH:
328         printf("Data type does not match, length of service parameter too high.");
329         break;
330     case SDO_ERR_DATATYPE_LOW:
331         printf("Data type does not match, length of service parameter too low.");
332         break;
333     case SDO_ERR_GENERAL:
334         printf("General error.");
335         break;
336     case SDO_ERR_GENERAL_DEVICE:
337         printf("General internal incompatibility in the device.");
338         break;
339     case SDO_ERR_GENERAL_PARAMETER:
340         printf("General parameter incompatibility reason.");
341         break;
342     case SDO_ERR_HARDWARE:
343         printf("Access failed due to a hardware error.");
344         break;
345     case SDO_ERR_MAPPING_LENGTH:
346         printf("The number and length of the objects to be mapped would exceed PDO length.");
347         break;
348     case SDO_ERR_MAPPING_OBJECT:
349         printf("Object cannot be mapped to the PDO.");
350         break;
351     case SDO_ERR_MEMORY:
352         printf("Out of memory.");
353         break;
354     case SDO_ERR_NO_DATA:
355         printf("No data available.");
356         break;
357     case SDO_ERR_NO_OBJECT:
358         printf("Object does not exist in the object dictionary.");
359         break;
360     case SDO_ERR_NO_SUB_INDEX:
361         printf("Sub-index does not exist.");
362         break;
363     case SDO_ERR_OBJECT_DICTIONARY:
364         printf("Object dictionary dynamic generation fails or no object dictionary is present.");
365         break;
366     case SDO_ERR_SDO_CONNECTION:
367         printf("Resource not available: SDO connection.");
368         break;
369     case SDO_ERR_TIMEOUT:
370         printf("SDO protocol timed out.");
371         break;
372     case SDO_ERR_TOGGLE_BIT:
373         printf("Toggle bit not altered.");
374         break;
375     case SDO_ERR_VALUE_RANGE:
376         printf("Invalid value for parameter (download only).");
377         break;
378     case SDO_ERR_VALUE_HIGH:
379         printf("Value of parameter written too high (download only).");
380         break;
381     case SDO_ERR_VALUE_LOW:
382         printf("Value of parameter written too low (download only).");
383         break;

```

```
384         case SDO_ERR_VALUE_MIN_MAX:
385             printf("Maximum value is less than minimum value.");
386             break;
387         default:
388             printf("Unknown error code (0x%lX).", errorCode);
389             break;
390     }
391 }
```

5.13 SDK_Information_General.mc File Reference

Functions to get general informations.

```
#include <SysDef.mh>
#include "SDK_Information_General.mh"
```

Functions

- long [sdkInfoPrintSoftware](#) ()
Prints the software versions of the controller.
- long [sdkInfoPrintHardware](#) ()
Prints the hardware information of the controller.
- long [sdkInfoPrintAxesPos](#) ()
Prints the position information of all axes.
- long [sdkInfoPrintPosPID](#) ()
Prints the parameters of the position PID controller.

5.13.1 Detailed Description

Functions to get general informations.

Revision

24

5.13.2 Function Documentation

5.13.2.1 sdkInfoPrintAxesPos() `long sdkInfoPrintAxesPos ()`

Prints the position information of all axes.

Function to print the actual, command and marker position of all axes.

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[InformationGeneral.mc](#), [Maxon_EC45_flat_1ax_BC_Enc.mc](#), [Maxon_EC45_flat_1ax_BC_Hall.mc](#), [Maxon_EC45_flat_1ax_SC_Hall_Inc.mc](#), [Maxon_ECi40_1ax_SC_OL_Inc.mc](#), [Maxon_ECi40_3ax_SC_OL_Inc.m](#), [Maxon_ECi52_1ax_SC_SSI.mc](#), [Maxon_RE_40_1ax_Inc.mc](#), [SetupAbsSSIEncoder.mc](#), [SetupIncEncoder.mc](#), [SetupSignalGeneratorAxis.mc](#), [SetupSignalGeneratorOpenloop.mc](#), [SetupSinCosEncoder.mc](#), [Stepper_XY_CL.mc](#), [Stepper_XY_OL.mc](#), and [VirtualMaster_ProfileMode.mc](#).

Definition at line 78 of file SDK_Information_General.mc.

```
79 {
80     long axiscnt, axis;
81
82     axiscnt = SYS_INFO(SYS_MAX_AXES);    // Number of axes on controller
83
84     print("Axis Positions:");
85     print("    Axis",chr(9),"Actual",chr(9),"Command", chr(9),"Marker");
86
87     for (axis = 0; axis < axiscnt; axis++)
88     {
89         print("    ",axis,chr(9),Apos(axis),chr(9),Cpos(axis), chr(9),Ipos(axis));
90     }
91     print("");
92     return(1);
93 }
```

5.13.2.2 sdkInfoPrintHardware() `long sdkInfoPrintHardware ()`

Prints the hardware information of the controller.

Function to print the hardware version of the controller.

Returns

value: Always 1 in this function
 value > 0 Process successful
 value = 0 Process is active
 value < 0 Error

Examples

[InformationGeneral.mc.](#)

Definition at line 48 of file SDK_Information_General.mc.

```

49 {
50
51     print("*****");
52     print("----- H/W information -----");
53     print("*****\n");
54
55     print("Amplifier:\t\t\t", SYS_INFO(SYS_ZB_AMP_NO));
56     print("Encoder:\t\t\t", SYS_INFO(SYS_MAX_ENCODER ));
57     print("Incremental counters:\t", SYS_INFO(SYS_MAX_CNTINC));
58     print("Absolute counters:\t\t", SYS_INFO(SYS_MAX_CNTABS));
59     print("Universal counters:\t", SYS_INFO(SYS_MAX_CNTUNI));
60     print("Comparators:\t\t", SYS_INFO(SYS_MAX_COMPARATORS));
61     print("Latches:\t\t\t", SYS_INFO(SYS_MAX_LATCH));
62     print("Signal generators:\t\t", SYS_INFO(SYS_MAX_SIGGEN));
63     print("CAN Module:\t\t", SYS_INFO(SYS_MAX_CANBUS));
64     print("");
65
66     return(1);
67 }
```

5.13.2.3 sdkInfoPrintPosPID() `long sdkInfoPrintPosPID ()`

Prints the parameters of the position PID controller.

Function to print the basic settings of the PID controller.

Returns

value: Always 1 in this function
 value > 0 Process successful
 value = 0 Process is active
 value < 0 Error

Examples

[InformationGeneral.mc.](#)

Definition at line 104 of file SDK_Information_General.mc.

```

105 {
106     long axisCnt, axis;
107
108     axisCnt = SYS_INFO(SYS_MAX_AXES);    // Number of axes on controller
109
110     print("PID control parameters:");
```

File Documentation

```

111     print("    Axis",chr(9),"Prop",chr(9),"Int",chr(9),"Diff",chr(9),"Timer",chr(9),"I-Limit");
112
113     for (axis = 0; axis < axisCnt; axis++)
114     {
115         printf("    %ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",
116             axis,
117             AXE_PARAM(axis, KPROP), // Proportional factor
118             AXE_PARAM(axis, KINT), // Integral factor
119             AXE_PARAM(axis, KDER), // Derivative factor
120             AXE_PARAM(axis, TIMER), // Derivative time
121             AXE_PARAM(axis, KILIM)); // Integration limit
122     }
123     print("");
124     return(1);
125 }

```

5.13.2.4 sdkInfoPrintSoftware() long sdkInfoPrintSoftware ()

Prints the software versions of the controller.

Function to print the software versions of the controller.

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[InformationGeneral.mc.](#)

Definition at line 22 of file SDK_Information_General.mc.

```

23 {
24
25     print("*****");
26     print("----- Software information -----");
27     print("*****\n");
28
29     print("Hardware ID:\t", SYS_INFO(SYS_HARDWARE_ID));
30     print("Firmware:\t\t", SYS_INFO(SYS_FW_CPU));
31     print("FPGA:\t\t", SYS_INFO(SYS_FPGA_SW_VERSION));
32     print("Amplifier:\t\t", SYS_INFO(SYS_FW_AMPLIFIER_VERSION));
33     print("Coprocessor:\t", SYS_INFO(SYS_FW_COPROCESSOR_VERSION));
34     print("Bootloader:\t", SYS_INFO(SYS_BOOTLOADER_SW_VERSION));
35     print("");
36
37     return(1);
38 }

```

5.14 SDK_Kinematics_Setup.mc File Reference

Functions for the kinematics setup.

```

#include <SysDef.mh>
#include "SDK_Kinematics_Setup.mh"

```

Functions

- long [sdkSetupDeltaKinematics](#) (long axis_0, long axis_1, long axis_2, long prms_A, long prms_B, long prms_C, long prms_D, long prms_minMotAng, long prms_maxMotAng)

Setup a Delta multi-axis kinematics model.

- long [sdkSetupCartesian2dKinematics](#) (long axis_0, long axis_1)

Setup a 2-dimensional cartesian multi-axis kinematics model.

- long [sdkSetupCartesian3dKinematics](#) (long axis_0, long axis_1, long axis_2)

Setup a 3-dimensional cartesian multi-axis kinematics model.

- long [sdkSetupHBotCoreXYKinematics](#) (long axis_0, long axis_1)

Setup a HBot or CoreXY multi-axis kinematics model.

- long [sdkSetupScara2dKinematics](#) (long axis_0, long axis_1, long prms_A, long prms_B, long prms_orient)

Setup a Scara 2d multi-axis kinematics model.

- long [sdkSetupScara3dKinematics](#) (long axis_0, long axis_1, long axis_2, long prms_A, long prms_B, long prms_orient)

Setup a Scara 3d multi-axis kinematics model.

- long [sdkSetupDualScara2dKinematics](#) (long axis_0, long axis_1, long prms_A, long prms_B, long prms_C, long prms_D, long prms_distance)

Setup a Dual Scara 2d multi-axis kinematics model.

- long [sdkSetupDualScara3dKinematics](#) (long axis_0, long axis_1, long axis_2, long prms_A, long prms_B, long prms_C, long prms_D, long prms_distance)

Setup a Dual Scara 3d multi-axis kinematics model.

- transform [sdkSetupWorkCoordKinematics](#) (transform workToMachine, double originTranslate_X, double originTranslate_Y, double originTranslate_Z, double rotation_XY, double scale)

Setup the working coordinates for a kinematics System.

5.14.1 Detailed Description

Functions for the kinematics setup.

Revision

246

5.14.2.1 sdkSetupCartesian2dKinematics() `long sdkSetupCartesian2dKinematics (`
 `long axis_0,`
 `long axis_1)`

Setup a 2-dimensional cartesian multi-axis kinematics model.

This function sets up a 2-dimensional cartesian kinematic. More information can be found in the ApossIDE Help.

Parameters

<i>axis</i> ↔ _0	X Axis
<i>axis</i> ↔ _1	Y Axis

Returns

value: Kinematics handle used to reference this group of axes.
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Definition at line 68 of file SDK_Kinematics_Setup.mc.

```
69 {
70     long   axes[2], handle;
71     double prms[1];
72
73     axes[0] = axis_0;    // Define X axis
74     axes[1] = axis_1;    // Define Y axis
75
76     prms[0] = 2;         // Set 2-dimensional system
77
78     handle = KinematicsSetup("Cartesian", axes, prms); // Get the kinematics handler
79
80     return(handle);
81 }
```

5.14.2.2 sdkSetupCartesian3dKinematics() long sdkSetupCartesian3dKinematics (

 long axis_0,

 long axis_1,

 long axis_2)

Setup a 3-dimensional cartesian multi-axis kinematics model.

This function sets up a 3-dimensional cartesian kinematic. More information can be found in the ApossIDE Help.

Parameters

<i>axis</i> ↔ _0	X Axis
<i>axis</i> ↔ _1	Y Axis
<i>axis</i> ↔ _2	Z Axis

value: Kinematics handle used to reference this group of axes.
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Definition at line 95 of file SDK_Kinematics_Setup.mc.

```
96 {  
97     long   axes[3], handle;  
98     double prms[1];  
99  
100     axes[0] = axis_0;    // Define X axis  
101     axes[1] = axis_1;    // Define Y axis  
102     axes[2] = axis_2;    // Define Z axis  
103  
104     prms[0] = 3;         // Set 3-dimensional system  
105  
106     handle = KinematicsSetup("Cartesian", axes, prms); // Get the kinematics handler  
107  
108     return(handle);  
109 }
```

5.14.2.3 sdkSetupDeltaKinematics() long sdkSetupDeltaKinematics (
 long axis_0,
 long axis_1,
 long axis_2,
 long prms_A,
 long prms_B,
 long prms_C,
 long prms_D,
 long prms_minMotAng,
 long prms_maxMotAng)

Setup a Delta multi-axis kinematics model.

This function sets up a Delta kinematic. The parameters of the mechanics must be known and are transferred to the function. More information can be found in the ApossIDE Help.

Parameters

<i>axis_0</i>	Axis module number
<i>axis_1</i>	Axis module number
<i>axis_2</i>	Axis module number
<i>prms_A</i>	Definition of length A [mm*10](80.0 mm => 800)
<i>prms_B</i>	Definition of length B [mm*10](80.0 mm => 800)
<i>prms_C</i>	Definition of length C [mm*10](80.0 mm => 800)
<i>prms_D</i>	Definition of length D [mm*10](80.0 mm => 800)
<i>prms_minMotAng</i>	Minimum motor angle [°]
<i>prms_maxMotAng</i>	Maximum motor angle [°]

Returns

value: Kinematics handle used to reference this group of axes.
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[DeltaRobot_No_SM_EPOS4_ECAT.mc.](#)

Definition at line 36 of file SDK_Kinematics_Setup.mc.

```
37 {
38     long    axes[3], handle;
39     double  prms[6];
40
41     axes[0] = axis_0;    // Define axis 0
42     axes[1] = axis_1;    // Define axis 1
43     axes[2] = axis_2;    // Define axis 2
44
45     prms[0] = prms_A;    // Length A
46     prms[1] = prms_B;    // Length B
47     prms[2] = prms_C;    // Length C
48     prms[3] = prms_D;    // Length D
49
50     prms[4] = rad(prms_minMotAng);    // The arm must not move higher than "prms_minMotAng" (for
safety).
51     prms[5] = rad(prms_maxMotAng);    // The arm must not move lower than "prms_maxMotAng" (for safety).
52
53     handle = KinematicsSetup("Delta", axes, prms); // Get the kinematics handler
54
55     return(handle);
56 }
```

5.14.2.4 sdkSetupDualScara2dKinematics() long sdkSetupDualScara2dKinematics (

long axis_0,

long axis_1,

long prms_A,

long prms_B,

long prms_C,

long prms_D,

long prms_distance)

Setup a Dual Scara 2d multi-axis kinematics model.

This function sets up a Dual Scara 2d kinematic. The parameters of the mechanics must be known and are transferred to the function. More information can be found in the ApossIDE Help.

Parameters

<i>axis_0</i>	Axis module number
<i>axis_1</i>	Axis module number
<i>prms_A</i>	The length of the left primary arm, in Machine Coordinate System units
<i>prms_B</i>	The length of the left secondary arm, in Machine Coordinate System units
<i>prms_C</i>	The length of the right primary arm, in Machine Coordinate System units
<i>prms_D</i>	The length of the right secondary arm, in Machine Coordinate System units
<i>prms_distance</i>	The separation distance between axis 1 and axis 2

value: Kinematics handle used to reference this group of axes.
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Definition at line 218 of file SDK_Kinematics_Setup.mc.

```
219 {
220     long    axes[2], handle;
221     double prms[6];
222
223     axes[0] = axis_0;    // Define axis 0
224     axes[1] = axis_1;    // Define axis 1
225
226     prms[0] = prms_A;    // Length A
227     prms[1] = prms_B;    // Length B
228     prms[2] = prms_C;    // Length C
229     prms[3] = prms_D;    // Length D
230     prms[4] = prms_distance; // Separation distance
231     prms[5] = 2;        // Set 2-dimensional system
232
233     handle = KinematicsSetup("DualScara", axes, prms); // Get the kinematics handler
234
235     return(handle);
236 }
```

```
5.14.2.5 sdkSetupDualScara3dKinematics() long sdkSetupDualScara3dKinematics (
    long axis_0,
    long axis_1,
    long axis_2,
    long prms_A,
    long prms_B,
    long prms_C,
    long prms_D,
    long prms_distance )
```

Setup a Dual Scara 3d multi-axis kinematics model.

This function sets up a Dual Scara 3d kinematic. The parameters of the mechanics must be known and are transferred to the function. More information can be found in the ApossIDE Help.

Parameters

axis_0	Axis module number
axis_1	Axis module number
axis_2	Axis module number
prms_A	The length of the left primary arm, in Machine Coordinate System units
prms_B	The length of the left secondary arm, in Machine Coordinate System units
prms_C	The length of the right primary arm, in Machine Coordinate System units
prms_D	The length of the right secondary arm, in Machine Coordinate System units
prms_distance	The separation distance between axis 1 and axis 2

value: Kinematics handle used to reference this group of axes.
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Definition at line 255 of file SDK_Kinematics_Setup.mc.

```
256 {
257     long    axes[3], handle;
258     double prms[6];
259
260     axes[0] = axis_0;    // Define axis 0
261     axes[1] = axis_1;    // Define axis 1
262     axes[2] = axis_2;    // Define axis 2
263
264     prms[0] = prms_A;    // Length A
265     prms[1] = prms_B;    // Length B
266     prms[2] = prms_C;    // Length C
267     prms[3] = prms_D;    // Length D
268     prms[4] = prms_distance; // Separation distance
269     prms[5] = 3;        // Set 3-dimensional system
270
271     handle = KinematicsSetup("DualScara", axes, prms); // Get the kinematics handler
272
273     return(handle);
274 }
```

5.14.2.6 sdkSetupHBotCoreXYKinematics() long sdkSetupHBotCoreXYKinematics (

 long axis_0,

 long axis_1)

Setup a HBot or CoreXY multi-axis kinematics model.

This function sets up a HBot or CoreXY kinematic. More information can be found in the ApossIDE Help.

Parameters

<i>axis</i> _↔ _0	Axis module number
<i>axis</i> _↔ _1	Axis module number

Returns

value: Kinematics handle used to reference this group of axes.
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Definition at line 122 of file SDK_Kinematics_Setup.mc.

```
123 {
```

File Documentation

```
124     long   axes[2], handle;
125     double prms[1];
126
127     axes[0] = axis_0;    // Define axis 0
128     axes[1] = axis_1;    // Define axis 1
129     prms[0] = 0;         // Just to prevent a "Referenced but never assigned" warning.
130
131     handle = KinematicsSetup("Hbot", axes, prms); // Get the kinematics handler
132
133     return(handle);
134 }
```

```
5.14.2.7 sdkSetupScara2dKinematics() long sdkSetupScara2dKinematics (
    long axis_0,
    long axis_1,
    long prms_A,
    long prms_B,
    long prms_orient )
```

Setup a Scara 2d multi-axis kinematics model.

This function sets up a Scara 2d kinematic. The parameters of the mechanics must be known and are transferred to the function. More information can be found in the ApossIDE Help.

Parameters

<i>axis_0</i>	Axis module number
<i>axis_1</i>	Axis module number
<i>prms_A</i>	The length of the primary arm, in Machine Coordinate System units
<i>prms_B</i>	The length of the secondary arm, in Machine Coordinate System units
<i>prms_orient</i>	The orientation of the arms: 1 - Left arm orientation, -1 - Right arm orientation

Returns

- value: Kinematics handle used to reference this group of axes.
- value > 0 Process successful
- value = 0 Process is active
- value < 0 Error

Definition at line 150 of file SDK_Kinematics_Setup.mc.

```
151 {
152     long   axes[2], handle;
153     double prms[4];
154
155     axes[0] = axis_0;    // Define axis 0
156     axes[1] = axis_1;    // Define axis 1
157
158     prms[0] = prms_A;     // Length A
159     prms[1] = prms_B;     // Length B
160     prms[2] = prms_orient; // Orientation of the System "Left" / "Right"
161     prms[3] = 2;          // Set 2-dimensional system
```

```
162
163     handle = KinematicsSetup("Scara", axes, prms); // Get the kinematics handler
164
165     return(handle);
166 }
```

5.14.2.8 sdkSetupScara3dKinematics() long sdkSetupScara3dKinematics (

long axis_0,

long axis_1,

long axis_2,

long prms_A,

long prms_B,

long prms_orient)

Setup a Scara 3d multi-axis kinematics model.

This function sets up a Scara 3d kinematic. The parameters of the mechanics must be known and are transferred to the function. More information can be found in the ApossIDE Help.

Parameters

<i>axis_0</i>	Axis module number
<i>axis_1</i>	Axis module number
<i>axis_2</i>	Axis module number
<i>prms_A</i>	The length of the primary arm, in Machine Coordinate System units
<i>prms_B</i>	The length of the secondary arm, in Machine Coordinate System units
<i>prms_orient</i>	The orientation of the arms: 1 - Left arm oientation, -1 - Right arm orientation

Returns

value: Kinematics handle used to reference this group of axes.

value > 0 Process successful

value = 0 Process is active

value < 0 Error

Definition at line 183 of file SDK_Kinematics_Setup.mc.

```
184 {
185     long   axes[3], handle;
186     double prms[4];
187
188     axes[0] = axis_0;    // Define axis 0
189     axes[1] = axis_1;    // Define axis 1
190     axes[2] = axis_2;    // Define axis 2
191
192     prms[0] = prms_A;    // Length A
193     prms[1] = prms_B;    // Length B
194     prms[2] = prms_orient; // Orientation of the System "Left" / "Right"
195     prms[3] = 3;         // Set 3-dimensional system
196
197     handle = KinematicsSetup("Scara", axes, prms); // Get the kinematics handler
```

File Documentation

```
198
199     return(handle);
200 }
```

5.14.2.9 sdkSetupWorkCoordKinematics() transform sdkSetupWorkCoordKinematics (

```
transform workToMachine,
double originTranslate_X,
double originTranslate_Y,
double originTranslate_Z,
double rotation_XY,
double scale )
```

Setup the working coordinates for a kinematics System.

Set up the Work-to-Machine transformation. This is the transformation from Work (i.e. paper) coordinates to Machine coordinates. There are three parts:

1. Scaling: For example. The Work coordinate system uses mm's and the Machine coordinate system uses micrometers. So one Work unit (i.e. 1 mm) will be 1000 Machine units (i.e. 1000 micrometers).
2. Rotation: We would like to define the orientation. For example. The Work coordinate system must be rotated CLOCKWISE by 90 degrees (i.e. -90 degrees counterclockwise) with respect to the Machine coordinate system.
3. Translation: The origin position of the coordinate system can be set in which there is a shift in relation to the machine coordinate system.

Parameters

<i>workToMachine</i>	In: Transform variable of the coordinate system
<i>originTranslate</i> ↔ <i>_X</i>	Translational movement of the coordinate system -> X [in mm depending on the scaling]
<i>originTranslate</i> ↔ <i>_Y</i>	Translational movement of the coordinate system -> Y [in mm depending on the scaling]
<i>originTranslate</i> ↔ <i>_Z</i>	Translational movement of the coordinate system -> Z [in mm depending on the scaling]
<i>rotation_XY</i>	Rotation around the XY plane [°]
<i>scale</i>	Scaling of the system []

Returns

value Transform variable of the coordinate system

Examples

[DeltaRobot_No_SM_EPOS4_ECAT.mc.](#)

Definition at line 300 of file SDK_Kinematics_Setup.mc.

```

301 {
302     TransIdent(workToMachine);
303     TransScale(workToMachine, scale);
304     TransRotateXY(workToMachine, rotation_XY*PI/180.0);
305     TransTranslate(workToMachine, originTranslate_X, originTranslate_Y, originTranslate_Z);
306
307     print("Working coordinates are changed");
308     return(workToMachine);
309 }
```

5.15 SDK_Miscellaneous_IO.mc File Reference

Functions with IO samples.

```
#include <SysDef.mh>
#include "SDK_Miscellaneous_IO.mh"
```

Functions

- long [sdkSetupPwmGenerator_UserParamMode](#) (long userParam, long digOutPWM, long frequency, long cycleRange, long polarity)

Setting up a PWM output with a user parameter as input source.

- long [sdkSetupPwmGenerator_AxisModeVel](#) (long axisNo, long digOutEnable, long digOutPWM, long frequency, long cycleRange, long polarity)

Setting up a PWM output with a virtual amplifier as input source.

- long [sdkScaleAnalogInput](#) (long analogInputNo, long minVoltage, long maxVoltage, long offset← Voltage, long minValue, long maxValue)

Scaling of an analog input with user units.

5.15.1 Detailed Description

Functions with IO samples.

Revision

183

5.15.2 Function Documentation

```
5.15.2.1 sdkScaleAnalogInput() long sdkScaleAnalogInput (
    long analogInputNo,
    long minVoltage,
    long maxVoltage,
    long offsetVoltage,
    long minValue,
    long maxValue )
```

Scaling of an analog input with user units.

This function scales an analog input signal to user specific user units. Additionally, an offset of the analog input signal can be specified.

Parameters

<i>analogInputNo</i>	Analog input number(0-xx zero based)
<i>minVoltage</i>	Minimum input voltage (mV)
<i>maxVoltage</i>	Maximum input voltage (mV)
<i>offsetVoltage</i>	Voltage offest, for example, if the voltage range is 0 to 4 V, 2 V should be interpreted as 0 (mV)
<i>minValue</i>	Scaling value minimum value to be used in the program (Uu)
<i>maxValue</i>	Scaling value maximal value to be used in the program (Uu)

Returns

- value: Always 1 in this function
- value > 0 Process successful
- value = 0 Process is active
- value < 0 Error

Examples

[PWM_1Ax_ESCON.mc.](#)

Definition at line 127 of file SDK_Miscellaneous_IO.mc.

```
128 {
129     double scale;
130
131     // Scale the maximal and minimal voltage to internal value
132     maxVoltage = ((double)0x8000/10000) * maxVoltage;
133     minVoltage = ((double)0x8000/10000) * minVoltage;
134
135     // Scale
136     scale = ((double)maxValue-(double)minValue) / ((double)maxVoltage-(double)minVoltage);
137     if((long)scale>1)
138     {
139         VIRTANIN_PARAM(analogInputNo,VIRTANIN_UUFACT_UNITNO)=1000;
140         VIRTANIN_PARAM(analogInputNo,VIRTANIN_UUFACT_DIGNO)=(long) (scale*1000);
141     }
142     else
143     {
144         VIRTANIN_PARAM(analogInputNo,VIRTANIN_UUFACT_UNITNO)=(long) (scale*1000);
145         VIRTANIN_PARAM(analogInputNo,VIRTANIN_UUFACT_DIGNO)=1000;
146     }
```

```
147
148 // Offset
149 offsetVoltage = -((double)0x8000/10000)* offsetVoltage;
150 HWANIN_PARAM(analogInputNo, HWANIN_OFFSET)=offsetVoltage;
151
152 print("sdkScaleAnalogInput: ", analogInputNo);
153 return(1);
154 }
```

```
5.15.2.2 sdkSetupPwmGenerator_AxisModeVel() long sdkSetupPwmGenerator_AxisModeVel (
    long axisNo,
    long digOutEnable,
    long digOutPWM,
    long frequency,
    long cycleRange,
    long polarity )
```

Setting up a PWM output with a virtual amplifier as input source.

This function sets all important parameters for the use of a PWM output. As input a virtual amplifier is used, where the input source is interpreted as signed 16bit value. Afterwards, standard ApossC move commands such as AxisCvelStart() can be used. A feedback is not implemented with this function. Not all MACS controllers support PWM signals.

Parameters

<i>axisNo</i>	Axis module number digOutEnable Enable output number which is automatically set when an axis drive command is started (0-xx zero based)
<i>digOutPWM</i>	Digital output number for the PWM (0-xx zero based)
<i>frequency</i>	Frequency of the PWM output (default 1000)
<i>cycleRange</i>	Duty cycle range A value of 90% means a duty cycle range of 5-95 (default 90)
<i>polarity</i>	Polarity of the reference signal (default 0) 0 HWPWMGEN_POLARITY_POSITIVE: Output Signal is not inverted 1 HWPWMGEN_POLARITY_NEGATIVE: Output Signal is inverted

Returns

- value: Always 1 in this function
- value > 0 Process successful
- value = 0 Process is active
- value < 0 Error

Examples

```
PWM_1Ax_ESCON.mc.
```

File Documentation

Definition at line 74 of file SDK_Miscellaneous_IO.mc.

```

75 {
76     // To enable PWM generation, the HW Digital Output has to be switched to this mode
77     HWDIGOUT_PARAM(0,HWDIGOUT_PISRC_BIT1+digOutPWM)          = 0x400;
78
79     // Input value is an unsigned 16bit value
80     HWPWMGEN_PARAM(digOutPWM,HWPWMGEN_MODE)                  = HWPWMGEN_MODE_SIGNED;
81
82     // Polarity of the reference signal
83     HWPWMGEN_PARAM(digOutPWM,HWPWMGEN_POLARITY)              = polarity;
84
85     // Frequency of the PWM output
86     HWPWMGEN_PARAM(digOutPWM,HWPWMGEN_FREQUENCY)             = frequency;
87
88     // Duty cycle range and source
89     HWPWMGEN_PARAM(digOutPWM,HWPWMGEN_DUTYCYCLE_RANGE)       = cycleRange;
90     HWPWMGEN_PARAM(digOutPWM,HWPWMGEN_PISRC_DUTYCYCLE)       =
VIRTAMP_PROCESS_SRCINDEX(axisNo,PO_VIRTAMP_REFVEL);
91
92     // Set up the virtual amplifier for axis command
93     // Resolution +/- 0x8000 (signed 16 bit value)
94     VIRTAMP_PARAM(axisNo,VIRTAMP_REF100PERC)                  = 0x8000;
95     // Automatic setting of the enable output
96     VIRTAMP_PARAM(axisNo,VIRTAMP_REFOUTP)                     = digOutEnable+1;
97     VIRTAMP_PARAM(axisNo,VIRTAMP_REFOUTN)                     = digOutEnable+1;
98
99     // Because no feedback is processed, the tracking error must be switched off.
100    AXE_PARAM(axisNo,POSERR)      = 0x7FFFFFFF;
101
102    // Position controller is bypassed, the set speed is transferred directly
103    AXE_PARAM(axisNo,KPROP)        = 0;
104    AXE_PARAM(axisNo,KDER)         = 0;
105    AXE_PARAM(axisNo,KINT)         = 0;
106    AXE_PARAM(axisNo,FFVEL)        = 1000;
107
108    print("sdkSetupPwmGenerator_AxisModeVel:");
109    return(1);
110 }
```

5.15.2.3 sdkSetupPwmGenerator_UserParamMode() long sdkSetupPwmGenerator_UserParamMode (

 long *userParam*,

 long *digOutPWM*,

 long *frequency*,

 long *cycleRange*,

 long *polarity*)

Setting up a PWM output with a user parameter as input source.

This function sets all important parameters for the use of a PWM output. As input a UserParameter is used, where the input source is interpreted as unsigned 16bit value. Not all MACS controllers support PWM signals.

Parameters

<i>userParam</i>	User parameter number 1-100 which sets the PWM Input value is an unsigned 16bit value
<i>digOutPWM</i>	Digital output number for the PWM (0-xx zero based)

Parameters

<i>frequency</i>	Frequency of the PWM output (default 1000)
<i>cycleRange</i>	Duty cycle range A value of 90% means a duty cycle range of 5-95 (default 90)
<i>polarity</i>	Polarity of the reference signal (default 0) 0 HWPWMGEN_POLARITY_POSITIVE: Output Signal is not inverted 1 HWPWMGEN_POLARITY_NEGATIVE: Output Signal is inverted

Returns

value: Always 1 in this function
 value > 0 Process successful
 value = 0 Process is active
 value < 0 Error

Examples

[PWM_1Ax_ESCON.mc.](#)

Definition at line 33 of file SDK_Miscellaneous_IO.mc.

```

34 {
35     // To enable PWM generation, the HW Digital Output has to be switched to this mode
36     HWDIGOUT_PARAM(0, HWDIGOUT_PISRC_BIT1+digOutPWM)      = 0x400;
37
38     // Input value is an unsigned 16bit value
39     HWPWMGEN_PARAM(digOutPWM, HWPWMGEN_MODE)              = HWPWMGEN_MODE_UNSIGNED;
40
41     // Polarity of the reference signal
42     HWPWMGEN_PARAM(digOutPWM, HWPWMGEN_POLARITY)          = polarity;
43
44     // Frequency of the PWM output
45     HWPWMGEN_PARAM(digOutPWM, HWPWMGEN_FREQUENCY)        = frequency;
46
47     // Duty cycle range and source
48     HWPWMGEN_PARAM(digOutPWM, HWPWMGEN_DUTYCYCLE_RANGE)   = cycleRange;
49     HWPWMGEN_PARAM(digOutPWM, HWPWMGEN_PISRC_DUTYCYCLE)   = USER_PARAM_SRCINDEX(userParam);
50
51     print("sdkSetupPwmGenerator_UserParamMode:");
52     return(1);
53 }
```

5.16 SDK_Miscellaneous_Recording.mc File Reference

Functions with recording samples.

```

#include <SysDef.mh>
#include "SDK_Miscellaneous_Recording.mh"
```

- long [sdkRecordGeneralAxisParam_1Ax](#) (long axis)

Starts a recording of general axis parameter of a single axis.

- long [sdkRecordGeneralAxisParam_4Ax](#) (long axis1, long axis2, long axis3, long axis4)

Starts a recording of general axis parameter of four axes.

5.16.1 Detailed Description

Functions with recording samples.

Revision

139

5.16.2 Function Documentation

5.16.2.1 [sdkRecordGeneralAxisParam_1Ax\(\)](#) `long sdkRecordGeneralAxisParam_1Ax (long axis)`

Starts a recording of general axis parameter of a single axis.

This function records the nominal and actual position of the axis. Additionally the trackerror, speed and current output is recorded. The recording can be stopped during a desired time with the "RecordStop(0,0)" method.

Parameters

<i>axis</i>	Axis module number
-------------	--------------------

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Definition at line 25 of file SDK_Miscellaneous_Recording.mc.

26 {

```
27     RecordTime(1); // Sampling period in milliseconds.
28     RecordIndex( AXE_PROCESS_INDEX(axis,REG_COMPOS),
29                 AXE_PROCESS_INDEX(axis,REG_ACTPOS),
30                 AXE_PROCESS_INDEX(axis,REG_TRACKERR),
31                 AXE_PROCESS_INDEX(axis,REG_AVEL),
32                 HWAMP_PROCESS_INDEX(axis, PO_HWAMP_CURRENT)
33             );
34
35     RecordDest (DYNMEM); // DYNMEM for recording data into free memory.
36     RecordType(1); // 1 = cyclic recording
37     RecordStart(0); // Start recording
38
39     return(1);
40 }
```

5.16.2.2 sdkRecordGeneralAxisParam_4Ax() long sdkRecordGeneralAxisParam_4Ax (

long axis1,

long axis2,

long axis3,

long axis4)

Starts a recording of general axis parameter of four axes.

This function records the nominal and actual position of four axes. Additionally the trackerror, speed and current output is recorded. The recording can be stopped during a desired time with the "RecordStop(0,0)" method.

Parameters

<i>axis1</i>	Axis module number 1
<i>axis2</i>	Axis module number 2
<i>axis3</i>	Axis module number 3
<i>axis4</i>	Axis module number 4

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Definition at line 57 of file SDK_Miscellaneous_Recording.mc.

```
58 {
59     RecordTime(1); // Sampling period in milliseconds.
60     RecordIndex( AXE_PROCESS_INDEX(axis1,REG_COMPOS),
61                 AXE_PROCESS_INDEX(axis1,REG_ACTPOS),
62                 AXE_PROCESS_INDEX(axis1,REG_TRACKERR),
63                 AXE_PROCESS_INDEX(axis1,REG_AVEL),
64                 HWAMP_PROCESS_INDEX(axis1, PO_HWAMP_CURRENT),
65                 AXE_PROCESS_INDEX(axis2,REG_COMPOS),
66                 AXE_PROCESS_INDEX(axis2,REG_ACTPOS),
67                 AXE_PROCESS_INDEX(axis2,REG_TRACKERR),
```

File Documentation

```

68         AXE_PROCESS_INDEX(axis2,REG_AVEL),
69         HWAMP_PROCESS_INDEX(axis2, PO_HWAMP_CURRENT),
70         AXE_PROCESS_INDEX(axis3,REG_COMPOS),
71         AXE_PROCESS_INDEX(axis3,REG_ACTPOS),
72         AXE_PROCESS_INDEX(axis3,REG_TRACKERR),
73         AXE_PROCESS_INDEX(axis3,REG_AVEL),
74         HWAMP_PROCESS_INDEX(axis3, PO_HWAMP_CURRENT),
75         AXE_PROCESS_INDEX(axis4,REG_COMPOS),
76         AXE_PROCESS_INDEX(axis4,REG_ACTPOS),
77         AXE_PROCESS_INDEX(axis4,REG_TRACKERR),
78         AXE_PROCESS_INDEX(axis4,REG_AVEL),
79         HWAMP_PROCESS_INDEX(axis4, PO_HWAMP_CURRENT)
80     );
81
82     RecordDest (DYNMEM);           // DYNMEM for recording data into free memory.
83     RecordType(1);                // 1 = cyclic recording
84     RecordStart(1);               // Start recording
85
86     return(1);
87 }
```

5.17 SDK_Motion_Movement.mc File Reference

Functions for the simplification of movements.

```

#include <SysDef.mh>
#include "SDK_Motion_Movement.mh"
```

Functions

- long [sdkStartContinuousMove](#) (long axis, long vel, long acc)
Start a movement in continuous position mode.
- long [sdkStopContinuousMove](#) (long axis, long dec)
Stop a movement in continuous position mode.

5.17.1 Detailed Description

Functions for the simplification of movements.

Revision

39

5.17.2 Function Documentation

5.17.2.1 sdkStartContinuousMove() `long sdkStartContinuousMove (`
 `long axis,`
 `long vel,`
 `long acc)`

Start a movement in continuous position mode.

The function starts a movement with a given acceleration and speed. The axis must have been switched on before. The specified velocity value will be scaled by the parameters Maximum velocity VELMAX and Velocity Resolution VELRES to determine the final command velocity. See Vel for more information.

Parameters

<i>axis</i>	Axis module number
<i>vel</i>	Velocity value (negative value for reversing), Range: 1..VELRES, Command Velocity = vel * VELMAX/VELRES
<i>acc</i>	Scaled acceleration value. Range: 1..VELRES

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[Maxon_EC45_flat_1ax_BC_Enc.mc](#), [Maxon_EC45_flat_1ax_BC_Hall.mc](#), [Maxon_EC45_flat_1ax_SC_Hall.mc](#), [Maxon_ECi40_1ax_SC_OL_Inc.mc](#), [Maxon_ECi40_3ax_SC_OL_Inc.mc](#), [Maxon_ECi52_1ax_SC_SSI.mc](#), [Maxon_RE_40_1ax_Inc.mc](#), [Stepper_XY_CL.mc](#), and [Stepper_XY_OL.mc](#).

Definition at line 26 of file SDK_Motion_Movement.mc.

```
27 {  
28   Cvel(axis, vel);           // Set the velocity  
29   Acc(axis, acc);           // Set the acceleration  
30   AxisCvelStart(axis);      // Start constant velocity mode  
31  
32   return(1);  
33 }
```

5.17.2.2 sdkStopContinuousMove() `long sdkStopContinuousMove (`
 `long axis,`
 `long dec)`

Stop a movement in continuous position mode.

The function stops a constant velocity movement with a given deceleration.

<i>axis</i>	Axis module number
<i>dec</i>	Scaled deceleration value. Range: 1..VELRES

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Definition at line 45 of file SDK_Motion_Movement.mc.

```

46 {
47     Dec(axis, dec);           // Set the deceleration
48     AxisCvelStop(axis);      // Stop constant velocity mode
49
50     return(1);
51 }
```

5.18 SDK_SignalGenerator.mc File Reference

Functions for the operation of the signal generator.

```

#include <SysDef.mh>
#include "SDK_SignalGenerator.mh"
```

Macros

- #define [HWSIGGEN_MODE_ENABLE](#) 0x0001

Functions

- long [sdkSigGenAxisSetupMiniMACS6](#) (long axis, long encPortOut, long indexDistance)
Settings for a Signal Generator.
- long [sdkSigGenOpenloopSetupMiniMACS6](#) (long encPort, long indexDistance)
Settings for a Signal Generator.
- void [sdkSigGenEnable](#) (long encPort, long enable)
Enable/Disable Signal Generator.
- void [sdkSigGenReset](#) (long encPort)
Reset Signal Generator.
- void [sdkSigGenOpenloopAcceleration](#) (long encPort, long acceleration)
Set acceleration of Signal Generator.
- void [sdkSigGenOpenloopDeceleration](#) (long encPort, long deceleration)
Set acceleration of Signal Generator.
- void [sdkSigGenOpenloopVelocity](#) (long encPort, long velocity)
Set velocity of Signal Generator.

5.18.1 Detailed Description

Functions for the operation of the signal generator.

Revision

210

5.18.2 Macro Definition Documentation

5.18.2.1 HWSIGGEN_MODE_ENABLE `#define HWSIGGEN_MODE_ENABLE 0x0001`

Definition at line 16 of file SDK_SignalGenerator.mc.

5.18.3 Function Documentation

5.18.3.1 sdkSigGenAxisSetupMiniMACS6() `long sdkSigGenAxisSetupMiniMACS6 (
 long axis,
 long encPortOut,
 long indexDistance)`

Settings for a Signal Generator.

With this function the settings can be made which are required for a signal generator. This is a closed loop signal generator which can generate encoder signals by running a virtual axis. The signal generator can also generate index signals with a defined index distance. This signal will be generated on a digital output. For signal generator 0 this will be digital output 1, for signal generator 1 digital output 2 and so on. The digital output can then not be controlled by normal output functions anymore.

The generated signal cannot be read back. A encoder signal connection to a different port has to be made to receive the generated signals.

Parameters

<i>axis</i>	Axis number 0-n
<i>encPortOut</i>	Encoder port number. Usually, module instance 0 is connected to X1 and so on. Please refer to product manual
<i>indexDistance</i>	Distance in increments between two index signals.

File Documentation

Returns

value: Process value
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[SetupSignalGeneratorAxis.mc.](#)

Definition at line 41 of file SDK_SignalGenerator.mc.

```

42 {
43     long virtMasterNo = encPortOut;
44
45     if(SYS_INFO(SYS_MAX_SIGGEN) == 0)
46     {
47         print("Controller does not support Signal Generator");
48         return(-1);
49     }
50
51     if((encPortOut+1) > SYS_INFO(SYS_MAX_SIGGEN))
52     {
53         print("Port does not support Signal Generator");
54         return(-2);
55     }
56
57     if(SYS_INFO(SYS_MAX_VIRTMAS) == 0)
58     {
59         print("Controller does not support Virtual Masters");
60         return(-3);
61     }
62
63     if((encPortOut+1) > SYS_INFO(SYS_MAX_VIRTMAS))
64     {
65         print("Port does not support Virtual Masters");
66         return(-4);
67     }
68
69     // Setup selected encoder port as signal generator
70     HWENC_PARAM(encPortOut, HWENCODER_MODE) = HWENCODER_MODE_INCROUTPUT;    // signal generator mode
71
72     HWSIGGEN_PARAM(encPortOut, HWSIGGEN_SIGDIST) = indexDistance;          // setup index distance
73
74     VIRTMAS_PARAM(encPortOut, VIRTMAS_UUFACT_UNITNO) = 0xFFFFF;
75     // Full reference value of REG_REFERENCE is 0xFFFFF
76     // Virtual master works in 1/1000 Hz on each line (Spur)
77     // This results in factor 1000/4 = 250
78     // REG_REFERENCE is scaled with 65'536
79     // 250 % 65.536 = 3.814697265625
80     VIRTMAS_PARAM(encPortOut, VIRTMAS_UUFACT_INCN) = AXE_PARAM(axis, VELMAXQC) * 3.814697265625;
81
82     // set velocity source to axis cmdvel value
83     VIRTMAS_PARAM(encPortOut, VIRTMAS_PISRC_CMDVEL) = AXE_PROCESS_SRCINDEX(axis, REG_REFERENCE);
84     AXE_PARAM(axis, FFVEL) = 1000;
85     AXE_PARAM(axis, KPROP) = 100;
86     AXE_PARAM(axis, KDER) = 0;
87
88     // scale to 1/1000 HZ (Pulses)
89     VIRTMAS_PARAM(encPortOut, VIRTMAS_MODE) = VIRTMAS_MODE_VELOCITY;
90
91     HWSIGGEN_PARAM(encPortOut, HWSIGGEN_MODE) = HWSIGGEN_MODE_ENABLE;    // enable signal generator
92
93     return(1);
94 }

```

5.18.3.2 sdkSigGenEnable() `void sdkSigGenEnable (`
 `long encPort,`
 `long enable)`

Enable/Disable Signal Generator.

Disabling and Enabling the Signal Generator will reset the index distance counter. This means, an index signal will immediately generated. The next signal will be generated after index distance.

Parameters

<i>encPort</i>	Encoder port number. Usually, module instance 0 is connected to X1 and so on. Please refer to product manual
<i>enable</i>	1 = enable, 0 = disable

Examples

[SetupSignalGeneratorOpenloop.mc.](#)

Definition at line 167 of file SDK_SignalGenerator.mc.

```
168 {
169     if(enable)
170         HWSIGGEN_PARAM(encPort, HWSIGGEN_MODE) = HWSIGGEN_MODE_ENABLE;           // enable signal generator
171     else
172         HWSIGGEN_PARAM(encPort, HWSIGGEN_MODE) = 0;                               // disable signal
173     generator
174 }
```

5.18.3.3 sdkSigGenOpenloopAcceleration() `void sdkSigGenOpenloopAcceleration (`
 `long encPort,`
 `long acceleration)`

Set acceleration of Signal Generator.

Parameters

<i>encPort</i>	Encoder port number. Usually, module instance 0 is connected to X1 and so on. Please refer to product manual
<i>acceleration</i>	The acceleration of the generated signal [Hz/s]

Examples

[SetupSignalGeneratorOpenloop.mc.](#)

Definition at line 195 of file SDK_SignalGenerator.mc.

```
196 {
197     VIRTMAST_PARAM(encPort, VIRTMAST_ACC) = acceleration; // acc is in Hz/s
198 }
```

5.18.3.4 sdkSigGenOpenloopDeceleration()

void sdkSigGenOpenloopDeceleration (
long encPort,
long deceleration)

Set acceleration of Signal Generator.

Parameters

encPort	Encoder port number. Usually, module instance 0 is connected to X1 and so on. Please refer to product manual
acceleration	The acceleration of the generated signal [Hz/s]

Examples

[SetupSignalGeneratorOpenloop.mc.](#)

Definition at line 207 of file SDK_SignalGenerator.mc.

208 {
209 VIRTMAST_PARAM(encPort, VIRTMAST_DEC) = deceleration; // dec is in Hz/s
210 }

5.18.3.5 sdkSigGenOpenloopSetupMiniMACS6()

long sdkSigGenOpenloopSetupMiniMACS6 (
long encPort,
long indexDistance)

Settings for a Signal Generator.

With this function the settings can be made which are required for a signal generator. This is an open loop signal generator which can generate an encoder signal with a defined signal speed and and with a ramp of defined acceleration and deceleration. The signal generator can also generate index signals with a defined index distance. This signal will be generated on a digital output. For signal generator 0 this will be digital output 1, for signal generator 1 digital output 2 and so on. The digital output can then not be controlled by normal output functions anymore.

The generated signal cannot be read back. A encoder signal connection to a different port can be made to receive the generated signals.

Parameters

encPort	Encoder port number. Usually, module instance 0 is connected to X1 and so on. Please refer to product manual
indexDistance	Distance in increments between two index signals.

Returns

value: Process value
 value > 0 Process successful
 value = 0 Process is active
 value < 0 Error

Examples

[SetupSignalGeneratorOpenloop.mc.](#)

Definition at line 117 of file SDK_SignalGenerator.mc.

```

118 {
119     long virtMasterNo = encPort;
120
121     if(SYS_INFO(SYS_MAX_SIGGEN) == 0)
122     {
123         print("Controller does not support Signal Generator");
124         return(-1);
125     }
126
127     if((encPort+1) > SYS_INFO(SYS_MAX_SIGGEN))
128     {
129         print("Port does not support Signal Generator");
130         return(-2);
131     }
132
133     if(SYS_INFO(SYS_MAX_VIRTMAS) == 0)
134     {
135         print("Controller does not support Virtual Masters");
136         return(-3);
137     }
138
139     if((encPort+1) > SYS_INFO(SYS_MAX_VIRTMAS))
140     {
141         print("Port does not support Virtual Masters");
142         return(-4);
143     }
144
145     // Setup selected encoder port as signal generator
146     HWENC_PARAM(encPort, HWENCODER_MODE) = HWENCODER_MODE_INCROUTPUT; // signal generator mode
147
148     HWSIGGEN_PARAM(encPort, HWSIGGEN_SIGDIST) = indexDistance; // setup index distance
149
150     VIRTMAS_PARAM(encPort, VIRTMAS_UUFACT_UNITNO) = 1;
151     VIRTMAS_PARAM(encPort, VIRTMAS_UUFACT_INCNO) = 1;
152     VIRTMAS_PARAM(encPort, VIRTMAS_MODE) = VIRTMAS_MODE_PROFILE;
153
154     HWSIGGEN_PARAM(encPort, HWSIGGEN_MODE) = HWSIGGEN_MODE_ENABLE; // enable signal generator
155
156     return(1);
157 }

```

5.18.3.6 sdkSigGenOpenloopVelocity() void sdkSigGenOpenloopVelocity (

long encPort,
 long velocity)

Set velocity of Signal Generator.

<i>encPort</i>	Encoder port number. Usually, module instance 0 is connected to X1 and so on. Please refer to product manual
<i>velocity</i>	The velocity of the generated signal [Hz]

Examples

SetupSignalGeneratorOpenloop.mc.

Definition at line 219 of file SDK_SignalGenerator.mc.

```
220 {
221     VIRTMAST_PARAM(encPort, VIRTMAST_VEL) = velocity*1000; // vel is in 1/1000 Hz
222 }
```

5.18.3.7 sdkSigGenReset()

```
void sdkSigGenReset (
    long encPort )
```

Reset Signal Generator.

This function will reset the index distance counter. This means, an index signal will immediately generated. The next signal will be generated after index distance.

Parameters

<i>encPort</i>	Encoder port number. Usually, module instance 0 is connected to X1 and so on. Please refer to product manual
----------------	--

Examples

SetupSignalGeneratorOpenloop.mc.

Definition at line 182 of file SDK_SignalGenerator.mc.

```
183 {
184     HWSIGGEN_PARAM(encPort, HWSIGGEN_MODE) = 0;           // disable signal generator
185     HWSIGGEN_PARAM(encPort, HWSIGGEN_MODE) = HWSIGGEN_MODE_ENABLE; // enable signal generator
186 }
```

5.19 SDK VirtualModule AxisSetup.mc File Reference

Functions to work with virtual axis modules.

```
#include "SDK_VirtualModule_AxisSetup.mh"
```

5.19.1 Detailed Description

39

ApossC SDK | Software Reference
mzub | Edition 2024 - 12 | SDK V01.15

5.20 SDK_VirtualModule_MasterSetup.mc File Reference

Functions to work with a virtual master.

```
#include "SDK_VirtualModule_MasterSetup.mh"
```

Functions

- long [sdkSetupVirtualMasterMode](#) (long master, long mode)
Setup a virtual master.
- long [sdkSetupVirtualMasterAxisLink](#) (long master, long axis)
Set the virtual master as master of an axis.
- long [sdkSetupVirtualMasterScale](#) (long master, long numerator, long denominator)
Scale the input parameters of the virtual master.
- long [sdkSetVirtualMasterProfile](#) (long master, long acc, long dec)
Set a virtual master profile.
- long [sdkStartVirtualMasterProfile](#) (long master, long vel)
Starts a virtual master in profile mode.
- long [sdkStopVirtualMasterProfile](#) (long master)
Stops a virtual master in profile mode.

5.20.1 Detailed Description

Functions to work with a virtual master.

Revision

155

5.20.2 Function Documentation

5.20.2.1 [sdkSetupVirtualMasterAxisLink\(\)](#) long [sdkSetupVirtualMasterAxisLink](#) (
 long *master*,
 long *axis*)

Set the virtual master as master of an axis.

Set the virtual master as master of an axis.

<i>master</i>	Master module number
<i>axis</i>	Axis module number

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[VirtualMaster_ProfileMode.mc.](#)

Definition at line 45 of file SDK_VirtualModule_MasterSetup.mc.

```
46 {
47
48     // Axis master module = axis module number + max axis module
49     VIRTCONTIN_PARAM(axis + SYS_PROCESS(SYS_MAXAX), VIRTCONTIN_PISRC_COUNTER) =
        VIRTMAST_PROCESS_SRCINDEX(master, PO_VIRTMAST_POS);
50
51     return(1);
52 }
```

5.20.2.2 sdkSetupVirtualMasterMode() long sdkSetupVirtualMasterMode (
long master,
long mode)

Setup a virtual master.

The virtual master module generates a virtual speed and position signal. There are two modes to operate the module (see VIRTMAST_MODE):

Parameters

<i>master</i>	Master module number
<i>mode</i>	Axis module number 0 VIRTMAST_MODE_DISABLED: Disabled, output velocity is zero 1 VIRTMAST_MODE_VELOCITY: Velocity mode, PO_VIRTMAST_VEL is taken directly from VIRTMAST_PISRC_CMDVEL 3 VIRTMAST_MODE_PROFILE: Velocity profile mode, PO_VIRTMAST_VEL is generated using VIRTMAST_VEL/ACC/DEC

File Documentation

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[VirtualMaster_ProfileMode.mc.](#)

Definition at line 28 of file SDK_VirtualModule_MasterSetup.mc.

```
29 {
30     VIRTMAST_PARAM(master,VIRTMAST_MODE)    = mode;
31
32     return (1);
33 }
```

5.20.2.3 sdkSetupVirtualMasterScale() long sdkSetupVirtualMasterScale (

 long *master*,

 long *numerator*,

 long *denominator*)

Scale the input parameters of the virtual master.

This function converts user units into increments (increments * (numerator/denominator))
The parameters VIRTMAST_VEL, VIRTMAST_ACC, VIRTMAST_DEC, VIRTMAST_PISRC_CMDVEL
and the output PO_VIRTMAST_VEL are changed according to this scaling.

Parameters

<i>master</i>	Master module number
<i>numerator</i>	User factor numerator to scale to increments
<i>denominator</i>	User factor denominator to scale to increments

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[VirtualMaster_ProfileMode.mc.](#)

Definition at line 67 of file SDK_VirtualModule_MasterSetup.mc.

```
68 {
69     VIRTMAST_PARAM(master,VIRTMAST_UUFACT_INCNO) = numerator;
```

```

5.20.2.4 sdkSetVirtualMasterProfile()  long sdkSetVirtualMasterProfile (
    long master,
    long acc,
    long dec )

```

Only if the virtual master is in profile mode (`sdkSetupVirtualMasterMode()`).

Parameters

Returns

Examples

Definition at line 88 of file SDK_VirtualModule_MasterSetup.mc.

5.20.2.5 sdkStartVirtualMasterProfile()

```
long sdkStartVirtualMasterProfile (
    long master,
    long vel )
```

154

File Documentation

Only if the virtual master is in profile mode ([sdkSetupVirtualMasterMode\(\)](#)).

Starts a virtual master in profile mode, where the speed is transferred. The scaling can be adjusted with the [sdkSetupVirtualMasterScale\(\)](#) function.

Parameters

<i>master</i>	Master module number
<i>vel</i>	Target velocity [uu/ms]

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[VirtualMaster_ProfileMode.mc.](#)

Definition at line 108 of file SDK_VirtualModule_MasterSetup.mc.

```

109 {
110     VIRTMAST_PARAM(master,VIRTMAST_VEL)    = vel;
111
112     return(1);
113 }
```

5.20.2.6 sdkStopVirtualMasterProfile() long sdkStopVirtualMasterProfile (
long master)

Stops a virtual master in profile mode.

Only if the virtual master is in profile mode ([sdkSetupVirtualMasterMode\(\)](#)).
Stops a virtual master in profile mode.

Parameters

<i>master</i>	Master module number
---------------	----------------------

Returns

value: Always 1 in this function
value > 0 Process successful
value = 0 Process is active
value < 0 Error

Examples

[VirtualMaster_ProfileMode.mc.](#)

Definition at line 124 of file SDK_VirtualModule_MasterSetup.mc.

File Documentation

```
125 {  
126     VIRTMAST_PARAM(master,VIRTMAST_VEL)    = 0;  
127  
128     return(1);  
129 }
```

5.21 zub_About.txt File Reference

Internal file to generate the about description.

5.21.1 Detailed Description

Internal file to generate the about description.

5.22 zub_ChangeLog.txt File Reference

Internal file to generate the changelog.

5.22.1 Detailed Description

Internal file to generate the changelog.

5.23 zub_HowToUse.txt File Reference

Internal file to generate the how to use description.

5.23.1 Detailed Description

Internal file to generate the how to use description.

6.1 CAN_1Ax_DS402_ProfilePositionMode-ppm-Test.mc

```
#include "..\..\..\SDK\SDK_ApossC.mc"
// Bus settings
#define SLAVE_CAN_BUSID 1
void errorHandler();
void faultMonitoring();
long main(void)
{
    long handle;
    long actState=0;
    //Reset CAN Bus 1
    handle = DefCanOut(0, 2);
    CanOut(handle, 0, 0x8100);
    //Reset CAN Bus 2
    handle = DefCanOut(100000, 2);
    CanOut(handle, 0, 0x8100);
    //Wait until the node has been resetted
    Delay(2000);
    ErrorClear();
    //set errorHandler for master
    InterruptSetup(ERROR, errorHandler);
    //Monitor the Fault bit in the status word of the slave, all 25ms
    InterruptSetup(PERIOD, faultMonitoring, 25);
    //-----
    // Application Setup
    //-----
    //Set the mode of operation to PPM
    sdkDS402_SetOpationMode(SLAVE_CAN_BUSID, DS402_OP_PPM);
    actState = sdkDS402_GetActDriveState(SLAVE_CAN_BUSID);
    if(actState == DS402_DRIVE_STATE_SWITCH_ON_DISABLED)
    {
        sdkDS402_WaitTransitionToState(SLAVE_CAN_BUSID, DS402_DRIVE_STATE_READY_TO_SWITCH_ON, -1);
    }
    //Set acceleration, deceleration and velocity
    sdkDS402_SetProfileMovementParameter(SLAVE_CAN_BUSID, 1000, 1000, 500);
    //Set the deceleration for quick stop
    sdkDS402_SetQuickStopDeceleration(SLAVE_CAN_BUSID, 5000);
    //-----
    // End of Application Setup
    //-----
    //Transition to drive state 'Operation enabled'
    sdkDS402_TransitionToState(SLAVE_CAN_BUSID, DS402_DRIVE_STATE_OPERATION_ENABLED);
    Delay(2);
    sdkDS402_Print_StatusWord(SLAVE_CAN_BUSID);
    while(1)
    {
        //Start absolute positioning to pos 100000 qc
        sdkDS402_PPM_PosAbsStart(SLAVE_CAN_BUSID, 100000, DS402_PPM_IMMEDIATELY);
        //Wait until the target position has been reached
        sdkDS402_PPM_WaitTargetReached(SLAVE_CAN_BUSID, -1);
        //Start relative positioning of -100000 qc
        sdkDS402_PPM_PosRelStart(SLAVE_CAN_BUSID, -100000, DS402_PPM_IMMEDIATELY);
        //Wait until the target position has been reached
        sdkDS402_PPM_WaitTargetReached(SLAVE_CAN_BUSID, -1);
    }
    return(1);
}
void faultMonitoring()
{
    long statusWord = sdkDS402_ReadStatusWord(SLAVE_CAN_BUSID);
    if(statusWord.i[DS402_SW_BIT_FAULT])
    {
        errorHandler();
    }
}
void errorHandler()
```

Example Documentation

```
{
    long axeNbr      = ErrorAxis();
    long errNbr      = ErrorNo();
    long errInfoNbr = ErrorInfo();
    long statusWord, sdoAbortCode;
    //Stop all nodes
    sdkDS402_QuickStop(SLAVE_CAN_BUSID);
    print("-----");
    printf("Error %d: ",errNbr);
    sdkErrorPrint_ApossIdeErrorDescription(errNbr);
    print();
    print("...Info: ",errInfoNbr, " AxisNo: ", axeNbr);
    print("-----");
    print();
    switch(errNbr)
    {
        case F_CANIO:
            sdoAbortCode = SYS_PROCESS(SYS_CANOM_SDOABORT);
            printf("SDO Abort Code 0x%lX: ", sdoAbortCode );
            sdkErrorPrint_SdoErrorDescription(SYS_PROCESS(SYS_CANOM_SDOABORT));
            print();
            print("Check Can baudrate & Can bus id");
        }
    if(axeNbr == 0xFF && errNbr != F_CANIO)
    {
        //The error is not releated to a certain axis
        statusWord = sdkDS402_ReadStatusWord(SLAVE_CAN_BUSID);
        if(statusWord.i[DS402_SW_BIT_FAULT])
        {
            print();
            printf("Bus Id %d, Slave device is in fault state\n", SLAVE_CAN_BUSID);
            if(statusWord.i[DS402_SW_PPM_BIT_FOLLOWING_ERROR])
            {
                print("...Following Error");
            }
            //Reset the fault
            sdkDS402_ResetFault(SLAVE_CAN_BUSID);
        }
    }
    ErrorClear();
    print();
    print(" There is no error handlig → Exit()");
    print("-----");
    Exit(0);
}
```

6.2 CAN_1Ax_DS402_ProfileVelocityMode-pvm-Test.mc

```
#include "..\..\..\SDK\SDK_ApossC.mc"
// Bus settings
#define SLAVE_CAN_BUSID 1
long waitTransitionToState(long busId, long targetState, long timeout);
void errorHandler();
void faultMonitoring();
long main(void)
{
    long handle, i;
    long actState=0;
    //Reset CAN Bus 1
    handle = DefCanOut(0, 2);
    CanOut(handle, 0, 0x8100);
    //Reset CAN Bus 2
    handle = DefCanOut(100000, 2);
    CanOut(handle, 0, 0x8100);
    //Wait until the node has been resetted
    Delay(2000);
    ErrorClear();
    //set errorHandler for master
```

```

InterruptSetup(ERROR, errorHandler);
//Monitor the Fault bit in the status word of the slave, all 25ms
InterruptSetup(PERIOD, faultMonitoring, 25);
//-----
// Application Setup
//-----
//Set the mode of operation to PVM
sdkDS402_SetOperationMode(SLAVE_CAN_BUSID, DS402_OP_PVM);
//Get the actual DS402 state machine state, and set it
actState = sdkDS402_GetActDriveState(SLAVE_CAN_BUSID);
if(actState == DS402_DRIVE_STATE_SWITCH_ON_DISABLED)
{
    //change the state to 'Ready to switch on', and verify the the state was achieved with infinite
    timeout (-1)
    sdkDS402_WaitTransitionToState(SLAVE_CAN_BUSID, DS402_DRIVE_STATE_READY_TO_SWITCH_ON, -1);
}
//Set acceleration, deceleration
sdkDS402_SetProfileAcceleration(SLAVE_CAN_BUSID, 2000);
sdkDS402_SetProfileDeceleration(SLAVE_CAN_BUSID, 2000);
//Set the deceleration for quick stop
sdkDS402_SetQuickStopDeceleration(SLAVE_CAN_BUSID, 5000);
//-----
// End of Application Setup
//-----
//Transition to drive state 'Operation enabled'
sdkDS402_WaitTransitionToState(SLAVE_CAN_BUSID, DS402_DRIVE_STATE_OPERATION_ENABLED, -1);
sdkDS402_Print_StatusWord(SLAVE_CAN_BUSID);
Delay(1000);
for(i = 0; i < 2; i++)
{
    //Set target velocity to 200rpm
    sdkDS402_PVM_SetTargetVelocity(SLAVE_CAN_BUSID, 200);
    //Start continuous velocity movement
    sdkDS402_PVM_CvelStart(SLAVE_CAN_BUSID);
    //Wait until target velocity is reached
    sdkDS402_PVM_WaitTargetReached(SLAVE_CAN_BUSID, -1);
    Delay(1500);
    //Set target velocity to 200rpm
    sdkDS402_PVM_SetTargetVelocity(SLAVE_CAN_BUSID, 100);
    //Start continuous velocity movement
    sdkDS402_PVM_CvelStart(SLAVE_CAN_BUSID);
    //Wait until target velocity is reached
    sdkDS402_PVM_WaitTargetReached(SLAVE_CAN_BUSID, -1);
    Delay(1500);
    //Stop
    sdkDS402_Halt(SLAVE_CAN_BUSID);
    Delay(500);
    //Set target velocity to 500rpm
    sdkDS402_PVM_SetTargetVelocity(SLAVE_CAN_BUSID, 500);
    //Start continuous velocity movement
    sdkDS402_PVM_CvelStart(SLAVE_CAN_BUSID);
    //Wait until target velocity is reached
    sdkDS402_PVM_WaitTargetReached(SLAVE_CAN_BUSID, -1);
    Delay(1500);
    //Stop continuous velocity movement
    sdkDS402_PVM_CvelStop(SLAVE_CAN_BUSID);
    Delay(500);
}
//Set target velocity to 1000rpm
sdkDS402_PVM_SetTargetVelocity(SLAVE_CAN_BUSID, 1000);
//Start continuous velocity movement
sdkDS402_PVM_CvelStart(SLAVE_CAN_BUSID);
//Wait until target velocity is reached
sdkDS402_PVM_WaitTargetReached(SLAVE_CAN_BUSID, -1);
Delay(2000);
//Quick Stop
sdkDS402_QuickStop(SLAVE_CAN_BUSID);
return(1);
}
void faultMonitoring()

```

Example Documentation

```
{
    long statusWord = sdkDS402_ReadStatusWord(SLAVE_CAN_BUSID);
    if(statusWord.i[DS402_SW_BIT_FAULT])
    {
        errorHandler();
    }
}

void errorHandler()
{
    long axeNbr      = ErrorAxis();
    long errNbr      = ErrorNo();
    long errInfoNbr = ErrorInfo();
    long statusWord, idIndex, sdoAbortCode;
    //Stop all nodes
    sdkDS402_QuickStop(SLAVE_CAN_BUSID);
    print("-----");
    printf("Error %d: ",errNbr);
    sdkErrorPrint_ApossIdeErrorDescription(errNbr);
    print();
    print("...Info: ",errInfoNbr, " AxisNo: ", axeNbr);
    print("-----");
    print();
    switch(errNbr)
    {
        case F_CANIO:
            printf("SDO Abort Code 0x%X: ", sdoAbortCode );
            sdkErrorPrint_SdoErrorDescription(SYS_PROCESS(SYS_CANOM_SDOABORT));
            print();
            print("Check Can baudrate & Can bus id");
        }
    if(axeNbr == 0xFF && errNbr != F_CANIO)
    {
        //The error is not releated to a certain axis
        statusWord = sdkDS402_ReadStatusWord(SLAVE_CAN_BUSID);
        if(statusWord.i[DS402_SW_BIT_FAULT])
        {
            print();
            printf("Bus Id %d, Slave device is in fault state\n", SLAVE_CAN_BUSID);
            if(statusWord.i[DS402_SW_PPM_BIT_FOLLLWING_ERROR])
            {
                print("...Following Error");
            }
            //Reset the fault
            sdkDS402_ResetFault(SLAVE_CAN_BUSID);
        }
    }
    ErrorClear();
    print();
    print(" There is no error handlig → Exit()");
    print("-----");
    Exit(0);
}
```

6.3 CAN_1Ax_EPOS4-Test_csp.mc

```
#include "..\..\..\SDK\SDK_ApossC.mc"
// EPOS4 Settings
#define C_AXIS1          0           // Axis module number
#define C_DRIVE_BUSID1   1           // The CAN drive busId
#define C_PDO_NUMBER     1           // Used PDO number
#define C_AXISPOLARITY   0           // Definition of the polarity 0: Normal, 1: Inverse
// Encoder settings & axis user units (MACS)
#define C_AXIS_ENCRESES   4*4096      // Resolution of the encoder for position feed
    back in increments (quadcounts)
#define C_AXIS_POSENCREV  1           // Number of revolutions of the motor
#define C_AXIS_POSENCQC   C_AXIS_ENCRESES // Number of quadcounts in POSENCREV
    revolutions
#define C_AXIS_POSFACT_Z  1           // Number of revolutions of the input shaft
```

```

#define C_AXIS_POSFACT_N      1                      // Number of revolutions of the output shaft
    in POSFACT_Z revolutions of the input shaft
#define C_AXIS_FEEDREV      1                      // Number of revolutions of the gear box
    output shaft
#define C_AXIS_FEEDDIST      C_AXIS_ENCRESP        // Distance travelled (in user units) in
    FEEDREV revolutions of the gear box output shaft [mm]
// Axis Movement Parameter
#define C_AXIS_MAX_RPM      4000                  // Maximum velocity in RPM
#define C_AXIS_VELRES      100                  // Velocity resolution, Scaling used for the
    velocity and acceleration/deceleration commands, default
#define C_AXIS_RAMPTYPE      RAMPTYPE_JERKLIMITED // Defines the ramptype
#define C_AXIS_RAMPMIN      800                  // Maximum acceleration
#define C_AXIS_JERKMIN      1000                 // Minimum time (ms) required before reaching the
    maximum acceleration
#define C_AXIS_TRACKERR      0                    // There is also a following error on EPOS4, could
    be set to zero on the MACS
// Axis MACS control loop settings
// the position controller of the MACS control is active, but the values are all set to 0.
#define C_AXIS_KPROP      0
#define C_AXIS_KINT      0
#define C_AXIS_KDER      0
#define C_AXIS_KILIM      0
#define C_AXIS_KILIMTIME  0
#define C_AXIS_BANDWIDTH  1000
#define C_AXIS_FFVEL      1000
#define C_AXIS_KFFAC      0
#define C_AXIS_KFFDEC      0
long main(void) {
    long i, homingState=0, retVal;
    print("-----");
    print(" Test application CANopen Master with 1 EPOS4 drive");
    print("-----");
    ErrorClear();
    AmpErrorClear(C_AXIS1);
    InterruptSetup(ERROR, ErrorHandler);
    //-----
    // Application Setup
    //-----
    if (GLB_PARAM(CANBAUD) != 88)
    {
        print("Set new Baudrate and save global parameters");
        // Set Baudrate of CAN 1 & CAN 2 to 1MBaud
        GLB_PARAM(CANBAUD)=88;
        CanOpenRestart();
        Save(GLBPARS);
    }
    //while(1);
    // Cycle time for sending SYNC telegrams on the CAN bus - must be set before calling the
    sdkEpos4_SetupCanSdoParam() function.
    GLB_PARAM(CANSYNCTIMER)= 1;
    // set all slaves to PREOPERATIONAL by sending an NMT.
    SYS_PROCESS(SYS_CANOM_MASTERSTATE) = 0;
    // initialising maxon drives
    sdkEpos4_SetupCanSdoParam(C_DRIVE_BUSID1, C_PDO_NUMBER, C_AXISPOLARITY, EPOS4_OP_CSP);
    // setup CANopen bus module for csp mode
    sdkEpos4_SetupCanBusModule(C_AXIS1, C_DRIVE_BUSID1, C_PDO_NUMBER, EPOS4_OP_CSP);
    // setup virtual amplifier for csp mode
    sdkEpos4_SetupCanVirtAmp(C_AXIS1, C_AXIS_MAX_RPM, EPOS4_OP_CSP);
    // setup irtual counter for csp mode
    sdkEpos4_SetupCanVirtCntin(C_AXIS1, EPOS4_OP_CSP);
    // start all slaves commanding them into OPERATIONAL.
    SYS_PROCESS(SYS_CANOM_MASTERSTATE) = 1;
    // Movement parameters for the axis
    sdkSetupAxisMovementParam( C_AXIS1,
        C_AXIS_VELRES,
        C_AXIS_MAX_RPM,
        C_AXIS_RAMPTYPE,
        C_AXIS_RAMPMIN,
        C_AXIS_JERKMIN,
        C_AXIS_TRACKERR

```

Example Documentation

```

    );
// Definition of the user units
sdkSetupAxisUserUnits(    C_AXIS1,
                          C_AXIS_POSENCREV,
                          C_AXIS_POSENCQC,
                          C_AXIS_POSFACT_Z,
                          C_AXIS_POSFACT_N,
                          C_AXIS_FEEDREV,
                          C_AXIS_FEEDDIST
                          );

// Position control setup
sdkSetupPositionPIDControlExt(    C_AXIS1,
                                  C_AXIS_KPROP,
                                  C_AXIS_KINT,
                                  C_AXIS_KDER,
                                  C_AXIS_KILIM,
                                  C_AXIS_KILIMTIME,
                                  C_AXIS_BANDWIDTH,
                                  C_AXIS_FFVEL,
                                  C_AXIS_KFFAC,
                                  C_AXIS_KFFDEC
                                  );

//-----
// End of Application Setup
//-----
// Homing setup
print("\nEPOS4 Homing:");
SdoWrite( C_DRIVE_BUSID1, EPOS4_HOMING_METHOD,    0,    37);    // 0x6098 Set homing method to
    "-4" : Homing Method -4 (Current Threshold Negative Speed)."
```

Homing Speed / Speed for switch speed [velocity units]

```

SdoWrite( C_DRIVE_BUSID1, EPOS4_HOMING_SPEEDS,    1,    20);    //
Homing Speed / Speed for zero search [velocity units]
SdoWrite( C_DRIVE_BUSID1, EPOS4_HOMING_SPEEDS,    2,    20);    //
Homing acceleration [acceleration units]
SdoWrite( C_DRIVE_BUSID1, EPOS4_HOMING_ACCELERATION,    0,    20);    //
Home offset move distance [position units]
SdoWrite( C_DRIVE_BUSID1, EPOS4_HOME_OFFSET_MOVE_DISTANCE,    0,    6400);    //
Home position [position units]
SdoWrite( C_DRIVE_BUSID1, EPOS4_HOME_POSITION,    0,    0);    //
Current threshold for homing mode [mA]
SdoWrite( C_DRIVE_BUSID1, EPOS4_CURRENT_THRESHOLD_FOR_HOMING_MODE,    0,    1500);    //
// Homing statemachine
retval=0;
while(! retval)
{
    retval = sdkEpos4_AxisHomingStart(C_AXIS1, C_DRIVE_BUSID1, EPOS4_OP_CSP, homingState);
    // Homing error - Exit programm
    if(retval==-1)
    {
        print("Exit programm");
        Exit(0);
    }
}
print("");
print("-----");
print("                Movement in CSP Mode                ");
print("----- \n");
Vel(C_AXIS1, 50);
Acc(C_AXIS1, 30);
Dec(C_AXIS1, 30);
AxisControl(C_AXIS1, ON);
for(i=10;i>=0;i--)
{
    print("Start, move to target position");
    AxisPosAbsStart(C_AXIS1, 20000);
    AxisWaitReached(C_AXIS1);
    print("Target position is reached \n");
    print("Start, back to start position");
    AxisPosAbsStart(C_AXIS1, 0);
    AxisWaitReached(C_AXIS1);
}

```

```

        print("Start position is reached");
        print(i, " repetitions to go \n");
    }
    AxisControl(C_AXIS1, OFF);
    print("Program done, Axis OFF ");
    return(0);
}

void ErrorHandler(void)
{
    long axeNbr      = ErrorAxis();
    long errNbr      = ErrorNo();
    long errInfoNbr = ErrorInfo();
    long eposErr, sdoAbortCode;
    AxisControl(AXALL, OFF);
    switch(errNbr)
    {
        case F_AMP:      if( axeNbr==C_AXIS1)
                        {
                            eposErr = SdoRead(C_DRIVE_BUSID1,EPOS4_ERROR_CODE,0x00);
                            printf("Error Axis:  %d, Epos4 Error 0x%X: ", axeNbr , eposErr);
                            sdkEpos4_PrintErrorDescription(eposErr);
                            print();
                            AmpErrorClear(axeNbr); // Clear error on EPOS4
                        }
                        else
                        {
                            print("ErrorNo:  ",errNbr," info:  ",errInfoNbr, " AxisNo:  ", axeNbr);
                        }
                        break;
        case F_CANIO:    print("ErrorNo:  ",errNbr," info:  ",errInfoNbr);
                        sdoAbortCode = SYS_PROCESS(SYS_CANOM_SDOABORT);
                        printf("SDO Abort Code 0x%X: ", sdoAbortCode );
                        sdkErrorPrint_SdoErrorDescription(sdoAbortCode);
                        print();
                        print("Check Can baudrate & Can bus id");
                        break;
        default:         print("ErrorNo:  ",errNbr," info:  ",errInfoNbr, " AxisNo:  ", axeNbr);
    }
    ErrorClear();
    print(""); print(" There is no error handlig → Exit()");
    Exit(0);
}

```

6.4 CAN_1Ax_EPOS4-Test_cst.mc

```

#include "..\..\..\SDK\SDK_ApossC.mc"
// Parameters for the SDK function
#define C_AXIS1          0           // Axis module number
#define C_DRIVE_BUSID1  1           // The CAN drive busId
#define C_PDO_NUMBER    1           // Used PDO number
#define C_AXISPOLARITY   0           // Definition of the polarity 0: Normal, 1: Inverse
// Encoder settings & axis user units (MACS)
#define C_AXIS_ENCRESES  4*1600      // Resolution of the encoder for position feed
                                     back in increments (quadcounts)
#define C_AXIS_POSENCREV 1           // Number of revolutions of the motor
#define C_AXIS_POSENCQC  C_AXIS_ENCRESES // Number of quadcounts in POSENCREV
                                     revolutions
#define C_AXIS_POSFACT_Z  1           // Number of revolutions of the input shaft
#define C_AXIS_POSFACT_N  1           // Number of revolutions of the output shaft
                                     in POSFACT_Z revolutions of the input shaft
#define C_AXIS_FEEDREV    1           // Number of revolutions of the gear box
                                     output shaft
#define C_AXIS_FEEDDIST   C_AXIS_ENCRESES // Distance travelled (in user units) in
                                     FEEDREV revolutions of the gear box output shaft [mm]
// Axis Movement Parameter
#define C_AXIS_MAX_RPM    2000        // Maximum velocity in RPM
#define C_AXIS_VELRES     100         // Velocity resolution, Scaling used for the
                                     velocity and acceleration/deceleration commands, default

```

Example Documentation

```
#define C_AXIS_RAMPTYPE          RAMPTYPE_JERKLIMITED // Defines the ramptype
#define C_AXIS_RAMPMIN          800 // Maximum acceleration
#define C_AXIS_JERKMIN          1000 // Minimum time (ms) required before reaching the
    maximum acceleration
#define C_AXIS_TRACKERR          0 // There is also a following error on EPOS4, could
    be set to zero on the MACS
// Axis MACS control loop settings
// MACS position control is not used
#define C_AXIS_KPROP            0
#define C_AXIS_KINT             0
#define C_AXIS_KDER             0
#define C_AXIS_KILIM            0
#define C_AXIS_KILIMTIME        0
#define C_AXIS_BANDWIDTH        1000
#define C_AXIS_FFVEL            1000
#define C_AXIS_KFFAC            0
#define C_AXIS_KFFDEC           0
long setCur=0;
long main(void) {
    long i, homingState=0, retVal;
    print("-----");
    print(" Test application CANopen Master with 1 EPOS4 drive");
    print("-----");
    ErrorClear();
    AmpErrorClear(C_AXIS1);
    InterruptSetup(ERROR, ErrorHandler);
    //-----
    // Application Setup
    //-----
    if(GLB_PARAM(CANBAUD)!=88)
    {
        print("Set new Baudrate and save global parameters");
        // Set Baudrate of CAN 1 & CAN 2 to 1MBaud
        GLB_PARAM(CANBAUD)=88;
        CanOpenRestart();
        Save(GLBPARS);
    }
    // Cycle time for sending SYNC telegrams on the CAN bus - must be set before calling the
    sdkEpos4_SetupCanSdoParam() function.
    GLB_PARAM(CANSYNCTIMER)= 1;
    // set all slaves to PREOPERATIONAL by sending an NMT.
    SYS_PROCESS(SYS_CANOM_MASTERSTATE) = 0;
    // initialising maxon drives
    sdkEpos4_SetupCanSdoParam(C_DRIVE_BUSID1, C_PDO_NUMBER, C_AXISPOLARITY, EPOS4_OP_CST);
    // setup CANopen bus module for csp mode
    sdkEpos4_SetupCanBusModule(C_AXIS1, C_DRIVE_BUSID1, C_PDO_NUMBER, EPOS4_OP_CST);
    // setup virtual amplifier for csp mode
    sdkEpos4_SetupCanVirtAmp(C_AXIS1, C_AXIS_MAX_RPM, EPOS4_OP_CST);
    // setup irtual counter for csp mode
    sdkEpos4_SetupCanVirtCntin(C_AXIS1, EPOS4_OP_CST);
    // start all slaves commanding them into OPERATIONAL.
    SYS_PROCESS(SYS_CANOM_MASTERSTATE) = 1;
    // Movement parameters for the axis
    sdkSetupAxisMovementParam( C_AXIS1,
        C_AXIS_VELRES,
        C_AXIS_MAX_RPM,
        C_AXIS_RAMPTYPE,
        C_AXIS_RAMPMIN,
        C_AXIS_JERKMIN,
        C_AXIS_TRACKERR
    );
    // Definition of the user units
    sdkSetupAxisUserUnits( C_AXIS1,
        C_AXIS_POSENCREV,
        C_AXIS_POSENCQC,
        C_AXIS_POSFACT_Z,
        C_AXIS_POSFACT_N,
        C_AXIS_FEEDREV,
        C_AXIS_FEEDDIST
    );
}
```

```
// Position control setup
sdkSetupPositionPIDControlExt(    C_AXIS1,
                                   C_AXIS_KPROP,
                                   C_AXIS_KINT,
                                   C_AXIS_KDER,
                                   C_AXIS_KILIM,
                                   C_AXIS_KILIMTIME,
                                   C_AXIS_BANDWIDTH,
                                   C_AXIS_FFVEL,
                                   C_AXIS_KFFAC,
                                   C_AXIS_KFFDEC
                                   );

//-----
// End of Application Setup
//-----
// Homing setup
print("\nEPOS4 Homing:");
SdoWrite( C_DRIVE_BUSID1, EPOS4_HOMING_METHOD,          0,    EPOS4_HOMING_CURRENT_N_SPEED); //
0x6098 Set homing method to "-4" : Homing Method -4 (Current Threshold Negative Speed)."
SdoWrite( C_DRIVE_BUSID1, EPOS4_HOMING_SPEEDS,          1,    20); //
Homing Speed / Speed for switch speed [velocity units]
SdoWrite( C_DRIVE_BUSID1, EPOS4_HOMING_SPEEDS,          2,    20); //
Homing Speed / Speed for zero search [velocity units]
SdoWrite( C_DRIVE_BUSID1, EPOS4_HOMING_ACCELERATION,     0,    20); //
Homing acceleration [acceleration units]
SdoWrite( C_DRIVE_BUSID1, EPOS4_HOME_OFFSET_MOVE_DISTANCE, 0,    6400); //
Home offset move distance [position units]
SdoWrite( C_DRIVE_BUSID1, EPOS4_HOME_POSITION,          0,    0); //
Home position [position units]
SdoWrite( C_DRIVE_BUSID1, EPOS4_CURRENT_THRESHOLD_FOR_HOMING_MODE, 0,    1500); //
Current threshold for homing mode [mA]
// Homing statemachine
retval=0;
while(! retval)
{
    retval = sdkEpos4_AxisHomingStart(C_AXIS1, C_DRIVE_BUSID1, EPOS4_OP_CST, homingState);
    // Homing error - Exit programm
    if(retval==--1)
    {
        print("Exit programm");
        Exit(0);
    }
}
print("");
print("-----");
print("                Set torque in CST Mode                ");
print("----- \n");
AxisControl(C_AXIS1, ON);
for(i=10;i>=0;i--)
{
    // The value is given in per thousand of "Motor rated torque"
    print("Set target torque → positive");
    AXE_PROCESS(C_AXIS1,REG_USERREFCUR)= 250;
    Delay(4000);
    print("Set target torque → negative");
    AXE_PROCESS(C_AXIS1,REG_USERREFCUR)= -250;
    Delay(4000);
    print("Set target torque → zero");
    AXE_PROCESS(C_AXIS1,REG_USERREFCUR)= 0;
    Delay(4000);
    print(i, " repetitions to go \n");
}
AxisControl(C_AXIS1, OFF);
print("Program done, Axis OFF ");
return(0);
}

void ErrorHandler(void)
{
    long axeNbr      = ErrorAxis();
    long errNbr      = ErrorNo();
}
```

Example Documentation

```

long errInfoNbr = ErrorInfo();
long eposErr, sdoAbortCode;
AxisControl (AXALL, OFF);
switch(errNbr)
{
    case F_AMP:      if( axeNbr==C_AXIS1)
                    {
                        eposErr = SdoRead(C_DRIVE_BUSID1,EPOS4_ERROR_CODE,0x00);
                        printf("Error Axis:  %d, Epos4 Error 0x%lX: ", axeNbr , eposErr);
                        sdkEpos4_PrintErrorDescription(eposErr);
                        print();
                        AmpErrorClear(axeNbr); // Clear error on EPOS4
                    }
                    else
                    {
                        print("ErrorNo:  ",errNbr," info:  ",errInfoNbr, " AxisNo:  ", axeNbr);
                    }
                    break;
    case F_CANIO:    print("ErrorNo:  ",errNbr," info:  ",errInfoNbr);
                    sdoAbortCode = SYS_PROCESS(SYS_CANOM_SDOABORT);
                    printf("SDO Abort Code 0x%lX: ", sdoAbortCode );
                    sdkErrorPrint_SdoErrorDescription(sdoAbortCode);
                    print();
                    print("Check Can baudrate & Can bus id");
                    break;
    default:         print("ErrorNo:  ",errNbr," info:  ",errInfoNbr, " AxisNo:  ", axeNbr);
}
ErrorClear();
print(""); print(" There is no error handlig → Exit()");
Exit(0);
}

```

6.5 CAN_1Ax_EPOS4-Test_csv.mc

```

#include "..\..\..\SDK\SDK_ApossC.mc"
// Parameters for the SDK function
#define C_AXIS1      0          // Axis module number
#define C_DRIVE_BUSID1 1        // The CAN drive busId
#define C_PDO_NUMBER 1         // Used PDO number
#define C_AXISPOLARITY 0        // Definition of the polarity 0: Normal, 1: Inverse
// Encoder settings & axis user units (MACS)
#define C_AXIS_ENCRESC 4*1600    // Resolution of the encoder for position feed
    back in increments (quadcounts)
#define C_AXIS_POSENCREV 1        // Number of revolutions of the motor
#define C_AXIS_POSENQC C_AXIS_ENCRESC // Number of quadcounts in POSENCREV
    revolutions
#define C_AXIS_POSFACT_Z 1        // Number of revolutions of the input shaft
#define C_AXIS_POSFACT_N 1        // Number of revolutions of the output shaft
    in POSFACT_Z revolutions of the input shaft
#define C_AXIS_FEEDREV 1          // Number of revolutions of the gear box
    output shaft
#define C_AXIS_FEEDDIST C_AXIS_ENCRESC // Distance travelled (in user units) in
    FEEDREV revolutions of the gear box output shaft [mm]
// Axis Movement Parameter
#define C_AXIS_MAX_RPM 2000        // Maximum velocity in RPM
#define C_AXIS_VELRES 100          // Velocity resolution, Scaling used for the
    velocity and acceleration/deceleration commands, default
#define C_AXIS_RAMPTYPE RAMPTYPE_JERKLIMITED // Defines the ramptype
#define C_AXIS_RAMPMIN 800         // Maximum acceleration
#define C_AXIS_JERKMIN 1000        // Minimum time (ms) required before reaching the
    maximum acceleration
#define C_AXIS_TRACKERR 0          // There is also a following error on EPOS4, could
    be set to zero on the MACS
// Axis MACS control loop settings
// MACS position control is not used but still active. FeedForward of FFVEL set the velocity of the EPOS4
#define C_AXIS_KPROP 0
#define C_AXIS_KINT 0
#define C_AXIS_KDER 0

```

```

#define C_AXIS_KILIM            0
#define C_AXIS_KILIMTIME       0
#define C_AXIS_BANDWIDTH       1000
#define C_AXIS_FFVEL           1000
#define C_AXIS_KFFAC           0
#define C_AXIS_KFFDEC          0
long main(void) {
    long i, homingState=0, retval;
    print("-----");
    print(" Test application CANopen Master with 1 EPOS4 drive");
    print("-----");
    ErrorClear();
    AmpErrorClear(C_AXIS1);
    InterruptSetup(ERROR, ErrorHandler);
    //-----
    // Application Setup
    //-----
    if (GLB_PARAM(CANBAUD) != 88)
    {
        print("Set new Baudrate and save global parameters");
        // Set Baudrate of CAN 1 & CAN 2 to 1MBaud
        GLB_PARAM(CANBAUD)=88;
        CanOpenRestart();
        Save(GLBPARS);
    }
    // Cycle time for sending SYNC telegrams on the CAN bus - must be set before calling the
    // sdkEpos4_SetupCanSdoParam() function.
    GLB_PARAM(CANSYNCTIMER)= 1;
    // set all slaves to PREOPERATIONAL by sending an NMT.
    SYS_PROCESS(SYS_CANOM_MASTERSTATE) = 0;
    // initialising maxon drives
    sdkEpos4_SetupCanSdoParam(C_DRIVE_BUSID1, C_PDO_NUMBER, C_AXISPOLARITY, EPOS4_OP_CSV);
    // setup CANopen bus module for csp mode
    sdkEpos4_SetupCanBusModule(C_AXIS1, C_DRIVE_BUSID1, C_PDO_NUMBER, EPOS4_OP_CSV);
    // setup virtual amplifier for csp mode
    sdkEpos4_SetupCanVirtAmp(C_AXIS1, C_AXIS_MAX_RPM, EPOS4_OP_CSV);
    // setup irtual counter for csp mode
    sdkEpos4_SetupCanVirtCntin(C_AXIS1, EPOS4_OP_CSV);
    // start all slaves commanding them into OPERATIONAL.
    SYS_PROCESS(SYS_CANOM_MASTERSTATE) = 1;
    // Movement parameters for the axis
    sdkSetupAxisMovementParam( C_AXIS1,
                               C_AXIS_VELRES,
                               C_AXIS_MAX_RPM,
                               C_AXIS_RAMPTYPE,
                               C_AXIS_RAMPMIN,
                               C_AXIS_JERKMIN,
                               C_AXIS_TRACKERR
                               );
    // Definition of the user units
    sdkSetupAxisUserUnits( C_AXIS1,
                           C_AXIS_POSENCREV,
                           C_AXIS_POSENCQC,
                           C_AXIS_POSFACT_Z,
                           C_AXIS_POSFACT_N,
                           C_AXIS_FEEDREV,
                           C_AXIS_FEEDDIST
                           );
    // Position control setup
    sdkSetupPositionPIDControlExt( C_AXIS1,
                                   C_AXIS_KPROP,
                                   C_AXIS_KINT,
                                   C_AXIS_KDER,
                                   C_AXIS_KILIM,
                                   C_AXIS_KILIMTIME,
                                   C_AXIS_BANDWIDTH,
                                   C_AXIS_FFVEL,
                                   C_AXIS_KFFAC,
                                   C_AXIS_KFFDEC
                                   );
}

```

Example Documentation

```
//-----
// End of Application Setup
//-----
// Homing setup
print("\nEPOS4 Homing:");
SdoWrite( C_DRIVE_BUSID1, EPOS4_HOMING_METHOD,          0,  EPOS4_HOMING_CURRENT_N_SPEED); //
    0x6098 Set homing method to "-4" : Homing Method -4 (Current Threshold Negative Speed)."
SdoWrite( C_DRIVE_BUSID1, EPOS4_HOMING_SPEEDS,          1,  20); //
    Homing Speed / Speed for switch speed [velocity units]
SdoWrite( C_DRIVE_BUSID1, EPOS4_HOMING_SPEEDS,          2,  20); //
    Homing Speed / Speed for zero search [velocity units]
SdoWrite( C_DRIVE_BUSID1, EPOS4_HOMING_ACCELERATION,     0,  20); //
    Homing acceleration [acceleration units]
SdoWrite( C_DRIVE_BUSID1, EPOS4_HOME_OFFSET_MOVE_DISTANCE, 0,  6400); //
    Home offset move distance [position units]
SdoWrite( C_DRIVE_BUSID1, EPOS4_HOME_POSITION,          0,  0); //
    Home position [position units]
SdoWrite( C_DRIVE_BUSID1, EPOS4_CURRENT_THRESHOLD_FOR_HOMING_MODE, 0,  1500); //
    Current threshold for homing mode [mA]
// Homing statemachine
retval=0;
while(! retval)
{
    retval = sdkEpos4_AxisHomingStart(C_AXIS1, C_DRIVE_BUSID1, EPOS4_OP_CSV, homingState);
    // Homing error - Exit programm
    if(retval==-1)
    {
        print("Exit programm");
        Exit(0);
    }
}
print("");
print("-----");
print("          Movement in CSV Mode          ");
print("----- \n");
Cvel(C_AXIS1, 50);
Acc(C_AXIS1, 30);
Dec(C_AXIS1, 30);
AxisControl(C_AXIS1, ON);
for(i=5;i>=0;i--)
{
    print("Start, move with constant vel → positive");
    Cvel(C_AXIS1, 50);
    AxisCvelStart(C_AXIS1);
    Delay(5000);
    print("Start, move with constant vel → negative");
    Cvel(C_AXIS1, -50);
    Delay(5000);
    print(i, " repetitions to go \n");
}
AxisControl(C_AXIS1, OFF);
print("Program done, Axis OFF ");
return(0);
}

void ErrorHandler(void)
{
    long axeNbr      = ErrorAxis();
    long errNbr      = ErrorNo();
    long errInfoNbr  = ErrorInfo();
long eposErr, sdoAbortCode;
AxisControl(AXALL,OFF);
switch(errNbr)
{
    case F_AMP:      if( axeNbr==C_AXIS1)
                    {
                        eposErr = SdoRead(C_DRIVE_BUSID1,EPOS4_ERROR_CODE,0x00);
                        printf("Error Axis:  %d, Epos4 Error 0x%lX: ", axeNbr , eposErr);
                        sdkEpos4_PrintErrorDescription(eposErr);
                        print();
                        AmpErrorClear(axeNbr); // Clear error on EPOS4
                    }

```

```

    }
    else
    {
        print("ErrorNo: ",errNbr," info: ",errInfoNbr, " AxisNo: ", axeNbr);
    }
    break;
case F_CANIO:
    print("ErrorNo: ",errNbr," info: ",errInfoNbr);
    sdoAbortCode = SYS_PROCESS(SYS_CANOM_SDOABORT);
    printf("SDO Abort Code 0x%X: ", sdoAbortCode );
    sdkErrorPrint_SdoErrorDescription(sdoAbortCode);
    print();
    print("Check Can baudrate & Can bus id");
    break;
default:
    print("ErrorNo: ",errNbr," info: ",errInfoNbr, " AxisNo: ", axeNbr);
}
ErrorClear();
print(""); print(" There is no error handlig → Exit()");
Exit(0);
}

```

6.6 CAN_2Ax_EPOS4-Test_csp.mc

```

#include "..\..\..\SDK\SDK_ApossC.mc"
// Parameters for the SDK function
#define C_AXIS1          0          // Axis module number
#define C_DRIVE_BUSID1  1          // The CAN drive busId
#define C_AXIS2          1          // Axis module number
#define C_DRIVE_BUSID2  2          // The CAN drive busId
#define C_PDO_NUMBER     1          // Used PDO number
#define C_AXISPOLARITY   0          // Definition of the polarity 0: Normal, 1: Inverse
// Encoder settings & axis user units (MACS)
#define C_AXIS_ENCRES     4*1600      // Resolution of the encoder for position feed
    back in increments (quadcounts)
#define C_AXIS_POSENCREV  1          // Number of revolutions of the motor
#define C_AXIS_POSENCQC   C_AXIS_ENCRES // Number of quadcounts in POSENCREV
    revolutions
#define C_AXIS_POSFACT_Z  1          // Number of revolutions of the input shaft
#define C_AXIS_POSFACT_N  1          // Number of revolutions of the output shaft
    in POSFACT_Z revolutions of the input shaft
#define C_AXIS_FEEDREV    1          // Number of revolutions of the gear box
    output shaft
#define C_AXIS_FEEDDIST   C_AXIS_ENCRES // Distance travelled (in user units) in
    FEEDREV revolutions of the gear box output shaft [mm]
// Axis Movement Parameter
#define C_AXIS_MAX_RPM    2000      // Maximum velocity in RPM
#define C_AXIS_VELRES     100      // Velocity resolution, Scaling used for the
    velocity and acceleration/deceleration commands, default
#define C_AXIS_RAMPTYPE    RAMPTYPE_JERKLIMITED // Defines the ramptype
#define C_AXIS_RAMPMIN    800      // Maximum acceleration
#define C_AXIS_JERKMIN     1000     // Minimum time (ms) required before reaching the
    maximum acceleration
#define C_AXIS_TRACKERR    0          // There is also a following error on EPOS4, could
    be set to zero on the MACS
// Axis MACS control loop settings
// MACS position control is not used
#define C_AXIS_KPROP       0
#define C_AXIS_KINT        0
#define C_AXIS_KDER        0
#define C_AXIS_KILIM       0
#define C_AXIS_KILIMTIME   0
#define C_AXIS_BANDWIDTH   1000
#define C_AXIS_FFVEL       1000
#define C_AXIS_KFFAC       0
#define C_AXIS_KFFDEC      0
// Set the bus ids of the all axis in the network
long axis[] = {C_AXIS1, C_AXIS2};
long busIds[] = {C_DRIVE_BUSID1, C_DRIVE_BUSID2};
long numberOfAxis = arraylen(busIds);

```

Example Documentation

```

long main(void) {
    long i, homingState=0, retval, axisIndex;
    print("-----");
    print(" Test application CANopen Master with 1 EPOS4 drive");
    print("-----");
    ErrorClear();
    AmpErrorClear(C_AXIS1);
    InterruptSetup(ERROR, ErrorHandler);
    //-----
    // Application Setup
    //-----
    if (GLB_PARAM(CANBAUD) != 88)
    {
        print("Set new Baudrate and save global parameters");
        // Set Baudrate of CAN 1 & CAN 2 to 1MBaud
        GLB_PARAM(CANBAUD)=88;
        CanOpenRestart();
        Save(GLBPARS);
    }
    // set all slaves to PREOPERATIONAL by sending an NMT.
    SYS_PROCESS(SYS_CANOM_MASTERSTATE) = 0;
    for(axisIndex = 0; axisIndex < numberOfAxis; axisIndex++ )
    {
        // initialising maxon drives
        sdkEpos4_SetupCanSdoParam(busIds[axisIndex], C_PDO_NUMBER, C_AXISPOLARITY, EPOS4_OP_CSP);
        // setup CANopen bus module for csp mode
        sdkEpos4_SetupCanBusModule(axis[axisIndex], busIds[axisIndex], C_PDO_NUMBER, EPOS4_OP_CSP);
        // setup virtual amplifier for csp mode
        sdkEpos4_SetupCanVirtAmp(axis[axisIndex], C_AXIS_MAX_RPM, EPOS4_OP_CSP);
        // setup irtual counter for csp mode
        sdkEpos4_SetupCanVirtCntin(axis[axisIndex], EPOS4_OP_CSP);
        // Movement parameters for the axis
        sdkSetupAxisMovementParam( axis[axisIndex],
                                   C_AXIS_VELRES,
                                   C_AXIS_MAX_RPM,
                                   C_AXIS_RAMPTYPE,
                                   C_AXIS_RAMPMIN,
                                   C_AXIS_JERKMIN,
                                   C_AXIS_TRACKERR
                                   );
        // Definition of the user units
        sdkSetupAxisUserUnits(      axis[axisIndex],
                                   C_AXIS_POSENCREV,
                                   C_AXIS_POSENCQC,
                                   C_AXIS_POSFACT_Z,
                                   C_AXIS_POSFACT_N,
                                   C_AXIS_FEEDREV,
                                   C_AXIS_FEEDDIST
                                   );
        // Position control setup
        sdkSetupPositionPIDControlExt(      axis[axisIndex],
                                             C_AXIS_KPROP,
                                             C_AXIS_KINT,
                                             C_AXIS_KDER,
                                             C_AXIS_KILIM,
                                             C_AXIS_KILIMTIME,
                                             C_AXIS_BANDWIDTH,
                                             C_AXIS_FFVEL,
                                             C_AXIS_KFFAC,
                                             C_AXIS_KFFDEC
                                             );
    }
    // start all slaves commanding them into OPERATIONAL.
    SYS_PROCESS(SYS_CANOM_MASTERSTATE) = 1;
    //-----
    // End of Application Setup
    //-----
    for(axisIndex = 0; axisIndex < numberOfAxis; axisIndex++ )
    {
        // Homing setup
    }
}

```

```

    print("\nEPOS4 Homing:");
    SdoWrite( busIds[axisIndex], EPOS4_HOMING_METHOD, 0,
EPOS4_HOMING_CURRENT_N_SPEED); // 0x6098 Set homing method to "-4" : Homing Method -4 (Current
Threshold Negative Speed).
    SdoWrite( busIds[axisIndex], EPOS4_HOMING_SPEEDS, 1, 20);
// Homing Speed / Speed for switch speed [velocity units]
    SdoWrite( busIds[axisIndex], EPOS4_HOMING_SPEEDS, 2, 20);
// Homing Speed / Speed for zero search [velocity units]
    SdoWrite( busIds[axisIndex], EPOS4_HOMING_ACCELERATION, 0, 20);
// Homing acceleration [acceleration units]
    SdoWrite( busIds[axisIndex], EPOS4_HOME_OFFSET_MOVE_DISTANCE, 0, 6400);
// Home offset move distance [position units]
    SdoWrite( busIds[axisIndex], EPOS4_HOME_POSITION, 0, 0);
// Home position [position units]
    SdoWrite( busIds[axisIndex], EPOS4_CURRENT_THRESHOLD_FOR_HOMING_MODE, 0, 1500); //
Current threshold for homing mode [mA]
// Disable MACS trackerror for homing
    AXE_PARAM(C_AXIS2, POSERR)=2000000000;
}
// Homing statemachine
retval=0;
while(! retval)
{
    retval = 1;
    for(axisIndex = 0; axisIndex < numberOfAxis; axisIndex++ )
    {
        retval = retval && sdkEpos4_AxisHomingStart(axis[axisIndex], busIds[axisIndex], EPOS4_OP_CSP,
homingState);
        // Homing error - Exit programm
        if(retval==1)
        {
            print("Exit programm");
            Exit(0);
        }
    }
}
// Enable MACS trackerror for homing
AXE_PARAM(C_AXIS1, POSERR)=C_AXIS_TRACKERR;
AXE_PARAM(C_AXIS2, POSERR)=C_AXIS_TRACKERR;
print("");
print("-----");
print("                Movement in CSP Mode                ");
print("----- \n");
Vel(C_AXIS1, 50,C_AXIS2, 50);
Acc(C_AXIS1, 30,C_AXIS2, 50);
Dec(C_AXIS1, 30,C_AXIS2, 50);
AxisControl(C_AXIS1, ON, C_AXIS2, ON);
for(i=10;i>=0;i--)
{
    print("Start, move to target position");
    AxisPosAbsStart(C_AXIS1, 20000, C_AXIS2, 20000);
    AxisWaitReached(C_AXIS1,C_AXIS2);
    print("Target position is reached \n");
    print("Start, back to start position");
    AxisPosAbsStart(C_AXIS1, 0,C_AXIS2, 0);
    AxisWaitReached(C_AXIS1,C_AXIS2);
    print("Start position is reached");
    print(i, " repetitions to go \n");
}
AxisControl(C_AXIS1, OFF,C_AXIS2, OFF);
print("Program done, Axis OFF ");
return(0);
}
void ErrorHandler(void)
{
    long axeNbr      = ErrorAxis();
    long errNbr      = ErrorNo();
    long errInfoNbr  = ErrorInfo();
    long eposErr, sdoAbortCode;
    AxisControl(AXALL, OFF);

```

Example Documentation

```
switch(errNbr)
{
    case F_AMP:    if( axeNbr==C_AXIS1 || axeNbr==C_AXIS2 )
                    {
                        eposErr = SdoRead(C_DRIVE_BUSID1,EPOS4_ERROR_CODE,0x00);
                        printf("Error Axis: %d, Epos4 Error 0x%lX: ", axeNbr , eposErr);
                        sdkEpos4_PrintErrorDescription(eposErr);
                        print();
                        AmpErrorClear(axeNbr); // Clear error on EPOS4
                    }
                    else
                    {
                        print("ErrorNo: ",errNbr," info: ",errInfoNbr, " AxisNo: ", axeNbr);
                    }
                    break;
    case F_CANIO:  print("ErrorNo: ",errNbr," info: ",errInfoNbr);
                    sdoAbortCode = SYS_PROCESS(SYS_CANOM_SDOABORT);
                    printf("SDO Abort Code 0x%lX: ", sdoAbortCode );
                    sdkErrorPrint_SdoErrorDescription(sdoAbortCode);
                    print();
                    print("Check Can baudrate & Can bus id");
                    break;
    default:       print("ErrorNo: ",errNbr," info: ",errInfoNbr, " AxisNo: ", axeNbr);
}
ErrorClear();
print(""); print(" There is no error handlig → Exit()");
Exit(0);
}
```

6.7 CAN_4Ax_MiniMACS6_csp.mc

```
#include "..\..\..\SDK\SDK_ApossC.mc"
// Parameters for the SDK function
#define M_AX_0      0           // Axis module number - DS402 Master
#define M_AX_1      1           // Axis module number - DS402 Master
#define M_AX_2      2           // Axis module number - DS402 Master
#define M_AX_3      3           // Axis module number - DS402 Master
#define S_AX_0      0           // Axis module number - DS402 Slave
#define S_AX_1      1           // Axis module number - DS402 Slave
#define S_AX_2      2           // Axis module number - DS402 Slave
#define S_AX_3      3           // Axis module number - DS402 Slave
#define DRIVE_BUSID1 3          // The CAN drive busId of the MiniMACS6
#define PDO_S_AX_0   1          // Used PDO number
#define PDO_S_AX_1   2          // Used PDO number
#define PDO_S_AX_2   3          // Used PDO number
#define PDO_S_AX_3   4          // Used PDO number
// Encoder settings & axis user units (Master)
#define AXIS_ENCRES   4*256      // Resolution of the encoder for position feed back in
                                increments (quadcounts)
#define AXIS_POSENCREV 1         // Number of revolutions of the motor
#define AXIS_POSENCQC  AXIS_ENCRES // Number of quadcounts in POSENCREV revolutions
#define AXIS_POSFACT_Z 1         // Number of revolutions of the input shaft
#define AXIS_POSFACT_N 1         // Number of revolutions of the output shaft in
                                POSFACT_Z revolutions of the input shaft
#define AXIS_FEEDREV   1         // Number of revolutions of the gear box output shaft
#define AXIS_FEEDDIST  360       // Distance travelled (in user units) in FEEDREV
                                revolutions of the gear box output shaft [°]
// Axis Movement Parameter (Master)
#define AXIS_MAX_RPM   5500      // Maximum velocity in RPM
#define AXIS_VELRES    100       // Velocity resolution, Scaling used for the velocity
                                and acceleration/deceleration commands, default
#define AXIS_RAMPTYPE  RAMPTYPE_JERKLIMITED // Defines the ramptype
#define AXIS_RAMPMIN   20000     // Maximum acceleration
#define AXIS_JERMIN     1000     // Minimum time (ms) required before reaching the
                                maximum acceleration
#define AXIS_TRACKERR   200000   // There is also a following error on DS402 Slave,
                                could be very high ond the MACS
// Motor settings for actual torque feedback → VIRTAMP_PROCESS(0,PO_VIRTAMP_CURRENT)
```

```

#define SLAVE_TORQUE_CONST      15700           // Represents the motor's torque constant uNm/A
#define SLAVE_MOT_RATED_TORQUE 135             // Motor rated torque mNm
// Axis MACS control loop settings, Master position control is not used
#define AXIS_KPROP              0
#define AXIS_KINT                0
#define AXIS_KDER                0
#define AXIS_KILIM              0
#define AXIS_KILIMTIME          0
#define AXIS_BANDWIDTH          1000
#define AXIS_FFVEL              1000
#define AXIS_KFFAC              0
#define AXIS_KFFDEC             0
// Define Cia402 Objects
#define DS402_AXESOFFSET        0x800
long main(void) {
    // local variables
    long i, retval;
    long homingStateAx_0=0,homingStateAx_1=0,homingStateAx_2=0,homingStateAx_3=0;
    print("-----");
    print(" Test application CANopen Master - MiniMACS6 DS402 Slave");
    print("-----");
    ErrorClear();
    AmpErrorClear(AXALL);
    InterruptSetup(ERROR, ErrorHandler);
    //-----
    // Device Setup
    //
    // For the operation of 4 axes via Can the baud rate must be set
    // to 1 Mbaud. In addition, the CANSYNCTIME must be set to 2 ms.
    // As a rule of thumb 3 axis synchronization via CAN must happen
    // with 1Mbaud and synctime 1 ms.
    //-----
    if (GLB_PARAM(CANBAUD) != 88 || GLB_PARAM(CANSYNCTIMER) != 2)
    {
        print("New Baudrate: 1Mbaud");
        GLB_PARAM(CANBAUD)=88;
        GLB_PARAM(CANSYNCTIMER)=2;
        CanOpenRestart();
        Delay(500);
    }
    //-----
    // Application Setup
    //-----
    // set all slaves to PREOPERATIONAL by sending an NMT.
    SYS_PROCESS(SYS_CANOM_MASTERSTATE) = 0;
    // initialising maxon drives
    sdkMiniMACS6_SetupCanSdoParam(DRIVE_BUSID1, PDO_S_AX_0, S_AX_0, MINIMACS6_OP_CSP);
    sdkMiniMACS6_SetupCanSdoParam(DRIVE_BUSID1, PDO_S_AX_1, S_AX_1, MINIMACS6_OP_CSP);
    sdkMiniMACS6_SetupCanSdoParam(DRIVE_BUSID1, PDO_S_AX_2, S_AX_2, MINIMACS6_OP_CSP);
    sdkMiniMACS6_SetupCanSdoParam(DRIVE_BUSID1, PDO_S_AX_3, S_AX_3, MINIMACS6_OP_CSP);
    // setup CANopen bus module for csp mode
    sdkMiniMACS6_SetupCanBusModule(M_AX_0, DRIVE_BUSID1, PDO_S_AX_0, MINIMACS6_OP_CSP);
    sdkMiniMACS6_SetupCanBusModule(M_AX_1, DRIVE_BUSID1, PDO_S_AX_1, MINIMACS6_OP_CSP);
    sdkMiniMACS6_SetupCanBusModule(M_AX_2, DRIVE_BUSID1, PDO_S_AX_2, MINIMACS6_OP_CSP);
    sdkMiniMACS6_SetupCanBusModule(M_AX_3, DRIVE_BUSID1, PDO_S_AX_3, MINIMACS6_OP_CSP);
    // setup virtual amplifier for csp mode
    sdkMiniMACS6_SetupCanVirtAmp(M_AX_0, AXIS_MAX_RPM, MINIMACS6_OP_CSP);
    sdkMiniMACS6_SetupCanVirtAmp(M_AX_1, AXIS_MAX_RPM, MINIMACS6_OP_CSP);
    sdkMiniMACS6_SetupCanVirtAmp(M_AX_2, AXIS_MAX_RPM, MINIMACS6_OP_CSP);
    sdkMiniMACS6_SetupCanVirtAmp(M_AX_3, AXIS_MAX_RPM, MINIMACS6_OP_CSP);
    // setup irtual counter for csp mode
    sdkMiniMACS6_SetupCanVirtCntin(M_AX_0, MINIMACS6_OP_CSP);
    sdkMiniMACS6_SetupCanVirtCntin(M_AX_1, MINIMACS6_OP_CSP);
    sdkMiniMACS6_SetupCanVirtCntin(M_AX_2, MINIMACS6_OP_CSP);
    sdkMiniMACS6_SetupCanVirtCntin(M_AX_3, MINIMACS6_OP_CSP);
    // start all slaves commanding them into OPERATIONAL.
    SYS_PROCESS(SYS_CANOM_MASTERSTATE) = 1;
    // Setup axsi parameter (Master)
    for (i = M_AX_0; i <= M_AX_3; i++)
    {

```

Example Documentation

```
// Movement parameters for the axis
sdkSetupAxisMovementParam( i,
    AXIS_VELRES,
    AXIS_MAX_RPM,
    AXIS_RAMPTYPE,
    AXIS_RAMPMIN,
    AXIS_JERKMIN,
    AXIS_TRACKERR
);

// Definition of the user units
sdkSetupAxisUserUnits( i,
    AXIS_POSENCREV,
    AXIS_POSENCQC,
    AXIS_POSFACT_Z,
    AXIS_POSFACT_N,
    AXIS_FEEDREV,
    AXIS_FEEDDIST
);

// Position control setup
sdkSetupPositionPIDControlExt( i,
    AXIS_KPROP,
    AXIS_KINT,
    AXIS_KDER,
    AXIS_KILIM,
    AXIS_KILIMTIME,
    AXIS_BANDWIDTH,
    AXIS_FFVEL,
    AXIS_KFFAC,
    AXIS_KFFDEC
);

// Slave information for correct display of current torque
SdoWrite( DRIVE_BUSID1, 0x6076 + DS402_AXESOFFSET*i, 00, SLAVE_MOT_RATED_TORQUE); // Set
Motor rated torque
SdoWrite( DRIVE_BUSID1, 0x28C0 + i, VIRTAMP_TORQUE_CONST, SLAVE_TORQUE_CONST); // Set
the motor's torque constant
print("Setup axis parameter: ", i);
}
//-----
// End of Application Setup
//-----
// Homing setup
print("\nDS402 MiniMACS6 Homing:");
// The homing setup can also be configured on the DS402 slave.
for (i = M_AX_0; i <= M_AX_3; i++)
{
    SdoWrite( DRIVE_BUSID1, 0x6098 + DS402_AXESOFFSET*i, 0, 37); // Select "Home at actual
    position"
    SdoWrite( DRIVE_BUSID1, 0x607C + DS402_AXESOFFSET*i, 0, 0); // Set homing offset to [uu]
}
// Homing statemachine
retval=0;
while(retval!=0x0F)
{
    retval.i[0] = sdkMiniMACS6_AxisHomingStart(M_AX_0, DRIVE_BUSID1, MINIMACS6_OP_CSP,
    homingStateAx_0);
    retval.i[1] = sdkMiniMACS6_AxisHomingStart(M_AX_1, DRIVE_BUSID1, MINIMACS6_OP_CSP,
    homingStateAx_1);
    retval.i[2] = sdkMiniMACS6_AxisHomingStart(M_AX_2, DRIVE_BUSID1, MINIMACS6_OP_CSP,
    homingStateAx_2);
    retval.i[3] = sdkMiniMACS6_AxisHomingStart(M_AX_3, DRIVE_BUSID1, MINIMACS6_OP_CSP,
    homingStateAx_3);
}
print("");
print("-----");
print(" Movement in CSP Mode ");
print("----- \n");
// Movement settings
Vel(AXALL, 100);
Acc(AXALL, 100);
Dec(AXALL, 100);
```

```

// Enable the axes
AxisControl(M_AX_0, ON,M_AX_1, ON,M_AX_2, ON,M_AX_3, ON);
while(1)
{
    print("Start, move to target position - 10 revolutions");
    AxisPosAbsStart(    M_AX_0, 10*AXIS_FEEDDIST,
                       M_AX_1, 10*AXIS_FEEDDIST,
                       M_AX_2, 10*AXIS_FEEDDIST,
                       M_AX_3, 10*AXIS_FEEDDIST);
    AxisWaitReached(M_AX_0,M_AX_1,M_AX_2,M_AX_3);
    print("Target position is reached \n");
    print("Start, back to start position");
    AxisPosAbsStart(    M_AX_0, 0,
                       M_AX_1, 0,
                       M_AX_2, 0,
                       M_AX_3, 0);
    AxisWaitReached(M_AX_0,M_AX_1,M_AX_2,M_AX_3);
    print("Start position is reached");
    print(i-1, " repetitions to go \n");
}
print("Program done, Axis OFF ");
AxisControl(AXALL, OFF);
return(0);
}

void ErrorHandler(void)
{
    long axeNbr      = ErrorAxis();
    long errNbr      = ErrorNo();
    long errInfoNbr = ErrorInfo();
    AxisControl(AXALL,OFF);
    switch(errNbr)
    {
        case F_AMP:      if( axeNbr==M_AX_0 ||axeNbr==M_AX_1 || axeNbr==M_AX_2 || axeNbr==M_AX_3 )
                        {
                            print("DS402 Slave Error");
                            print("Error Axis: ", axeNbr ," ErrorCode: 0x",
                                radixstr(SdoRead(DRIVE_BUSID1,0x603F+DS402_AXESOFFSET*axeNbr,0x00),16));
                            // Clear error on Slave Device
                            Delay(5000);
                            AmpErrorClear(axeNbr);
                        }
                        else
                        {
                            print("ErrorNo: ",errNbr," info: ",errInfoNbr, " AxisNo: ", axeNbr);
                        }
                        break;
        case F_CANIO:    print("ErrorNo: ",errNbr," info: ",errInfoNbr);
                        printf("SDO Abort Code %lX\n", SYS_PROCESS(SYS_CANOM_SDOABORT) );
                        print("Check Can baudrate & Can bus id");
                        break;
        default:         print("ErrorNo: ",errNbr," info: ",errInfoNbr, " AxisNo: ", axeNbr);
    }
    ErrorClear();
    print(""); print(" There is no error handlig → Exit()");
    Exit(0);
}

```

6.8 CAN_4Ax_MiniMACS6_cst.mc

```

#include "..\..\..\SDK\SDK_ApossC.mc"
// Parameters for the SDK function
#define M_AX_0      0           // Axis module number - DS402 Master
#define M_AX_1      1           // Axis module number - DS402 Master
#define M_AX_2      2           // Axis module number - DS402 Master
#define M_AX_3      3           // Axis module number - DS402 Master
#define S_AX_0      0           // Axis module number - DS402 Slave
#define S_AX_1      1           // Axis module number - DS402 Slave
#define S_AX_2      2           // Axis module number - DS402 Slave

```

Example Documentation

```
#define S_AX_3      3           // Axis module number - DS402 Slave
#define DRIVE_BUSID1  3           // The CAN drive busId of the MiniMACS6
#define PDO_S_AX_0    1           // Used PDO number
#define PDO_S_AX_1    2           // Used PDO number
#define PDO_S_AX_2    3           // Used PDO number
#define PDO_S_AX_3    4           // Used PDO number
// Encoder settings & axis user units (Master)
#define AXIS_ENCRES    4*256       // Resolution of the encoder for position feed back in
    increments (quadcounts)
#define AXIS_POSENCREV 1           // Number of revolutions of the motor
#define AXIS_POSENCQC  AXIS_ENCRES // Number of quadcounts in POSENCREV revolutions
#define AXIS_POSFACT_Z 1           // Number of revolutions of the input shaft
#define AXIS_POSFACT_N 1           // Number of revolutions of the output shaft in
    POSFACT_Z revolutions of the input shaft
#define AXIS_FEEDREV    1           // Number of revolutions of the gear box output shaft
#define AXIS_FEEDDIST   360         // Distance travelled (in user units) in FEEDREV
    revolutions of the gear box output shaft [mm]
// Axis Movement Parameter (Master)
#define AXIS_MAX_RPM    5500       // Maximum velocity in RPM
#define AXIS_VELRES     100        // Velocity resolution, Scaling used for the velocity
    and acceleration/deceleration commands, default
#define AXIS_RAMPTYPE   RAMPTYPE_JERKLIMITED // Defines the ramptype
#define AXIS_RAMPMIN    20000      // Maximum acceleration
#define AXIS_JERKMIN     1000       // Minimum time (ms) required before reaching the
    maximum acceleration
#define AXIS_TRACKERR    0          // There is also a following error on DS402 Slave,
    could be very high and the MACS
// Motor settings for actual torque feedback → VIRTAMP_PROCESS(0,PO_VIRTAMP_CURRENT)
#define SLAVE_TORQUE_CONST 15700    // Represents the motor's torque constant uNm/A
#define SLAVE_MOT_RATED_TORQUE 135 // Motor rated torque mNm
// Axis MACS control loop settings, Master position control is not used
#define AXIS_KPROP      0
#define AXIS_KINT       0
#define AXIS_KDER       0
#define AXIS_KILIM      0
#define AXIS_KILIMTIME  0
#define AXIS_BANDWIDTH  1000
#define AXIS_FFVEL      1000
#define AXIS_KFFAC      0
#define AXIS_KFFDEC     0
// Define Cia402 Objects
#define DS402_AXESOFFSET 0x800
long main(void) {
    // local variables
    long i, retval;
    long homingStateAx_0=0,homingStateAx_1=0,homingStateAx_2=0,homingStateAx_3=0;
    print("-----");
    print(" Test application CANopen Master - MiniMACS6 DS402 Slave");
    print("-----");
    ErrorClear();
    AmpErrorClear(AXALL);
    InterruptSetup(ERROR, ErrorHandler);
    //-----
    // Device Setup
    //
    // For the operation of 4 axes via Can the baud rate must be set
    // to 1 MBaud. In addition, the CANSYNCTIME must be set to 2 ms.
    // As a rule of thumb 3axis synchronization via CAN must happen
    // with 1MBaud and synctime 1 ms.
    //-----
    if(GLB_PARAM(CANBAUD) !=88 || GLB_PARAM(CANSYNCTIMER) !=2)
    {
        print("New Baudrate: 1MBaud");
        GLB_PARAM(CANBAUD)=88;
        GLB_PARAM(CANSYNCTIMER)=2;
        CanOpenRestart();
        Delay(500);
    }
    //-----
    // Application Setup
```

```

//-----
// set all slaves to PREOPERATIONAL by sending an NMT.
SYS_PROCESS(SYS_CANOM_MASTERSTATE) = 0;
// initialising maxon drives
sdkMiniMACS6_SetupCanSdoParam(DRIVE_BUSID1, PDO_S_AX_0, S_AX_0, MINIMACS6_OP_CST);
sdkMiniMACS6_SetupCanSdoParam(DRIVE_BUSID1, PDO_S_AX_1, S_AX_1, MINIMACS6_OP_CST);
sdkMiniMACS6_SetupCanSdoParam(DRIVE_BUSID1, PDO_S_AX_2, S_AX_2, MINIMACS6_OP_CST);
sdkMiniMACS6_SetupCanSdoParam(DRIVE_BUSID1, PDO_S_AX_3, S_AX_3, MINIMACS6_OP_CST);
// setup CANopen bus module for csp mode
sdkMiniMACS6_SetupCanBusModule(M_AX_0, DRIVE_BUSID1, PDO_S_AX_0, MINIMACS6_OP_CST);
sdkMiniMACS6_SetupCanBusModule(M_AX_1, DRIVE_BUSID1, PDO_S_AX_1, MINIMACS6_OP_CST);
sdkMiniMACS6_SetupCanBusModule(M_AX_2, DRIVE_BUSID1, PDO_S_AX_2, MINIMACS6_OP_CST);
sdkMiniMACS6_SetupCanBusModule(M_AX_3, DRIVE_BUSID1, PDO_S_AX_3, MINIMACS6_OP_CST);
// setup virtual amplifier for csp mode
sdkMiniMACS6_SetupCanVirtAmp(M_AX_0, AXIS_MAX_RPM, MINIMACS6_OP_CST);
sdkMiniMACS6_SetupCanVirtAmp(M_AX_1, AXIS_MAX_RPM, MINIMACS6_OP_CST);
sdkMiniMACS6_SetupCanVirtAmp(M_AX_2, AXIS_MAX_RPM, MINIMACS6_OP_CST);
sdkMiniMACS6_SetupCanVirtAmp(M_AX_3, AXIS_MAX_RPM, MINIMACS6_OP_CST);
// setup irtual counter for csp mode
sdkMiniMACS6_SetupCanVirtCntin(M_AX_0, MINIMACS6_OP_CST);
sdkMiniMACS6_SetupCanVirtCntin(M_AX_1, MINIMACS6_OP_CST);
sdkMiniMACS6_SetupCanVirtCntin(M_AX_2, MINIMACS6_OP_CST);
sdkMiniMACS6_SetupCanVirtCntin(M_AX_3, MINIMACS6_OP_CST);
// start all slaves commanding them into OPERATIONAL.
SYS_PROCESS(SYS_CANOM_MASTERSTATE) = 1;
// Setup axsi parameter (Master)
for (i = M_AX_0; i <= M_AX_3; i++)
{
    // Movement parameters for the axis
    sdkSetupAxisMovementParam( i,
                                AXIS_VELRES,
                                AXIS_MAX_RPM,
                                AXIS_RAMPTYPE,
                                AXIS_RAMPMIN,
                                AXIS_JERKMIN,
                                AXIS_TRACKERR
                                );

    // Definition of the user units
    sdkSetupAxisUserUnits(      i,
                                AXIS_POSENCREV,
                                AXIS_POSENCQC,
                                AXIS_POSFACT_Z,
                                AXIS_POSFACT_N,
                                AXIS_FEEDREV,
                                AXIS_FEEDDIST
                                );

    // Position control setup
    sdkSetupPositionPIDControlExt( i,
                                    AXIS_KPROP,
                                    AXIS_KINT,
                                    AXIS_KDER,
                                    AXIS_KILIM,
                                    AXIS_KILIMTIME,
                                    AXIS_BANDWIDTH,
                                    AXIS_FFVEL,
                                    AXIS_KFFAC,
                                    AXIS_KFFDEC
                                    );

    // Slave information for correct display of current torque
    SdoWrite( DRIVE_BUSID1, 0x6076 + DS402_AXESOFFSET*i, 00, SLAVE_MOT_RATED_TORQUE); // Set
    Motor rated torque
    SdoWrite( DRIVE_BUSID1, 0x28C0 + i, VIRTAMP_TORQUE_CONST, SLAVE_TORQUE_CONST); // Set
    the motor's torque constant
    print("Setup axis parameter: ", i);
}
//-----
// End of Application Setup
//-----
// Homing setup
print("\nDS402 MiniMACS6 Homing:");

```

Example Documentation

```
// The homing setup can also be configured on the DS402 slave.
for (i = M_AX_0; i <= M_AX_3; i++)
{
    SdoWrite( DRIVE_BUSID1, 0x6098 + DS402_AXESOFFSET*i, 0, 37); // Select "Home at actual
    position"
    SdoWrite( DRIVE_BUSID1, 0x607C + DS402_AXESOFFSET*i, 0, 0); // Set homing offset to [uu]
}
// Homing statemachine
retval=0;
while(retval!=0x0F)
{
    retval.i[0] = sdkMiniMACS6_AxisHomingStart(M_AX_0, DRIVE_BUSID1, MINIMACS6_OP_CST,
    homingStateAx_0);
    retval.i[1] = sdkMiniMACS6_AxisHomingStart(M_AX_1, DRIVE_BUSID1, MINIMACS6_OP_CST,
    homingStateAx_1);
    retval.i[2] = sdkMiniMACS6_AxisHomingStart(M_AX_2, DRIVE_BUSID1, MINIMACS6_OP_CST,
    homingStateAx_2);
    retval.i[3] = sdkMiniMACS6_AxisHomingStart(M_AX_3, DRIVE_BUSID1, MINIMACS6_OP_CST,
    homingStateAx_3);
}
// Disable MACS trackerror for cst mode
AXE_PARAM(M_AX_0, POSERR)=0;
print("");
print("-----");
print("                Set torque in CST Mode                ");
print("----- \n");
AxisControl(M_AX_0, ON,M_AX_1, ON,M_AX_2, ON,M_AX_3, ON);
for(i=10;i>=0;i--)
{
    // The value is given in per thousand of "Motor rated torque"
    print("Set target torque → positive");
    AXE_PROCESS(M_AX_0,REG_USERREFCUR)= 500; // 50.0 torque%
    AXE_PROCESS(M_AX_1,REG_USERREFCUR)= 500; // 50.0 torque%
    AXE_PROCESS(M_AX_2,REG_USERREFCUR)= 500; // 50.0 torque%
    AXE_PROCESS(M_AX_3,REG_USERREFCUR)= 500; // 50.0 torque%
    Delay(4000);
    // The value is given in per thousand of "Motor rated torque"
    print("Set target torque → negative");
    AXE_PROCESS(M_AX_0,REG_USERREFCUR)= -500; // -50.0 torque%
    AXE_PROCESS(M_AX_1,REG_USERREFCUR)= -500; // -50.0 torque%
    AXE_PROCESS(M_AX_2,REG_USERREFCUR)= -500; // -50.0 torque%
    AXE_PROCESS(M_AX_3,REG_USERREFCUR)= -500; // -50.0 torque%
    Delay(4000);
    // The value is given in per thousand of "Motor rated torque"
    print("Set target torque → zero");
    AXE_PROCESS(M_AX_0,REG_USERREFCUR)= 0; // 0 torque%
    AXE_PROCESS(M_AX_1,REG_USERREFCUR)= 0; // 0 torque%
    AXE_PROCESS(M_AX_2,REG_USERREFCUR)= 0; // 0 torque%
    AXE_PROCESS(M_AX_3,REG_USERREFCUR)= 0; // 0 torque%
    Delay(2000);
    print(i, " repetitions to go \n");
}
AxisControl(M_AX_0, OFF);
// Enable the axes
AxisControl(M_AX_0, ON,M_AX_1, ON,M_AX_2, ON,M_AX_3, ON);
if(VIRTAMP_PROCESS(M_AX_0,PO_VIRTAMP_STATUS).i[12]==1)
    print("Drive Ready");
while(1)
{
    print("Start, move to target position - 10 revolutions");
    AxisPosAbsStart(
        M_AX_0, 10*AXIS_FEEDDIST,
        M_AX_1, 10*AXIS_FEEDDIST,
        M_AX_2, 10*AXIS_FEEDDIST,
        M_AX_3, 10*AXIS_FEEDDIST);
    AxisWaitReached(M_AX_0,M_AX_1,M_AX_2,M_AX_3);
    print("Target position is reached \n");
    print("Start, back to start position");
    AxisPosAbsStart(
        M_AX_0, 0,
        M_AX_1, 0,
        M_AX_2, 0,
```

```

        M_AX_3, 0);
    AxisWaitReached(M_AX_0,M_AX_1,M_AX_2,M_AX_3);
    print("Start position is reached");
    print(i-1, " repetitions to go \n");
}
print("Program done, Axis OFF ");
AxisControl (AXALL, OFF);
return(0);
}
void ErrorHandler(void)
{
    long axeNbr      = ErrorAxis();
    long errNbr      = ErrorNo();
    long errInfoNbr = ErrorInfo();
    AxisControl (AXALL, OFF);
    switch(errNbr)
    {
        case F_AMP:      if( axeNbr==M_AX_0 ||axeNbr==M_AX_1 || axeNbr==M_AX_2 || axeNbr==M_AX_3 )
                        {
                            print("DS402 Slave Error");
                            print("Error Axis: ", axeNbr , " ErrorCode:  0x",
                                radixstr(SdoRead(DRIVE_BUSID1,0x603F,0x00),16));
                            // Clear error on Slave Device
                            Delay(5000);
                            AmpErrorClear(axeNbr);
                        }
                        else
                        {
                            print("ErrorNo:  ",errNbr," info:  ",errInfoNbr, " AxisNo:  ", axeNbr);
                        }
                        break;
        case F_CANIO:    print("ErrorNo:  ",errNbr," info:  ",errInfoNbr);
                        printf("SDO Abort Code %lx\n", SYS_PROCESS(SYS_CANOM_SDOABORT) );
                        print("Check Can baudrate & Can bus id");
                        break;
        default:         print("ErrorNo:  ",errNbr," info:  ",errInfoNbr, " AxisNo:  ", axeNbr);
    }
    ErrorClear();
    print(""); print(" There is no error handlig → Exit()");
    Exit(0);
}

```

6.9 CAN_4Ax_MiniMACS6_csv.mc

```

#include "..\..\..\SDK\SDK_ApossC.mc"
// Parameters for the SDK function
#define M_AX_0      0           // Axis module number - DS402 Master
#define M_AX_1      1           // Axis module number - DS402 Master
#define M_AX_2      2           // Axis module number - DS402 Master
#define M_AX_3      3           // Axis module number - DS402 Master
#define S_AX_0      0           // Axis module number - DS402 Slave
#define S_AX_1      1           // Axis module number - DS402 Slave
#define S_AX_2      2           // Axis module number - DS402 Slave
#define S_AX_3      3           // Axis module number - DS402 Slave
#define DRIVE_BUSID1 100003     // The CAN drive busId of the MiniMACS6
#define PDO_S_AX_0   1           // Used PDO number
#define PDO_S_AX_1   2           // Used PDO number
#define PDO_S_AX_2   3           // Used PDO number
#define PDO_S_AX_3   4           // Used PDO number
// Encoder settings & axis user units (Master)
#define AXIS_ENCRES   4*256      // Resolution of the encoder for position feed back in
                                increments (quadcounts)
#define AXIS_POSENCREV 1         // Number of revolutions of the motor
#define AXIS_POSENCQC  AXIS_ENCRES // Number of quadcounts in POSENCREV revolutions
#define AXIS_POSFACT_Z 1         // Number of revolutions of the input shaft
#define AXIS_POSFACT_N 1         // Number of revolutions of the output shaft in
                                POSFACT_Z revolutions of the input shaft
#define AXIS_FEEDREV   1         // Number of revolutions of the gear box output shaft

```

Example Documentation

```
#define AXIS_FEEDDIST      60                      // Distance travelled (in user units) in FEEDREV
               revolutions of the gear box output shaft [rps * 60]
// Axis Movement Parameter (Master)
#define AXIS_MAX_RPM      5000                    // Maximum velocity in RPM
#define AXIS_VELRES       AXIS_MAX_RPM           // Velocity resolution, Scaling used for the velocity
               and acceleration/deceleration commands, default
#define AXIS_RAMPTYPE     RAMPTYPE_JERKLIMITED    // Defines the ramptype
#define AXIS_RAMPMIN      2000                    // Maximum acceleration
#define AXIS_JERKMIN      1000                    // Minimum time (ms) required before reaching the
               maximum acceleration
#define AXIS_TRACKERR     0                      // There is also a following error on DS402 Slave,
               could be very high and the MACS
// Motor settings for actual torque feedback → VIRTAMP_PROCESS(0,PO_VIRTAMP_CURRENT)
#define SLAVE_TORQUE_CONST 15700                  // Represents the motor's torque constant uNm/A
#define SLAVE_MOT_RATED_TORQUE 135                // Motor rated torque mNm
// Axis MACS control loop settings, Master position control is not used
#define AXIS_KPROP        0
#define AXIS_KINT         0
#define AXIS_KDER         0
#define AXIS_KILIM        0
#define AXIS_KILIMTIME    0
#define AXIS_BANDWIDTH    1000
#define AXIS_FFVEL        1000
#define AXIS_KFFAC        0
#define AXIS_KFFDEC       0
// Define Cia402 Objects
#define DS402_AXESOFFSET  0x800
long main(void) {
    // local variables
    long i, retval;
    long homingStateAx_0=0,homingStateAx_1=0,homingStateAx_2=0,homingStateAx_3=0;
    print("-----");
    print(" Test application CANopen Master - MiniMACS6 DS402 Slave");
    print("-----");
    ErrorClear();
    AmpErrorClear(AXALL);
    InterruptSetup(ERROR, ErrorHandler);
    //-----
    // Device Setup
    //
    // For the operation of 4 axes via Can the baud rate must be set
    // to 1 Mbaud. In addition, the CANSYNCTIME must be set to 2 ms.
    // As a rule of thumb 3axis synchronization via CAN must happen
    // with 1Mbaud and synctime 1 ms.
    //-----
    if(GLB_PARAM(CANBAUD) != 88 || GLB_PARAM(CANSYNCTIMER) != 2)
    {
        print("New Baudrate: 1Mbaud");
        GLB_PARAM(CANBAUD)=88;
        GLB_PARAM(CANSYNCTIMER)=2;
        CanOpenRestart();
        Delay(500);
    }
    //-----
    // Application Setup
    //-----
    // set all slaves to PREOPERATIONAL by sending an NMT.
    SYS_PROCESS(SYS_CANOM_MASTERSTATE) = 0;
    // initialising maxon drives
    sdkMiniMACS6_SetupCanSdoParam(DRIVE_BUSID1, PDO_S_AX_0, S_AX_0, MINIMACS6_OP_CSV);
    sdkMiniMACS6_SetupCanSdoParam(DRIVE_BUSID1, PDO_S_AX_1, S_AX_1, MINIMACS6_OP_CSV);
    sdkMiniMACS6_SetupCanSdoParam(DRIVE_BUSID1, PDO_S_AX_2, S_AX_2, MINIMACS6_OP_CSV);
    sdkMiniMACS6_SetupCanSdoParam(DRIVE_BUSID1, PDO_S_AX_3, S_AX_3, MINIMACS6_OP_CSV);
    // setup CANopen bus module for csp mode
    sdkMiniMACS6_SetupCanBusModule(M_AX_0, DRIVE_BUSID1, PDO_S_AX_0, MINIMACS6_OP_CSV);
    sdkMiniMACS6_SetupCanBusModule(M_AX_1, DRIVE_BUSID1, PDO_S_AX_1, MINIMACS6_OP_CSV);
    sdkMiniMACS6_SetupCanBusModule(M_AX_2, DRIVE_BUSID1, PDO_S_AX_2, MINIMACS6_OP_CSV);
    sdkMiniMACS6_SetupCanBusModule(M_AX_3, DRIVE_BUSID1, PDO_S_AX_3, MINIMACS6_OP_CSV);
    // setup virtual amplifier for csp mode
    sdkMiniMACS6_SetupCanVirtAmp(M_AX_0, AXIS_MAX_RPM, MINIMACS6_OP_CSV);
```

```

sdkMiniMACS6_SetupCanVirtAmp(M_AX_1, AXIS_MAX_RPM, MINIMACS6_OP_CSV);
sdkMiniMACS6_SetupCanVirtAmp(M_AX_2, AXIS_MAX_RPM, MINIMACS6_OP_CSV);
sdkMiniMACS6_SetupCanVirtAmp(M_AX_3, AXIS_MAX_RPM, MINIMACS6_OP_CSV);
// setup irtual counter for csp mode
sdkMiniMACS6_SetupCanVirtCntin(M_AX_0, MINIMACS6_OP_CSV);
sdkMiniMACS6_SetupCanVirtCntin(M_AX_1, MINIMACS6_OP_CSV);
sdkMiniMACS6_SetupCanVirtCntin(M_AX_2, MINIMACS6_OP_CSV);
sdkMiniMACS6_SetupCanVirtCntin(M_AX_3, MINIMACS6_OP_CSV);
// start all slaves commanding them into OPERATIONAL.
SYS_PROCESS(SYS_CANOM_MASTERSTATE) = 1;
// Setup axsi parameter (Master)
for (i = M_AX_0; i <= M_AX_3; i++)
{
    // Movement parameters for the axis
    sdkSetupAxisMovementParam( i,
                               AXIS_VELRES,
                               AXIS_MAX_RPM,
                               AXIS_RAMPTYPE,
                               AXIS_RAMPMIN,
                               AXIS_JERKMIN,
                               AXIS_TRACKERR
                               );
    // Definition of the user units
    sdkSetupAxisUserUnits(      i,
                               AXIS_POSENCREV,
                               AXIS_POSENCQC,
                               AXIS_POSFACT_Z,
                               AXIS_POSFACT_N,
                               AXIS_FEEDREV,
                               AXIS_FEEDDIST
                               );
    // Position control setup
    sdkSetupPositionPIDControlExt( i,
                                   AXIS_KPROP,
                                   AXIS_KINT,
                                   AXIS_KDER,
                                   AXIS_KILIM,
                                   AXIS_KILIMTIME,
                                   AXIS_BANDWIDTH,
                                   AXIS_FFVEL,
                                   AXIS_KFFAC,
                                   AXIS_KFFDEC
                                   );
    // Slave information for correct display of current torque
    SdoWrite( DRIVE_BUSID1, 0x6076 + DS402_AXESOFFSET*i, 00, SLAVE_MOT_RATED_TORQUE); // Set
    Motor rated torque
    SdoWrite( DRIVE_BUSID1, 0x28C0 + i, VIRTAMP_TORQUE_CONST, SLAVE_TORQUE_CONST); // Set
    the motor's torque constant
    print("Setup axis parameter: ", i);
}
//-----
// End of Application Setup
//-----
// Homing setup
print("\nDS402 MiniMACS6 Homing:");
// The homing setup can also be configured on the DS402 slave.
for (i = M_AX_0; i <= M_AX_3; i++)
{
    SdoWrite( DRIVE_BUSID1, 0x6098 + DS402_AXESOFFSET*i, 0, 37); // Select "Home at actual
    position"
    SdoWrite( DRIVE_BUSID1, 0x607C + DS402_AXESOFFSET*i, 0, 0); // Set homing offset to [uu]
}
// Homing statemachine
retval=0;
while(retval!=0x0F)
{
    retval.i[0] = sdkMiniMACS6_AxisHomingStart(M_AX_0, DRIVE_BUSID1, MINIMACS6_OP_CSV,
    homingStateAx_0);
    retval.i[1] = sdkMiniMACS6_AxisHomingStart(M_AX_1, DRIVE_BUSID1, MINIMACS6_OP_CSV,
    homingStateAx_1);
}

```

Example Documentation

```

    retval.i[2] = sdkMiniMACS6_AxisHomingStart(M_AX_2, DRIVE_BUSID1, MINIMACS6_OP_CSV,
    homingStateAx_2);
    retval.i[3] = sdkMiniMACS6_AxisHomingStart(M_AX_3, DRIVE_BUSID1, MINIMACS6_OP_CSV,
    homingStateAx_3);
}
print("");
print("-----");
print("                Movement in CSV Mode                ");
print("----- \n");
// Enable the axes
AxisControl(M_AX_0, ON,M_AX_1, ON,M_AX_2, ON,M_AX_3, ON);
// Movement settings
Acc(AXALL, AXIS_MAX_RPM);
Dec(AXALL, AXIS_MAX_RPM);
for(i=5;i>=0;i--)
{
    print("Start, move with constant vel → positive");
    Cvel(AXALL, AXIS_MAX_RPM);
    AxisCvelStart( M_AX_0,
                   M_AX_1,
                   M_AX_2,
                   M_AX_3);

    Delay(10000);
    print("Start, move with constant vel → negative");
    Cvel(AXALL, -AXIS_MAX_RPM);
    AxisCvelStart( M_AX_0,
                   M_AX_1,
                   M_AX_2,
                   M_AX_3);

    Delay(10000);
    print(i, " repetitions to go \n");
}
print("Program done, Axis OFF ");
AxisControl(AXALL, OFF);
return(0);
}

void ErrorHandler(void)
{
    long axeNbr      = ErrorAxis();
    long errNbr      = ErrorNo();
    long errInfoNbr = ErrorInfo();
    AxisControl(AXALL,OFF);
    switch(errNbr)
    {
        case F_AMP:      if( axeNbr==M_AX_0 ||axeNbr==M_AX_1 || axeNbr==M_AX_2 || axeNbr==M_AX_3 )
                        {
                            print("DS402 Slave Error");
                            print("Error Axis: ", axeNbr ," ErrorCode:  0x",
                                radixstr(SdoRead(DRIVE_BUSID1,0x603F,0x00),16));
                            // Clear error on Slave Device
                            Delay(5000);
                            AmpErrorClear(axeNbr);
                        }
                        else
                        {
                            print("ErrorNo:  ",errNbr," info:  ",errInfoNbr, " AxisNo:  ", axeNbr);
                        }
                        break;
        case F_CANIO:    print("ErrorNo:  ",errNbr," info:  ",errInfoNbr);
                        printf("SDO Abort Code %lx\n", SYS_PROCESS(SYS_CANOM_SDOABORT) );
                        print("Check Can baudrate & Can bus id");
                        break;
        default:         print("ErrorNo:  ",errNbr," info:  ",errInfoNbr, " AxisNo:  ", axeNbr);
    }
    ErrorClear();
    print(""); print(" There is no error handlig → Exit()");
    Exit(0);
}

```

6.10 CAN_MultipleAx_DS402_ProfilePositionMode-ppm-Test.mc

```

#include "..\..\..\SDK\SDK_ApossC.mc"
// Set the node ids of the all nodes in the network
long busIds[] = {1, 4};
long numberOfBusIds = arraylen(busIds);
void errorHandler();
void faultMonitoring();
void posAbsStartForAllNodes(long pos);
void posRelStartForAllNodes(long distance);
void verifyAllNodesStateAchieved(long state);
void verfiyAllNodesTargetReached();
long main(void)
{
    long handle;
    long actState=0;
    long idIndex;
    //Reset CAN Bus 1
    handle = DefCanOut(0, 2);
    CanOut(handle, 0, 0x8100);
    //Reset CAN Bus 2
    handle = DefCanOut(100000, 2);
    CanOut(handle, 0, 0x8100);
    //Wait until nodes have been resetted
    Delay(2000);
    ErrorClear();
    //set errorHandler for master
    InterruptSetup(ERROR, errorHandler);
    //Monitior the Fault bit in the status word of the slave, all 25ms
    InterruptSetup(PERIOD, faultMonitoring, 25);
    //-----
    // Application Setup
    //-----
    for(idIndex = 0; idIndex < numberOfBusIds; idIndex++)
    {
        //Set the mode of operation to PPM
        sdkDS402_SetOpartionMode(busIds[idIndex], DS402_OP_PPM);
        actState = sdkDS402_GetActDriveState(busIds[idIndex]);
        if(actState == DS402_DRIVE_STATE_SWITCH_ON_DISABLED)
        {
            sdkDS402_TransitionToState(busIds[idIndex], DS402_DRIVE_STATE_READY_TO_SWITCH_ON);
        }
        //Set acceleration, deceleration and velocity
        sdkDS402_SetProfileMovementParameter(busIds[idIndex], 1000, 1000, 500);
        //Set the deceleration for quick stop
        sdkDS402_SetQuickStopDeceleration(busIds[idIndex], 5000);
    }
    verifyAllNodesStateAchieved(DS402_DRIVE_STATE_READY_TO_SWITCH_ON);
    //-----
    // End of Application Setup
    //-----
    //Transition to drive state 'Operation enabled'
    for(idIndex = 0; idIndex < numberOfBusIds; idIndex++)
    {
        //Transition to drive state 'Operation enabled'
        sdkDS402_TransitionToState(busIds[idIndex], DS402_DRIVE_STATE_OPERATION_ENABLED);
    }
    verifyAllNodesStateAchieved(DS402_DRIVE_STATE_OPERATION_ENABLED);
    while(1)
    {
        //Start absolute positioning to pos 100000 qc
        posAbsStartForAllNodes(100000);
        //Wait until the target position has been reached
        verfiyAllNodesTargetReached();
        //Start relative positioning of -100000 qc
        posRelStartForAllNodes(-100000);
        //Wait until the target position has been reached
        verfiyAllNodesTargetReached();
    }
    return(1);
}

```

Example Documentation

```

}
void posAbsStartForAllNodes(long pos)
{
    long idIndex;
    for(idIndex = 0; idIndex < numberOfBusIds; idIndex++)
    {
        //Start absolute positioning
        sdkDS402_PPM_PosAbsStart(busIds[idIndex], pos, DS402_PPM_IMMEDIATELY);
    }
}
void posRelStartForAllNodes(long distance)
{
    long idIndex;
    for(idIndex = 0; idIndex < numberOfBusIds; idIndex++)
    {
        //Start relative positioning of distance
        sdkDS402_PPM_PosRelStart(busIds[idIndex], distance, DS402_PPM_IMMEDIATELY);
    }
}
void verifyAllNodesStateAchieved(long state)
{
    long idIndex;
    long actState;
    long result;
    while(result != 1)
    {
        result = 1;
        for(idIndex = 0; idIndex < numberOfBusIds; idIndex++)
        {
            actState = sdkDS402_GetActDriveState(busIds[idIndex]);
            result = (actState == state) && result;
        }
    }
}
void verfiyAllNodesTargetReached()
{
    long idIndex;
    long result;
    print("Wait until all nodes target is reached");
    while(result != 1)
    {
        result = 1;
        for(idIndex = 0; idIndex < numberOfBusIds; idIndex++)
        {
            result = sdkDS402_PPM_TargetReached(busIds[idIndex]) && result;
        }
    }
}
void quickStopAllNodes()
{
    long idIndex;
    for(idIndex = 0; idIndex < numberOfBusIds; idIndex++)
    {
        sdkDS402_QuickStop(busIds[idIndex]);
    }
}
void faultMonitoring()
{
    long idIndex;
    for(idIndex = 0; idIndex < numberOfBusIds; idIndex++)
    {
        long statusWord = sdkDS402_ReadStatusWord(busIds[idIndex]);
        if(statusWord.i[DS402_SW_BIT_FAULT])
        {
            errorHandler();
        }
    }
}
void errorHandler()
{

```

```

long axeNbr      = ErrorAxis();
long errNbr      = ErrorNo();
long errInfoNbr = ErrorInfo();
long statusWord, idIndex, sdoAbortCode;
//Stop all nodes
quickStopAllNodes();
print("-----");
printf("Error %d: ",errNbr);
sdkErrorPrint_ApossIdeErrorDescription(errNbr);
print();
print("...Info: ",errInfoNbr, " AxisNo: ", axeNbr);
print("-----");
print();
switch(errNbr)
{
    case F_CANIO:
        printf("SDO Abort Code 0x%X: ", sdoAbortCode );
        sdkErrorPrint_SdoErrorDescription(SYS_PROCESS(SYS_CANOM_SDOABORT));
        print();
        print("Check Can baudrate & Can bus id");
    }
if(axeNbr == 0xFF && errNbr != F_CANIO)
{
    //The error is not releated to a certain axis
    for(idIndex = 0; idIndex < numberOfBusIds; idIndex++)
    {
        statusWord = sdkDS402_ReadStatusWord(busIds[idIndex]);
        if(statusWord.i[DS402_SW_BIT_FAULT])
        {
            print();
            printf("Bus Id %d, Slave device is in fault state\n", busIds[idIndex]);
            if(statusWord.i[DS402_SW_PPM_BIT_FOLLOWING_ERROR])
            {
                print("...Following Error");
            }
            //Reset the fault
            sdkDS402_ResetFault(busIds[idIndex]);
        }
    }
}
ErrorClear();
print();
print(" There is no error handlig → Exit()");
print("-----");
Exit(0);
}

```

6.11 CAN_MultipleAx_DS402_ProfilePositionMode-pvm-Test.mc

6.12 DeltaRobot_No_SM_EPOS4_ECAT.mc

```

#include "..\..\..\SDK\SDK_ApossC.mc"
// Parameters for the SDK function
#define C_AXIS1 0 // Axis module number
#define C_AXIS2 1 // Axis module number
#define C_AXIS3 2 // Axis module number
#define C_DRIVE_BUSID1 1000001 // The driveBusId is 1000000 plus the EtherCAT slave position in
the bus
#define C_DRIVE_BUSID2 1000002 // The driveBusId is 1000000 plus the EtherCAT slave position in
the bus
#define C_DRIVE_BUSID3 1000003 // The driveBusId is 1000000 plus the EtherCAT slave position in
the bus
#define C_EC_CYCLE_TIME 1 // Cycletime in milliseconds
#define C_EC_OFFSET 1 // Shift offset
#define C_PDO_NUMBER 1 // Used PDO number
#define C_OP_MODE_CSP 1 // Definition of the operation mode 1: CSP, 2: CSV

```

Example Documentation

```
#define C_MOTOR_MAX_RPM      2000          // Maximum velocity in RPM
#define C_AXISPOLARITY      1              // Definition of the polarity 0: Normal, 1: Inverse
#define C_HOMING_START      1
#define C_HOMING_WAIT_DONE  2
// EPOS4 Parameter for homing methode
#define C_EPOS4_HOMING_SPEED      20          // Homing Speed / Speed for
switch speed[rpm]
#define C_EPOS4_HOMING_POS_OFFSET      999          // Homing position offset [mA]
#define C_EPOS4_HOMING_CUR_THRESHOLD      2500          // Current threshold for
homing mode [mA]
#define C_EPOS4_HOMING_METH_PRES_POS      37          // Homing to present position
#define C_EPOS4_HOMING_METH_CUR_THRESHOLD_POS      -3          // Current Threshold Positive
Speed
#define C_EPOS4_HOMING_METH_CUR_THRESHOLD_NEG      -4          // Current Threshold Negative
Speed
#define C_EPOS4_HOMING_METHODE C_EPOS4_HOMING_METH_CUR_THRESHOLD_POS          // Used homing methode
// EPOS4 operation mode
#define C_EPOS4_OP_MODE_HOMING  6          // x6060 Operation mode "6: Homing mode."
#define C_EPOS4_OP_MODE_CSP      8          // x6060 Operation mode "8: CSP."
// Kinematics definition machine 1
#define C_MACH1_PARAM_0      800.0          // Length A is 80.0 mm.
#define C_MACH1_PARAM_1      1200.0          // Length B is 120.0 mm.
#define C_MACH1_PARAM_2      1800.0          // Length C is 180.0 mm.
#define C_MACH1_PARAM_3      300.0          // Length D is 30.0 mm.
#define C_MACH1_PARAM_4      -10.0          // The arm must not move higher than -10 degrees.
#define C_MACH1_PARAM_5      90.0          // The arm must not move lower than +10 degrees.
// Define work coordinate system
#define C_WORK1_SCALE      10          // Scale by 10 to convert from mm's to 1/10 of a mm.
#define C_WORK1_ORIENT      90          // Coordinate system must be rotated CLOCKWISE by 90 degrees
#define C_WORK1_TRANS_X      0          // Machine coordinate X offset (in micrometers) from home position to
Work origin.
#define C_WORK1_TRANS_Y      0          // Machine coordinate Y offset (in micrometers) from home position to
Work origin.
#define C_WORK1_TRANS_Z      -2600.0          // Machine coordinate Y offset (in micrometers) from home position to
Work origin.
// Global variable. Its also possible to set them as a local variable directly in the function.
// Only global variables can be used in a osziloscop or watchlist
long Robot;          // Kinematics handle for the robot that is active.
transform WorkToMachine;          // Work-to-Machine transformation.
dpoint TmpPoint;
dpoint TmpPointWork;
dpoint TmpPointMach;
// Declaration of local functioms
long SetupAxisParam(long axis, long EncCpt, long MaxRpm, long MaxAcc);
long EncCpt = 6400/4;          // Encoder resolution [cpt]
long MaxRpm = 2000;          // Max. speed [rpm]
long MaxAcc = 1;          // Max. acceleration [ms]: 0 -> MaxRpm
long main(void) {
    long slaveCount, i, homingState;
    homingState = C_HOMING_START;
    print("-----");
    print(" Test application EtherCAT Master with 3 EPOS4 drive");
    print("-----");
    ErrorClear();
    AxisControl(AXALL,ON);
    ECatMasterCommand(0x1000, 0);
    InterruptSetup(ERROR, ErrorHandler);
    //-----
    // Application Setup
    //-----
    slaveCount = sdkEtherCATMasterInitialize();
    print("slavecount: ",slaveCount);
    // initialising maxon drives
    sdkEpos4_SetupECatSdoParam(C_DRIVE_BUSID1, C_PDO_NUMBER, C_AXISPOLARITY, C_OP_MODE_CSP );
    sdkEpos4_SetupECatSdoParam(C_DRIVE_BUSID2, C_PDO_NUMBER, C_AXISPOLARITY, C_OP_MODE_CSP );
    sdkEpos4_SetupECatSdoParam(C_DRIVE_BUSID3, C_PDO_NUMBER, C_AXISPOLARITY, C_OP_MODE_CSP );
    for (i = 1; i <= 3; i++) {
        sdkEtherCATSetupDC(i, C_EC_CYCLE_TIME, C_EC_OFFSET);          // Setup EtherCAT DC (cycle_time [ms],
        offset [us]
    }
}
```

```

// starting the ethercat
sdkEtherCATMasterStart();
// setup EtherCAT bus module for csp mode
sdkEpos4_SetupECatBusModule(C_AXIS1, C_DRIVE_BUSID1, C_PDO_NUMBER, C_OP_MODE_CSP);
sdkEpos4_SetupECatBusModule(C_AXIS2, C_DRIVE_BUSID2, C_PDO_NUMBER, C_OP_MODE_CSP);
sdkEpos4_SetupECatBusModule(C_AXIS3, C_DRIVE_BUSID3, C_PDO_NUMBER, C_OP_MODE_CSP);
// setup virtual amplifier for csp mode
sdkEpos4_SetupECatVirtAmp(C_AXIS1, C_MOTOR_MAX_RPM, C_OP_MODE_CSP);
sdkEpos4_SetupECatVirtAmp(C_AXIS2, C_MOTOR_MAX_RPM, C_OP_MODE_CSP);
sdkEpos4_SetupECatVirtAmp(C_AXIS3, C_MOTOR_MAX_RPM, C_OP_MODE_CSP);
// setup irtual counter for csp mode
sdkEpos4_SetupECatVirtCntin(C_AXIS1, C_OP_MODE_CSP);
sdkEpos4_SetupECatVirtCntin(C_AXIS2, C_OP_MODE_CSP);
sdkEpos4_SetupECatVirtCntin(C_AXIS3, C_OP_MODE_CSP);
// Setup MACS axis parameters
SetupAxisParam(C_AXIS1, EncCpt, MaxRpm, MaxAcc);
SetupAxisParam(C_AXIS2, EncCpt, MaxRpm, MaxAcc);
SetupAxisParam(C_AXIS3, EncCpt, MaxRpm, MaxAcc);
//-----
// End of Application Setup
//-----
//-----
// Homing start
//-----
switch (homingState) {
case C_HOMING_START:
    print("\n -----");
    print("                        Homing Start.  Methode:  ", C_EPOS4_HOMING_METHODE);
    print("----- \n");
    AxisControl(AXALL,OFF);
    //Delay(100);
    for (i = 0; i < 3; i++) {
        //0x6040 Set "Control word, Bit40 (0x000x):  HALT."
        AxisControl(i,ON);
        VIRTAMP_PARAM(i, VIRTAMP_CNTRLW_PWRONENP) = 0x0F;
        VIRTAMP_PARAM(i, VIRTAMP_CNTRLW_PWRONENN) = 0x0F;
        // Homing setup
        SdoWriten( (1000001+i), 0x6099, 1, 4, C_EPOS4_HOMING_SPEED); // Homing Speed /
Speed for switch speed[rpm]
        SdoWriten( (1000001+i), 0x30B0, 0, 4, C_EPOS4_HOMING_POS_OFFSET); // homing position
offset [mA]
        SdoWriten( (1000001+i), 0x30B2, 0, 2, C_EPOS4_HOMING_CUR_THRESHOLD); // Current threshold
for homing mode [mA]
        SdoWriten( (1000001+i), 0x6098, 0, 1, C_EPOS4_HOMING_METHODE); // 0x6098 Set homing
method to "37" : Homing to present position."
        SdoWriten( (1000001+i), 0x6060, 0, 1, C_EPOS4_OP_MODE_HOMING); // x6060 Change
operation mode to "6: Homing mode."
        print("mode of OP: ",(SdoRead(1000001, 0x6061, 0)), " : ",(SdoRead(1000002, 0x6061, 0)),
: ",(SdoRead(1000003, 0x6061, 0));
        //0x6040 Set "Control word, Bit41 (0x0010): Homing start."
        VIRTAMP_PARAM(i, VIRTAMP_CNTRLW_PWRONENP) = 0x1F;
        VIRTAMP_PARAM(i, VIRTAMP_CNTRLW_PWRONENN) = 0x1F;
        AxisControl(i,ON);
    }
    while (1) {
        printf("Status:      %lx ; %lx ; %lx \n", (SdoRead(1000001, 0x6041, 0)),
(SdoRead(1000002, 0x6041, 0)), (SdoRead(1000003, 0x6041, 0)) );
        Delay(1000);
        if ( (((SdoRead(1000001, 0x6041, 0)) & 0x600) == 0x600) &&
(((SdoRead(1000002, 0x6041, 0)) & 0x600) == 0x600) &&
(((SdoRead(1000003, 0x6041, 0)) & 0x600) == 0x600) ) {
            printf("Status:      %lx ; %lx ; %lx \n", (SdoRead(1000001, 0x6041, 0)),
(SdoRead(1000002, 0x6041, 0)), (SdoRead(1000003, 0x6041, 0)) );
            print("\n ALL Homing done");
            for (i = 0; i < 3; i++) {
                VIRTAMP_PARAM(i, VIRTAMP_CNTRLW_PWRONENP) = 0x0F;
                VIRTAMP_PARAM(i, VIRTAMP_CNTRLW_PWRONENN) = 0x0F;
                AxisControl(i,OFF);
                Delay(4);
                SdoWriten( (1000001+i), 0x6060, 0, 1, C_EPOS4_OP_MODE_CSP ); // 0x6060

```

Example Documentation

```

    Change operation mode to "8: CSP mode."
        AxisControl(i,ON);
        Delay(4);
    }
    break;
}
}
homingState = C_HOMING_WAIT_DONE;
break;
case C_HOMING_WAIT_DONE:
    break;
}
AxisControl(C_AXIS1,ON, C_AXIS2,ON, C_AXIS3,ON);
print("-----");
print("          Setup Delta Kinematics          ");
print("----- \n");
//Setup kinematics
Robot = sdkSetupDeltaKinematics(C_AXIS1,C_AXIS2,C_AXIS3, C_MACH1_PARAM_0, C_MACH1_PARAM_1,
    C_MACH1_PARAM_2, C_MACH1_PARAM_3, C_MACH1_PARAM_4, C_MACH1_PARAM_5);
WorkToMachine = sdkSetupWorkCoordKinematics(WorkToMachine, C_WORK1_TRANS_X, C_WORK1_TRANS_Y,
    C_WORK1_TRANS_Z, C_WORK1_ORIENT, C_WORK1_SCALE);
//Set a Work-to-Machine transformation that will be used with the selected kinematics
KinematicsWorkToMc(Robot,WorkToMachine);
KinematicsSetup(ENABLE, Robot);
print("Kinematics enabled");
PathVel(Robot,10.0);
PathAcc(Robot,10.0);
PathDec(Robot,10.0);
PathJerk(Robot,10.0);
print("Set path speed");
// Set move point to drive
TmpPoint.x = 250;
TmpPoint.y = 0;
TmpPoint.z = 0;
//-----
// Test simple kinematics
//-----
// If set to 1 -> print kinematics tcp position
if (1)
{
    print("Axis -> Off. coordinate system can be tested");
    AxisControl(AXALL,OFF);
    while(1)
    {
        KinematicsCpoint(Robot, TmpPointWork, PATHCOORD_WORK);
        KinematicsCpoint(Robot, TmpPointMach, PATHCOORD_MACHINE);
        print("Work");
        print("X: ",TmpPointWork.x," Y: ",TmpPointWork.y);
        print("Machine");
        print("X: ",TmpPointMach.x," Y: ",TmpPointMach.y);
        Delay(500);
    }
}
// Else go to TmpPoint
else
{
    print("Got to TmpPoint in Machine Coord");
    PathMove(Robot, TmpPoint, PATHCOORD_MACHINE);          // Machine coordinate system
    AxisWaitReached(AXALL);
    //PathStart(Robot, 0, 0, WorkToMachine, 0, PATHSTART_NORMAL);
    AxisControl(AXALL,OFF);
}
return(0);
} // main
// Setup MACS axis parameters
long SetupAxisParam(long axis, long EncCpt, long MaxRpm, long MaxAcc)
{
    // Ensure that this corresponds to the encoder and EPOS4 configuration!
    AXE_PARAM(axis,POSENCQC)    = EncCpt*4;          // Set enc. resolution per turn [inc]
    AXE_PARAM(axis,POSENCREV)   = 1;                // Default: 1

```

```

AXE_PARAM(axis,FEEDDIST)    = 36000;           // Set feed resolution (= output) per turn [inc]
AXE_PARAM(axis,FEEDREV)     = 1;               // Default: 1
AXE_PARAM(axis, VELMAX)     = MaxRpm;          // Set max. velocity [rpm]
AXE_PARAM(axis, RAMPMIN)    = MaxAcc;          // Set max. acceleration [ms] (0- > MaxRpm)
AXE_PARAM(axis, KPROP)      = 0;               // Disable P-Gain of PID-Controller
AXE_PARAM(axis, KDER)       = 0;               // Disable D-Gain of PID-Controller
AXE_PARAM(axis, KINT)       = 0;               // Disable I-Gain of PID-Controller
AXE_PARAM(axis, POSERR)     = 2000000;         // Set max following error [inc]
return(1);
}

void ErrorHandler(void) {
    long ErrorAxe;
    ErrorAxe=ErrorAxis();
    print("Error ",ErrorNo()," errax: ",ErrorAxis() );
    if(ErrorNo() == 89) {
        printf("SDO Abort Code %lX", SYS_PROCESS(SYS_CANOM_SDOABORT) );
    }
    if (ErrorNo() == 8) {
        print(" Trackerror   time: ",Time() );
        print("ErrorAx: ",ErrorAxis() );
        Delay(10);
        // disable Master
        ECatMasterCommand(0x1000, 0);
    }
    if(ErrorNo() == 40) {
        print("Error Axis: ", ErrorAxe ," ErrorCode: ",
            radixstr(SdoRead(1000000+ErrorAxe+1,0x603F,0x00),16));
    }
    // ErrorClear();
    Delay(2000);
}

```

6.13 DeltaRobot_SM.mc

```

#include <SysDef.mh>
//Define encoder simulation or real motors on a MiniMACS6 → The file "Motors\Maxon_XYZ.mc" must be
    adapted before start
#define MINIMACS6_MOTORS      1
#define SIMULATION            2
#define MACHINE                SIMULATION
#include "include\UserInterface.mh"
#include "include\MachineSpecification.mh"
#include "include\Global_Variables.mh"
#include "include\General_Functions.mc"
#include "include\AxisSetup.mc"
/*
** Load the path arrays from the ".zbc" configuration file.
*/
#config "include\Delta_Show.zbc", CONFIGFLG_DIM
#include "include\StateMachine_Main.mc"
long main(void)
{
    /*
    ** Start by clearing any previous errors.
    */
    ErrorClear();
    /*
    ** Initialize the user parameters.      These are used to communicate
    ** with the Monitor dialog.
    */
    UserParamInit();
    /*
    ** Start the state machine.
    */
    print("Starting...");
    SmRun(Delta);
    print("Done.");
    return(0);
}

```

6.14 ECAT_1Ax_EPOS4-Test_cst.mc

```
#include "..\..\..\SDK\SDK_Apossc.mc"
// Parameters for the SDK function
#define C_AXIS1 0 // Axis module number
#define C_AXIS1_POLARITY 0 // Definition of the polarity 0: Normal, 1: Inverse
#define C_DRIVE_BUSID1 1000001 // The driveBusId is 1000000 plus the EtherCAT slave position in
    the bus
#define C_EC_CYCLE_TIME 1 // Cycletime in milliseconds
#define C_EC_OFFSET 1 // Shift offset
#define C_PDO_NUMBER 1 // Used PDO number
// Encoder settings & axis user units (MACS)
#define C_AXIS_ENCREV 4*1600 // Resolution of the encoder for position feed
    back in increments (quadcounts)
#define C_AXIS_POSENCREV 1 // Number of revolutions of the motor
#define C_AXIS_POSENCQC C_AXIS_ENCREV // Number of quadcounts in POSENCREV
    revolutions
#define C_AXIS_POSFACT_Z 1 // Number of revolutions of the input shaft
#define C_AXIS_POSFACT_N 1 // Number of revolutions of the output shaft
    in POSFACT_Z revolutions of the input shaft
#define C_AXIS_FEEDREV 1 // Number of revolutions of the gear box
    output shaft
#define C_AXIS_FEEDDIST C_AXIS_ENCREV // Distance travelled (in user units) in
    FEEDREV revolutions of the gear box output shaft [mm]
// Axis Movement Parameter
#define C_AXIS_MAX_RPM 2000 // Maximum velocity in RPM
#define C_AXIS_VELRES 100 // Velocity resolution, Scaling used for the
    velocity and acceleration/deceleration commands, default
#define C_AXIS_RAMPTYPE RAMPTYPE_JERKLIMITED // Defines the ramptype
#define C_AXIS_RAMPMIN 800 // Maximum acceleration
#define C_AXIS_JERKMIN 1000 // Minimum time (ms) required before reaching
    the maximum acceleration
#define C_AXIS_TRACKERR 0 // There is also a following error on EPOS4, could
    be set to zero on the MACS
// Axis MACS control loop settings
// MACS position control is not used
#define C_AXIS_KPROP 0
#define C_AXIS_KINT 0
#define C_AXIS_KDER 0
#define C_AXIS_KILIM 0
#define C_AXIS_KILIMTIME 0
#define C_AXIS_BANDWIDTH 1000
#define C_AXIS_FFVEL 1000
#define C_AXIS_KFFAC 0
#define C_AXIS_KFFDEC 0
long main(void) {
    long slaveCount, i,retval, axisIndex;
    long homingStateAx_0=0;
    print("-----");
    print(" Test application EtherCAT Master with 1 EPOS4 drive");
    print("-----");
    ErrorClear();
    AmpErrorClear(C_AXIS1); // Clear error on EPOS4
    InterruptSetup(ERROR, ErrorHandler);
    ECatMasterCommand(0x1000, 0);
    //-----
    // Application Setup
    //-----
    slaveCount = sdkEtherCATMasterInitialize();
    print("slavecount: ",slaveCount);
    // initialising maxon drives
    sdkEpos4_SetupECatSdoParam(C_DRIVE_BUSID1, C_PDO_NUMBER, C_AXIS1_POLARITY, EPOS4_OP_CST );
    sdkEtherCATMasterDoMapping();
    sdkEtherCATSetupDC(1, C_EC_CYCLE_TIME, C_EC_OFFSET); // Setup EtherCAT DC (cycle_time [ms], offset
        [us]
    // starting the ethercat
    sdkEtherCATMasterStart();
    // All axis have in this example the same parameters
    // setup EtherCAT bus module for cst mode
    sdkEpos4_SetupECatBusModule(C_AXIS1, C_DRIVE_BUSID1, C_PDO_NUMBER, EPOS4_OP_CST);
```

```
// setup virtual amplifier for cst mode
sdkEpos4_SetupECatVirtAmp(C_AXIS1, C_AXIS_MAX_RPM, EPOS4_OP_CST);
// setup irtual counter for cst mode
sdkEpos4_SetupECatVirtCntin(C_AXIS1, EPOS4_OP_CST);
// Movement parameters for the axis
sdkSetupAxisMovementParam( C_AXIS1,
                           C_AXIS_VELRES,
                           C_AXIS_MAX_RPM,
                           C_AXIS_RAMPTYPE,
                           C_AXIS_RAMPMIN,
                           C_AXIS_JERKMIN,
                           C_AXIS_TRACKERR
                           );

// Definition of the user units
sdkSetupAxisUserUnits(    C_AXIS1,
                           C_AXIS_POSENCREV,
                           C_AXIS_POSENCQC,
                           C_AXIS_POSFACT_Z,
                           C_AXIS_POSFACT_N,
                           C_AXIS_FEEDREV,
                           C_AXIS_FEEDDIST
                           );

// Position control setup
sdkSetupPositionPIDControlExt(    C_AXIS1,
                                   C_AXIS_KPROP,
                                   C_AXIS_KINT,
                                   C_AXIS_KDER,
                                   C_AXIS_KILIM,
                                   C_AXIS_KILIMTIME,
                                   C_AXIS_BANDWIDTH,
                                   C_AXIS_FFVEL,
                                   C_AXIS_KFFAC,
                                   C_AXIS_KFFDEC
                                   );

//-----
// End of Application Setup
//-----
ErrorClear();
//-----
// Homing start
//-----
// Homing setup
SdoWrite( C_DRIVE_BUSID1, EPOS4_HOMING_METHOD,          0,    EPOS4_HOMING_ACT_POSITION);
// 0x6098 Set homing method to "-4" : Homing Method -4 (Current Threshold Negative Speed).
SdoWrite( C_DRIVE_BUSID1, EPOS4_HOMING_SPEEDS,          1,    20);
// Homing Speed / Speed for switch speed [velocity units]
SdoWrite( C_DRIVE_BUSID1, EPOS4_HOMING_SPEEDS,          2,    20);
// Homing Speed / Speed for zero search [velocity units]
SdoWrite( C_DRIVE_BUSID1, EPOS4_HOMING_ACCELERATION,     0,    20);
// Homing acceleration [acceleration units]
SdoWrite( C_DRIVE_BUSID1, EPOS4_HOME_OFFSET_MOVE_DISTANCE, 0,    6400);
// Home offset move distance [position units]
SdoWrite( C_DRIVE_BUSID1, EPOS4_HOME_POSITION,          0,    0);
// Home position [position units]
SdoWrite( C_DRIVE_BUSID1, EPOS4_CURRENT_THRESHOLD_FOR_HOMING_MODE, 0,    1500);
// Current threshold for homing mode [mA]
print("...: EPOS4 Homing AxisNo: ",C_AXIS1," - Homing methode: ",SdoRead(C_DRIVE_BUSID1,
    EPOS4_HOMING_METHOD,0));
// Disable MACS trackerror for homing
AXE_PARAM(C_AXIS1,POSERR)=0;
// Homing statemachine
retval=0;
while(retval!=1)
{
    retval.i[0] =    sdkEpos4_AxisHomingStart(C_AXIS1, C_DRIVE_BUSID1, EPOS4_OP_CST, homingStateAx_0);
}
AxisControl(C_AXIS1,ON);
print("");
print("-----");
print("                Set torque in CST Mode                ");
```

Example Documentation

```

print("----- \n");
AxisControl(C_AXIS1, ON);
for(i=10;i>=0;i--)
{
    // The value is given in per thousand of "Motor rated torque"
    print("Set target torque → positive");
    AXE_PROCESS(C_AXIS1,REG_USERREFCUR)= 100;
    Delay(4000);
    print("Set target torque → negative");
    AXE_PROCESS(C_AXIS1,REG_USERREFCUR)= -100;
    Delay(4000);
    print("Set target torque → zero");
    AXE_PROCESS(C_AXIS1,REG_USERREFCUR)= 0;
    Delay(4000);
    print(i, " repetitions to go \n");
}
AxisControl(C_AXIS1, OFF);
print("Program done, Axis OFF ");
return(0);
}

void ErrorHandler(void)
{
    long axeNbr      = ErrorAxis();
    long errNbr      = ErrorNo();
    long errInfoNbr = ErrorInfo();
    long sdoAbortCode, eposErr;
    AxisControl(AXALL,OFF);
    print("-----");
    printf("Error %d: ",errNbr);
    sdkErrorPrint_ApossIdeErrorDescription(errNbr);
    print();
    print("...Info: ",errInfoNbr, " AxisNo: ", axeNbr);
    print("-----");
    print();
    switch(errNbr)
    {
        case F_AMP:      eposErr = SdoRead(C_DRIVE_BUSID1,EPOS4_ERROR_CODE,0x00);
                        printf("Error Axis:  %d, Epos4 Error 0x%X: ", axeNbr , eposErr);
                        sdkEpos4_PrintErrorDescription(eposErr);
                        print();
                        AmpErrorClear(axeNbr); // Clear error on EPOS4
                        break;
        case F_CANIO:    print("ErrorNo: ",errNbr," SlaveAddr(info): ",errInfoNbr);
                        ECatMasterInfo(0x1000, 22, sdoAbortCode);
                        printf("SDO Abort Code EtherCAT 0x%X\n", sdoAbortCode);
                        break;
        default:         print("ErrorNo: ",errNbr," info: ",errInfoNbr, " AxisNo: ", axeNbr);
    }
    ErrorClear();
    print(""); print(" There is no error handlig → Exit()");
    Exit(0);
}

```

6.15 ECAT_3Ax_Epos4-Test_csp.mc

```

#include "..\..\..\SDK\SDK_ApossC.mc"
// Parameters for the SDK function
#define C_AXIS1 0           // Axis module number
#define C_AXIS2 1           // Axis module number
#define C_AXIS3 2           // Axis module number
#define C_AXIS1_POLARITY 0  // Definition of the polarity 0: Normal, 1: Inverse
#define C_AXIS2_POLARITY 0  // Definition of the polarity 0: Normal, 1: Inverse
#define C_AXIS3_POLARITY 0  // Definition of the polarity 0: Normal, 1: Inverse
#define C_DRIVE_BUSID1 1000001 // The driveBusId is 1000000 plus the EtherCAT slave position in
                                the bus
#define C_DRIVE_BUSID2 1000002 // The driveBusId is 1000000 plus the EtherCAT slave position in
                                the bus

```

```

#define C_DRIVE_BUSID3 1000003          // The driveBusId is 1000000 plus the EtherCAT slave position in
    the bus
#define C_EC_CYCLE_TIME 1              // Cycletime in milliseconds
#define C_EC_OFFSET 1                  // Shift offset
#define C_PDO_NUMBER 1                // Used PDO number
// Encoder settings & axis user units (MACS)
#define C_AXIS_ENCREV 4*1600           // Resolution of the encoder for position feed
    back in increments (quadcounts)
#define C_AXIS_POSENCREV 1             // Number of revolutions of the motor
#define C_AXIS_POSENCQC C_AXIS_ENCREV // Number of quadcounts in POSENCREV
    revolutions
#define C_AXIS_POSFACT_Z 1             // Number of revolutions of the input shaft
#define C_AXIS_POSFACT_N 1            // Number of revolutions of the output shaft
    in POSFACT_Z revolutions of the input shaft
#define C_AXIS_FEEDREV 1              // Number of revolutions of the gear box
    output shaft
#define C_AXIS_FEEDDIST C_AXIS_ENCREV // Distance travelled (in user units) in
    FEEDREV revolutions of the gear box output shaft [mm]
// Axis Movement Parameter
#define C_AXIS_MAX_RPM 2000            // Maximum velocity in RPM
#define C_AXIS_VELRES 100             // Velocity resolution, Scaling used for the
    velocity and acceleration/deceleration commands, default
#define C_AXIS_RAMPTYPE RAMPTYPE_JERKLIMITED // Defines the ramptype
#define C_AXIS_RAMPMIN 800            // Maximum acceleration
#define C_AXIS_JERKMIN 1000           // Minimum time (ms) required before reaching
    the maximum acceleration
#define C_AXIS_TRACKERR 0              // There is also a following error on EPOS4, could
    be set to zero on the MACS
// Axis MACS control loop settings
// MACS position control is not used
#define C_AXIS_KPROP 0
#define C_AXIS_KINT 0
#define C_AXIS_KDER 0
#define C_AXIS_KILIM 0
#define C_AXIS_KILIMTIME 0
#define C_AXIS_BANDWIDTH 1000
#define C_AXIS_FFVEL 1000
#define C_AXIS_KFFAC 0
#define C_AXIS_KFFDEC 0
// Set the bus ids of the all axis in the network
long axis[] = {C_AXIS1, C_AXIS2, C_AXIS3};
long busIds[] = {C_DRIVE_BUSID1, C_DRIVE_BUSID2, C_DRIVE_BUSID3};
long numberOfAxis = arraylen(busIds);
long main(void) {
    long slaveCount, i, retval, axisIndex;
    long homingStateAx_0=0, homingStateAx_1=0, homingStateAx_2=0;
    print("-----");
    print(" Test application EtherCAT Master with 3 EPOS4 drive");
    print("-----");
    ErrorClear();
    AmpErrorClear(C_AXIS1); // Clear error on EPOS4
    AmpErrorClear(C_AXIS2); // Clear error on EPOS4
    AmpErrorClear(C_AXIS3); // Clear error on EPOS4
    InterruptSetup(ERROR, ErrorHandler);
    ECatMasterCommand(0x1000, 0);
    //-----
    // Application Setup
    //-----
    slaveCount = sdkEtherCATMasterInitialize();
    print("slavecount: ", slaveCount);
    // initialising maxon drives
    sdkEpos4_SetupECatSdoParam(C_DRIVE_BUSID1, C_PDO_NUMBER, C_AXIS1_POLARITY, EPOS4_OP_CSP );
    sdkEpos4_SetupECatSdoParam(C_DRIVE_BUSID2, C_PDO_NUMBER, C_AXIS2_POLARITY, EPOS4_OP_CSP );
    sdkEpos4_SetupECatSdoParam(C_DRIVE_BUSID3, C_PDO_NUMBER, C_AXIS3_POLARITY, EPOS4_OP_CSP );
    sdkEtherCATMasterDoMapping();
    for (i = 1; i <= 3; i++) {
        sdkEtherCATSetupDC(i, C_EC_CYCLE_TIME, C_EC_OFFSET); // Setup EtherCAT DC (cycle_time [ms],
        offset [us])
    }
    // starting the ethercat

```

Example Documentation

```

sdkEtherCATMasterStart();
// All axis have in this example the same parameters
for(axisIndex = 0; axisIndex < numberOfAxis; axisIndex++)
{
    // setup EtherCAT bus module for csp mode
    sdkEpos4_SetupECatBusModule(axis[axisIndex], busIds[axisIndex], C_PDO_NUMBER, EPOS4_OP_CSP);
    // setup virtual amplifier for csp mode
    sdkEpos4_SetupECatVirtAmp(axis[axisIndex], C_AXIS_MAX_RPM, EPOS4_OP_CSP);
    // setup irtual counter for csp mode
    sdkEpos4_SetupECatVirtCntin(axis[axisIndex], EPOS4_OP_CSP);
    // Movement parameters for the axis
    sdkSetupAxisMovementParam( axis[axisIndex],
                                C_AXIS_VELRES,
                                C_AXIS_MAX_RPM,
                                C_AXIS_RAMPTYPE,
                                C_AXIS_RAMPMIN,
                                C_AXIS_JERKMIN,
                                C_AXIS_TRACKERR
                                );

    // Definition of the user units
    sdkSetupAxisUserUnits(      axis[axisIndex],
                                C_AXIS_POSENCREV,
                                C_AXIS_POSENCQC,
                                C_AXIS_POSFACT_Z,
                                C_AXIS_POSFACT_N,
                                C_AXIS_FEEDREV,
                                C_AXIS_FEEDDIST
                                );

    // Position control setup
    sdkSetupPositionPIDControlExt(      axis[axisIndex],
                                        C_AXIS_KPROP,
                                        C_AXIS_KINT,
                                        C_AXIS_KDER,
                                        C_AXIS_KILIM,
                                        C_AXIS_KILIMTIME,
                                        C_AXIS_BANDWIDTH,
                                        C_AXIS_FFVEL,
                                        C_AXIS_KFFAC,
                                        C_AXIS_KFFDEC
                                        );
}
//-----
// End of Application Setup
//-----
ErrorClear();
//-----
// Homing start
//-----
// Homing setup
for(axisIndex = 0; axisIndex < numberOfAxis; axisIndex++)
{
    SdoWrite( busIds[axisIndex], EPOS4_HOMING_METHOD, 0,
EPOS4_HOMING_ACT_POSITION); // 0x6098 Set homing method to "-4" : Homing Method -4 (Current
Threshold Negative Speed).
    SdoWrite( busIds[axisIndex], EPOS4_HOMING_SPEEDS, 1, 20);
    // Homing Speed / Speed for switch speed [velocity units]
    SdoWrite( busIds[axisIndex], EPOS4_HOMING_SPEEDS, 2, 20);
    // Homing Speed / Speed for zero search [velocity units]
    SdoWrite( busIds[axisIndex], EPOS4_HOMING_ACCELERATION, 0, 20);
    // Homing acceleration [acceleration units]
    SdoWrite( busIds[axisIndex], EPOS4_HOME_OFFSET_MOVE_DISTANCE, 0, 6400);
    // Home offset move distance [position units]
    SdoWrite( busIds[axisIndex], EPOS4_HOME_POSITION, 0, 0);
    // Home position [position units]
    SdoWrite( busIds[axisIndex], EPOS4_CURRENT_THRESHOLD_FOR_HOMING_MODE, 0, 1500);
    // Current threshold for homing mode [mA]
    print("...: EPOS4 Homing AxisNo: ",axis," - Homing methode: ",SdoRead(busIds[axisIndex],
EPOS4_HOMING_METHOD,0));
    // Disable MACS trackerror for homing
    AXE_PARAM(axisIndex,POSERR)=0;
}

```

```

}
// Homing statemachine
retval=0;
while(retval!=7)
{
    retval.i[0] = sdkEpos4_AxisHomingStart(C_AXIS1, C_DRIVE_BUSID1, EPOS4_OP_CSP, homingStateAx_0);
    retval.i[1] = sdkEpos4_AxisHomingStart(C_AXIS2, C_DRIVE_BUSID2, EPOS4_OP_CSP, homingStateAx_1);
    retval.i[2] = sdkEpos4_AxisHomingStart(C_AXIS3, C_DRIVE_BUSID3, EPOS4_OP_CSP, homingStateAx_2);
}
AxisControl(C_AXIS1,ON, C_AXIS2,ON, C_AXIS3,ON);
print("-----");
print("                Movement in CSP Mode                ");
print("----- \n");
Vel(C_AXIS1, 40, C_AXIS2, 40, C_AXIS3, 40);
Acc(C_AXIS1, 30, C_AXIS2, 30, C_AXIS3, 30);
Dec(C_AXIS1, 30, C_AXIS2, 30, C_AXIS3, 30);
for(i=10;i>=0;i--)
{
    print("Start, move to target position");
    AxisPosAbsStart( C_AXIS1, 20000 , C_AXIS2, 20000, C_AXIS3, 20000 );
    AxisWaitReached(C_AXIS1,C_AXIS2,C_AXIS3);
    print("Target position is reached \n");
    print("Start, back to start position");
    AxisPosAbsStart( C_AXIS1, 0, C_AXIS2, 0, C_AXIS3, 0);
    AxisWaitReached(C_AXIS1,C_AXIS2,C_AXIS3);
    print("Start position is reached");
    print(i, " repetitions to go \n");
}
AxisControl(AXALL, OFF);
RecordStop(0, 0);
print("Program done, Axis OFF ");
return(0);
}

void ErrorHandler(void)
{
    long axeNbr      = ErrorAxis();
    long errNbr      = ErrorNo();
    long errInfoNbr = ErrorInfo();
    long sdoAbortCode, eposErr;
    AxisControl(AXALL,OFF);
    switch(errNbr)
    {
        case F_AMP:      eposErr = SdoRead(C_DRIVE_BUSID1,EPOS4_ERROR_CODE,0x00);
                        printf("Error Axis:  %d, Epos4 Error 0x%X: ", axeNbr , eposErr);
                        sdkEpos4_PrintErrorDescription(eposErr);
                        print();
                        AmpErrorClear(axeNbr); // Clear error on EPOS4
                        break;

        case F_CANIO:    print("ErrorNo:  ",errNbr," SlaveAddr(info):  ",errInfoNbr);
                        ECatMasterInfo(0x1000, 22, sdoAbortCode);
                        printf("SDO Abort Code EtherCAT 0x%X\n", sdoAbortCode);
                        break;

        default:         print("ErrorNo:  ",errNbr," info:  ",errInfoNbr, " AxisNo:  ", axeNbr);
    }
    ErrorClear();
    print(""); print(" There is no error handlig → Exit()");
    Exit(0);
}

```

6.16 EthernetSocket_OpenClient.mc

```

#include "..\..\..\SDK\SDK_ApossC.mc"
#include <SysDef.mh>
// Parameters for the example
#define SERVER_IP_ADDRESS      192.168.1.81    // IP address of the server socket.
#define SERVER_PORT_NUM       3000           // Port of the server socket.
#define CYCLE_TIME            1000           // ms → 0 to disable
// Local functions

```

Example Documentation

```

void EthernetSendData_sendCycle(void);
void FunctionEthernetHandler(void);
// global variables
long client_handle;
long status, status_old;
long main(void)
{
    // All errors are cleared at the beginning of the program.
    ErrorClear();
    // opening an ethernet client in case of a failure the program will be terminated
    print("EthernetOpenClient with IP: ",SERVER_IP_ADDRESS," and Port: ",SERVERT_PORT_NUM);
    client_handle = EthernetOpenClient (PROT_TCP, SERVER_IP_ADDRESS, SERVERT_PORT_NUM); // ipAddress,
        portNumber);
    print(" returned handler 'client_handle' : ",client_handle);
    if(client_handle<0)
    {
        sdkEthernetPrintGeneralResult (client_handle);
        print("Exit program");
        Exit(0);
    }
    // Wating for connection
    print("Wating for connection");
    while(3 != status)
    {
        status = EthernetGetConnectionStatus(client_handle);
        if(status != status_old)
        {
            sdkEthernetPrintConnectionStatus (status);
            status_old=status;
            Delay(1);
        }
    }
    // Start interrupts
    InterruptSetup(ETHERNET, FunctionEthernetHandler, client_handle);
    InterruptSetup(PERIOD, EthernetSendData_sendCycle, CYCLE_TIME);
    while (1) {
        status = EthernetGetConnectionStatus(client_handle);
        if(status != status_old)
        {
            sdkEthernetPrintConnectionStatus (status);
            status_old=status;
            Delay(1);
        }
    }
    return(0);
}
void FunctionEthernetHandler(void)
{
    long result;
    wchar receiveArray[50]="";
    wchar sendResponse[]="sendResponse";
    arrayset (receiveArray,0,result);
    result = EthernetReceiveTelegram(client_handle, receiveArray);
    print("\nwe received a Ethernet Telegram at time: ", Time());
    print("num of received data: ",result);
    print("receiveArray: ", receiveArray);
    result = sdkEthernetSendString(sendResponse, client_handle);
    sdkEthernetPrintGeneralResult (result);
    return;
}
void EthernetSendData_sendCycle(void)
{
    long result;
    wchar sendArray[30] = "";
    sprintf(sendArray, "EthernetSendData_sendCycle\n");
    result = sdkEthernetSendString(sendArray, client_handle);
    sdkEthernetPrintGeneralResult (result);
    return;
}

```

Sends a string as ethernet telegram Simplified usage of the function EthernetSendTelegram() and can be used to send a string.

Parameters

<i>sendArray</i>	Array which contains the data to be sent
<i>handle</i>	The handle returned by the EthernetOpenClient() or EthernetOpenServer() command.

Returns

value: See list in Ethernet Commands or use [sdkEthernetPrintGeneralResult\(\)](#)

value > 0 -

value = 0 No error, everything OK

value < 0 Error

```
#include "..\..\..\SDK\SDK_ApossC.mc"
#include <SysDef.mh>
// Parameters for the SDK function
#define MY_PORT_NUM      3000    // Define TCP Port number
#define CYCLE_TIME      1000    // ms → 0 to disable
// Local functions
void EthernetSendData_sendCycle(void);
void FunctionEthernetHandler(void);
// global variables
long server_handle;
long status,status_old;
long main(void)
{
    // All errors are cleared at the beginning of the program.
    ErrorClear();
    print("EthernetOpenServer with port: ",MY_PORT_NUM);
    // opening an ethernet server in case of a failure the program will be terminated
    server_handle = EthernetOpenServer(PROT_TCP, MY_PORT_NUM);
    print(" returned handler 'server_handle' : ",server_handle);
    if(server_handle<0)
    {
        print("Exit program");
        Exit(0);
    }
    // Wating for connection
    print("Wating for connection");
    while(3 != EthernetGetConnectionStatus(server_handle));
    // Start ethernet interrupts
    InterruptSetup(ETHERNET, FunctionEthernetHandler, server_handle);
    InterruptSetup(PERIOD, EthernetSendData_sendCycle, CYCLE_TIME);
    while (1)
    {
        status = EthernetGetConnectionStatus(server_handle);
        if(status != status_old)
        {
            sdkEthernetPrintConnectionStatus(status);
            status_old=status;
            Delay(1);
        }
    }
    return(0);
}
void FunctionEthernetHandler(void)
{
    long result;
    wchar receiveArray[50]="";
```

Example Documentation

```
wchar sendResponse[]="sendResponse";
arrayset(receiveArray,0,result);
result = EthernetReceiveTelegram(server_handle, receiveArray);
print("\nwe received a Ethernet Telegram at time: ", Time());
print("num of received data: ",result);
print("receiveArray: ", receiveArray);
result = sdkEthernetSendString(sendResponse, server_handle);
sdkEthernetPrintGeneralResult(result);
return;
}

void EthernetSendData_sendCycle(void)
{
    long result;
    wchar sendArray[30] = "";
    sprintf(sendArray, "EthernetSendData_sendCycle\n");
    result = sdkEthernetSendString(sendArray, server_handle);
    sdkEthernetPrintGeneralResult(result);
    return;
}
```

6.18 EthernetUDPSocket_OpenClient.mc

```
#include "..\..\..\SDK\SDK_ApossC.mc"
#include <SysDef.mh>
// Parameters for the example
#define SERVER_IP_ADDRESS      192.168.1.3 // IP address of the server socket.
#define SERVER_PORT_NUM       3000        // Port of the server socket.
#define CYCLE_TIME             200        // ms → 0 to disable
// Local functions
void EthernetSendData_sendCycle(void);
void FunctionEthernetHandler(void);
// global variables
long client_handle;
long status, status_old;
long main(void)
{
    long ip = SERVER_IP_ADDRESS;
    // All errors are cleared at the beginning of the program.
    ErrorClear();
    // opening an ethernet client in case of a failure the program will be terminated
    print("EthernetOpenClient (UDP) to IP:
        ", ((ip)&0xFF), ".", ((ip>>8)&0xFF), ".", ((ip>>16)&0xFF), ".", ((ip>>24)&0xFF), " and Port:
        ", SERVER_PORT_NUM);
    client_handle = EthernetOpenClient(PROT_UDP, SERVER_IP_ADDRESS, SERVER_PORT_NUM); // ipAddress,
        portNumber);
    ip = SYS_INFO(SYS_IP_ADDRESS_CONTROLLER);
    print(" IP address of this client is:
        ", ((ip)&0xFF), ".", ((ip>>8)&0xFF), ".", ((ip>>16)&0xFF), ".", ((ip>>24)&0xFF) );
    print(" returned handler 'client_handle' : ",client_handle);
    if(client_handle<0)
    {
        sdkEthernetPrintGeneralResult(client_handle);
        print("Exit program");
        Exit(0);
    }
    // Wating for connection
    print("Wating for connection");
    while(3 != status)
    {
        status = EthernetGetConnectionStatus(client_handle);
        if(status != status_old)
        {
            sdkEthernetPrintConnectionStatus(status);
            status_old=status;
            Delay(1);
        }
    }
    // Start interrupts
```

```
InterruptSetup(ETHERNET, FunctionEthernetHandler, client_handle);
InterruptSetup(PERIOD, EthernetSendData_sendCycle, CYCLE_TIME);
while (1) {
    status = EthernetGetConnectionStatus(client_handle);
    if(status != status_old)
    {
        sdkEthernetPrintConnectionStatus(status);
        status_old=status;
        Delay(1);
    }
}
return(0);
}
void FunctionEthernetHandler(void)
{
    long result = 0;
    wchar receiveArray[50]="";
    arrayset(receiveArray,0,50);
    result = EthernetReceiveTelegram(client_handle, receiveArray);
    print("we received a UDP Telegram at time: ", Time()," size: ",result," data: ",receiveArray);
    return;
}
void EthernetSendData_sendCycle(void)
{
    long result;
    wchar sendArray[40] = "";
    sprintf(sendArray, "UDP Client says Hello time: %d\n",Time());
    result = EthernetSendTelegram(client_handle, sendArray, arraylen(sendArray));
    print("SendCyclic time: ",Time());
    if (result != 0) {
        sdkEthernetPrintGeneralResult(result);
    }
    return;
}
```

6.19 EthernetUDPSocket_OpenServer.mc

```
#include "..\..\..\SDK\SDK_ApossC.mc"
#include <SysDef.mh>
// Parameters for the SDK function
#define MY_PORT_NUM      3000    // Define TCP Port number
#define CYCLE_TIME      1000    // ms → 0 to disable
// Local functions
void EthernetSendData_sendCycle(void);
void FunctionEthernetHandler(void);
// global variables
long server_handle;
long status,status_old;
long main(void)
{
    long ip = SYS_INFO(SYS_IP_ADDRESS_CONTROLLER);
;
    // All errors are cleared at the beginning of the program.
    ErrorClear();
    print("EthernetOpenServer (UDP) with port: ",MY_PORT_NUM);
    print("IP address of this Server is:
        ",((ip)&0xFF),".",((ip>>8)&0xFF),".",((ip>>16)&0xFF),".",((ip>>24)&0xFF));
    // opening an ethernet server in case of a failure the program will be terminated
    server_handle = EthernetOpenServer(PROT_UDP, MY_PORT_NUM);
    print(" returned handler 'server_handle' : ",server_handle);
    if(server_handle<0)
    {
        print("Exit program");
        Exit(0);
    }
    // Wating for connection
    print("Wating for connection");
    while(3 != EthernetGetConnectionStatus(server_handle));
```

Example Documentation

```
// Start ethernet interrupts
InterruptSetup(ETHERNET, FunctionEthernetHandler, server_handle);
InterruptSetup(PERIOD, EthernetSendData_sendCycle, CYCLE_TIME);
while (1)
{
    status = EthernetGetConnectionStatus(server_handle);
    if(status != status_old)
    {
        sdkEthernetPrintConnectionStatus(status);
        status_old=status;
        Delay(1);
    }
}
return(0);
}

void FunctionEthernetHandler(void)
{
    long result = 0;
    wchar receiveArray[50]="";
    arrayset(receiveArray,0,50);
    result = EthernetReceiveTelegram(server_handle, receiveArray);
    print("we received a UDP Telegram at time: ", Time()," size: ",result," data: ",receiveArray);
    return;
}

void EthernetSendData_sendCycle(void)
{
    long result;
    wchar sendArray[40] = "";
    sprintf(sendArray, "UDP Server says Hello time: %d\n",Time());
    result = EthernetSendTelegram(server_handle, sendArray, arraylen(sendArray));
    print("SendCyclic time: ",Time());
    if (result != 0) {
        sdkEthernetPrintGeneralResult(result);
    }
    return;
}
```

6.20 HowToUse_entire_SDK.mc

```
#include <SysDef.mh>
#include "SDK\SDK_ApossC.mc"
long main(void)
{
    // This sample program shows how the SDK can be used. The folder "SDK" must be on the same level as
    // the example program "ExampleProgramSDK.mc".
    // In order to keep the memory requirements of the SDK as low as possible, the compiler must be set
    // to include only referenced files.
    // The procedure for this is shown in detail in the supplied "ApossCSKGettingStarted.pdf".
    //
    // Short version: The checkbox "Settings -> Compiler-> Include only refereced files" must be
    // selected.
    return(0);
}
```

6.21 HowToUse_individual_SDK.mc

261

```
#include <SysDef.mh>
//These include files have no real link in this example.
#include "SDK_Axis_Setup.mc"
#include "SDK_Encoder_Setup.mc"
#include "SDK_Amplifier_General.mc"
long main(void)
{
    // This sample program shows how the SDK can be used as a individual SDK. The desired files
    // can be copied from the folder structure of the SDK and integrated into a separate storage system.
    // In this example, the documents have the same hierarchy level.
    //
    // Attention: There are different SDK files which are dependent on other SDK files. Therefore both
    // files have to be added and if necessary the include path has to be adjusted.
    //
    // The procedure for this is shown in detail in the supplied "ApossCSKGettingStarted.pdf".
    return(0);
}
```

6.22 InformationGeneral.mc

```
#include <SysDef.mh>
#include "..\..\SDK\SDK_ApossC.mc"
long main(void)
{
    sdkInfoPrintSoftware();
    sdkInfoPrintHardware();
    sdkInfoPrintAxesPos();
    sdkInfoPrintPosPID();
    return(0);
}
```

6.23 Maxon_EC45_flat_1ax_BC_Enc.mc

```
#include <SysDef.mh>
// Relative path to the main folder ApossC_SDK_Vxx.xx
#include "..\..\..\..\..\SDK\SDK_ApossC.mc"
// Setting the axis number. Hall ports are always bound to the respective axis number.
#define AXIS1_NO 0 // Axis module number
#define AXIS1_ENCPORT 0 // Encoder port number. Usually, module instance 0 is connected to X1
// and so on. Please refer to product manual
// Axe settings
#define EC45FLAT_ENCRES 4*2048 // Resolution of the encoder for position feed back in
// increments (quadcounts)
#define EC45FLAT_ENC_LATCHTYPE 0 // Default latchType Encoder Z signal
#define EC45FLAT_ENC_LATCHPARAM 0 // Not used for Encoder Z Latch
#define EC45FLAT_ENC_LATCHSLOPE HWLATCH_SLOPE_RISING // Defines the slope of the trigger signal (Default 1)
#define EC45FLAT_CONTROLMODE HWAMP_MODE_POS_CUR // Define control typ
#define EC45FLAT_HALL_ALIGNMENT HALL_ALIGNMENT_120DEGREE // Hall alignment 120° standard
#define EC45FLAT_POLEPAIRS 8 // Number of pole pairs
#define EC45FLAT_CONTCUR 2970 // Nomial continious current allowed in mA
#define EC45FLAT_MAXCUR EC45FLAT_CONTCUR*1.25 // Maximal current allowed in mA
#define EC45FLAT_THERMAL_TIME 13400 // Thermal time constant of the winding
#define EC45FLAT_MAX_RPM 5500 // Maximum velocity in RPM
#define EC45FLAT_CURKPROP 2200 // Proportional factor of current controller
#define EC45FLAT_CURKINT 300 // Integral factor of current controller
#define EC45FLAT_CURKILIM 18000 // Integral limit of current controller
// not used in this example → HWAMP_MODE_POS_CUR
#define EC45FLAT_VELKPROP 2000 // Proportional factor of velocity controller
#define EC45FLAT_VELKINT 250 // Integral factor of velocity controller
#define EC45FLAT_VELKILIM 1000 // Integral limit of velocity controller
#define EC45FLAT_VELRES 100 // Velocity resolution, Scaling used for the velocity
// and acceleration/deceleration commands
```

Example Documentation

```
#define EC45FLAT_RAMPTYPE    RAMPTYPE_JERKLIMITED    // Defines the ramptype
#define EC45FLAT_RAMPMIN    1000                    // Maximum acceleration
#define EC45FLAT_JERKMIN    500                     // Minimum time (ms) required before reaching the
    maximum acceleration
#define EC45FLAT_POSEERR    500                     // Maximal track/ position error allowed in qc
#define EC45FLAT_DIRECTION  1                       // User units have normal orientation. Increasing
    encoder values result in increasing user positions.
#define EC45FLAT_KPROP      1000                    // Proportional value for PID position control loop
#define EC45FLAT_KINT       0                       // Integral value for PID position control loop
#define EC45FLAT_KDER       2000                    // Derivative value for PID position control loop
#define EC45FLAT_KILIM      1000                    // Limit value for the integral sum of the PID
    position control loop
#define EC45FLAT_KILIMTIME  0                       // Time used to increase or decrease the integral
    limit
#define EC45FLAT_BANDWIDTH  1000                    // Bandwidth within which the PID filter is active.
    1000 equals to 100% velocity setpoint
#define EC45FLAT_FFVEL      0                       // Velocity Feed forward
#define EC45FLAT_KFFACC     100                     // Acceleration Feed forward
#define EC45FLAT_KFFDEC     100                     // Deceleration Feed Forward
#define EC45FLAT_POSENCREV  1                       // Number of revolutions of the motor
#define EC45FLAT_POSENCQC   EC45FLAT_ENCRESES       // Number of quadcounts in POSENCREV revolutions
#define EC45FLAT_POSFACT_Z  169                     // Number of revolutions of the input shaft
#define EC45FLAT_POSFACT_N  9                       // Number of revolutions of the output shaft in
    POSFACT_Z revolutions of the input shaft
#define EC45FLAT_FEEDREV    1                       // Number of revolutions of the gear box output shaft
#define EC45FLAT_FEEDDIST   36000                   // Distance travelled (in user units) in FEEDREV
    revolutions of the gear box output shaft
// Function delkratation
long setupEC45Flat_BC_Inc(long axisNo, long encPort);
long main(void)
{
    ErrorClear();
    DefOrigin(AXIS1_NO);
    // Setup axis & amplifier
    setupEC45Flat_BC_Inc(AXIS1_NO, AXIS1_ENCPORT);
    // Activate the axis
    AxisControl(AXIS1_NO, ON);
    // Start an endless movement with continuous move control with 50% velocity and 50% acceleration
    sdkStartContinuousMove(AXIS1_NO, 50, 50);
while(1)
{
    // Print axis information all 500 ms
    sdkInfoPrintAxesPos();
    Delay(500);
}
return(0);
}
long setupEC45Flat_BC_Inc(long axisNo, long encPort)
{
    // Encoder setup
    sdkSetupIncEncoder(
        axisNo,
        encPort,
        EC45FLAT_ENCRESES,
        EC45FLAT_ENC_LATCHTYPE,
        EC45FLAT_ENC_LATCHPARAM,
        EC45FLAT_ENC_LATCHSLOPE
    );

    // Amplifier setup
    sdkSetupAmpBldcMotor(
        axisNo,
        EC45FLAT_HALL_ALIGNMENT,
        EC45FLAT_CONTROLMODE,
        EC45FLAT_POLEPAIRS,
        EC45FLAT_MAXCUR,
        EC45FLAT_ENCRESES,
        EC45FLAT_MAX_RPM
    );

    // Current control setup
    sdkSetupCurrentPControl(
        axisNo,
        EC45FLAT_CURKPROP,
        EC45FLAT_CURKINT,
```

```

        EC45FLAT_CURKILIM
    );

    // Velocity control setup - not used at the moment
    sdkSetupVelocityPIControl(    axisNo,
        EC45FLAT_VELKPROP,
        EC45FLAT_VELKINT,
        EC45FLAT_VELKILIM
    );

    // Movement parameters for the axis
    sdkSetupAxisMovementParam( axisNo,
        EC45FLAT_VELRES,
        EC45FLAT_MAX_RPM,
        EC45FLAT_RAMPTYPE,
        EC45FLAT_RAMPMIN,
        EC45FLAT_JERKMIN,
        EC45FLAT_POSERR
    );

    // Set the direction of the axis
    sdkSetupAxisDirection(    axisNo,
        EC45FLAT_DIRECTION);

    // Position control setup
    sdkSetupPositionPIDControlExt(    axisNo,
        EC45FLAT_KPROP,
        EC45FLAT_KINT,
        EC45FLAT_KDER,
        EC45FLAT_KILIM,
        EC45FLAT_KILIMTIME,
        EC45FLAT_BANDWIDTH,
        EC45FLAT_FFVEL,
        EC45FLAT_KFFACC,
        EC45FLAT_KFFDEC
    );

    // Definition of the user units
    sdkSetupAxisUserUnits(    axisNo,
        EC45FLAT_POSENCREV,
        EC45FLAT_POSENCQC,
        EC45FLAT_POSFACT_Z,
        EC45FLAT_POSFACT_N,
        EC45FLAT_FEEDREV,
        EC45FLAT_FEEDDIST
    );

    // Setup virtual I2T
    sdkSetupVirtualI2T(axisNo, EC45FLAT_CONTCUR, EC45FLAT_THERMAL_TIME);
    return(1);
}

// $X {KPROP,1,1,0,-1,0,-1,0,(-1),-1},0x2300,12,0
// $X {KDER,1,1,0,-1,0,-1,0,(-1),-1},0x2300,13,0
// $X {KINT,1,1,0,-1,0,-1,0,(-1),-1},0x2300,14,0

```

6.24 Maxon_EC45_flat_1ax_BC_Hall.mc

```

#include <SysDef.mh>
// Relative path to the main folder ApossC_SDK_Vxx.xx
#include "..\..\..\..\..\SDK\SDK_ApossC.mc"
// Setting the axis number. Hall ports are always bound to the respective axis number.
#define AXIS1_NO 0 // Axis module number
// Axe settings
#define EC45FLAT_CONTROLMODE HWAMP_MODE_POS_CUR // Define control typ
#define EC45FLAT_HALL_ALIGNMENT HALL_ALIGNMENT_120DEGREE // Hall alignment 120° standard
#define EC45FLAT_HALL_DIRECT -1 // Hall direction is inverted when using
    maxon motors
#define EC45FLAT_POLEPAIRS 8 // Number of pole pairs
#define EC45FLAT_CONTCUR 2970 // Nomial continious current allowed in mA
#define EC45FLAT_MAXCUR EC45FLAT_CONTCUR*1.25 // Maximal current allowed in mA
#define EC45FLAT_THERMAL_TIME 13400 // Thermal time constant of the winding
#define EC45FLAT_MAX_RPM 5500 // Maximum velocity in RPM
#define EC45FLAT_CURKPROP 2200 // Proportional factor of current controller
#define EC45FLAT_CURKINT 300 // Integral factor of current controller

```

Example Documentation

```
#define EC45FLAT_CURKILIM      18000                // Integral limit of current controller
// not used in this example → HWAMP_MODE_POS_CUR
#define EC45FLAT_VELKPROP      2000                // Proportional factor of velocity controller
#define EC45FLAT_VELKINT       250                 // Integral factor of velocity controller
#define EC45FLAT_VELKILIM      1000               // Integral limit of velocity controller
#define EC45FLAT_VELRES        100                 // Velocity resolution, Scaling used for the velocity
// and acceleration/deceleration commands
#define EC45FLAT_RAMPTYPE      RAMPTYPE_JERKLIMITED // Defines the ramptype
#define EC45FLAT_RAMPMIN       1000                // Maximum acceleration
#define EC45FLAT_JERKMIN       500                 // Minimum time (ms) required before reaching the
// maximum acceleration
#define EC45FLAT_POSERR        500                 // Maximal track/ position error allowed in qc
#define EC45FLAT_DIRECTION     1                   // User units have normal orientation. Increasing
// encoder values result in increasing user positions.
#define EC45FLAT_KPROP         1000                // Proportional value for PID position control loop
#define EC45FLAT_KINT          0                   // Integral value for PID position control loop
#define EC45FLAT_KDER          2000                // Derivative value for PID position control loop
#define EC45FLAT_KILIM         1000                // Limit value for the integral sum of the PID
// position control loop
#define EC45FLAT_KILIMTIME     0                   // Time used to increase or decrease the integral
// limit
#define EC45FLAT_BANDWIDTH     1000                // Bandwidth within which the PID filter is active.
// 1000 equals to 100% velocity setpoint
#define EC45FLAT_FFVEL         0                   // Velocity Feed forward
#define EC45FLAT_KFFACC        100                 // Acceleration Feed forward
#define EC45FLAT_KFFDEC        100                 // Deceleration Feed Forward
#define EC45FLAT_POSENCREV     1                   // Number of revolutions of the motor
#define EC45FLAT_POSENCQC      EC45FLAT_POLEPAIRS*6 // Number of quadcounts in POSENCREV revolutions
#define EC45FLAT_POSFACT_Z     169                 // Number of revolutions of the input shaft
#define EC45FLAT_POSFACT_N     9                   // Number of revolutions of the output shaft in
// POSFACT_Z revolutions of the input shaft
#define EC45FLAT_FEEDREV       1                   // Number of revolutions of the gear box output
// shaft
#define EC45FLAT_FEEDDIST      36000                // Distance travelled (in user units) in FEEDREV
// revolutions of the gear box output shaft
// Function delkratation
long setupEC45Flat_BC_Hall(long axisNo);
long main(void)
{
    ErrorClear();
    DefOrigin(AXIS1_NO);
    // Setup axis & amplifier
    setupEC45Flat_BC_Hall(AXIS1_NO);
    // Activate the axis
    AxisControl(AXIS1_NO, ON);
    // Start an endless movement with continuous move control with 50% velocity and 50% acceleration
    sdkStartContinuousMove(AXIS1_NO, 50, 50);
while(1)
{
    // Print axis information all 500 ms
    sdkInfoPrintAxesPos();
    Delay(500);
}
    return(0);
}
long setupEC45Flat_BC_Hall(long axisNo)
{
    // Hall setup for using it as encoder
    sdkSetupHallEncoder(    axisNo, HALL_ALIGNMENT_120DEGREE, EC45FLAT_HALL_DIRECT);
    // Amplifier setup
    HWAMP_PARAM(axisNo, HWAMP_COMMTYPE) = HWAMP_COMMTYPE_BLDC;                // Set motor type
    HWAMP_PARAM(axisNo, HWAMP_MODE)     = EC45FLAT_CONTROLMODE;                // Set controller priciple
    HWAMP_PARAM(axisNo, HWAMP_POLES)    = EC45FLAT_POLEPAIRS;                 // Number of pole pairs
    HWAMP_PARAM(axisNo, HWAMP_MAXCUR)   = EC45FLAT_MAXCUR;                     // Max current in mA
    HWAMP_PARAM(axisNo, HWAMP_MAXRPM)   = EC45FLAT_MAX_RPM;                    // Given in RPM
    // Current control setup
    sdkSetupCurrentPIControl(    axisNo,
                                EC45FLAT_CURKPROP,
                                EC45FLAT_CURKINT,
                                EC45FLAT_CURKILIM
```

```

    );
// Velocity control setup - not used at the moment
sdkSetupVelocityPIControl(    axisNo,
                             EC45FLAT_VELKPROP,
                             EC45FLAT_VELKINT,
                             EC45FLAT_VELKILIM
                             );
// Movement parameters for the axis
sdkSetupAxisMovementParam( axisNo,
                           EC45FLAT_VELRES,
                           EC45FLAT_MAX_RPM,
                           EC45FLAT_RAMPTYPE,
                           EC45FLAT_RAMPMIN,
                           EC45FLAT_JERKMIN,
                           EC45FLAT_POSERR
                           );
// Set the direction of the axis
sdkSetupAxisDirection(    axisNo,
                          EC45FLAT_DIRECTION);
// Position control setup
sdkSetupPositionPIDControlExt(    axisNo,
                                  EC45FLAT_KPROP,
                                  EC45FLAT_KINT,
                                  EC45FLAT_KDER,
                                  EC45FLAT_KILIM,
                                  EC45FLAT_KILIMTIME,
                                  EC45FLAT_BANDWIDTH,
                                  EC45FLAT_FFVEL,
                                  EC45FLAT_KFFACC,
                                  EC45FLAT_KFFDEC
                                  );
// Definition of the user units
sdkSetupAxisUserUnits(    axisNo,
                         EC45FLAT_POSENCREV,
                         EC45FLAT_POSENCQC,
                         EC45FLAT_POSFACT_Z,
                         EC45FLAT_POSFACT_N,
                         EC45FLAT_FEEDREV,
                         EC45FLAT_FEEDDIST
                         );
// Setup virtual I2T
sdkSetupVirtualI2T(axisNo, EC45FLAT_CONTCUR, EC45FLAT_THERMAL_TIME);
return(1);
}

```

6.25 Maxon_EC45_flat_1ax_SC_Hall_Inc.mc

```

#include <SysDef.mh>
// Relative path to the main folder ApossC_SDK_Vxx.xx
#include "..\..\..\..\..\SDK\SDK_ApossC.mc"
// Setting the axis number. Hall ports are always bound to the respective axis number.
#define AXIS1_NO 0 // Axis module number
#define AXIS1_ENCPORT 0 // Encoder port number. Usually, module instance 0 is connected to X1
// and so on. Please refer to product manual
// Axe settings
#define EC45FLAT_ENCRESP 4*2048 // Resolution of the encoder for position feed back in
// increments (quadcounts)
#define EC45FLAT_ENC_LATCHTYPE 0 // Default latchTyp Encoder Z signal
#define EC45FLAT_ENC_LATCHPARAM 0 // Not used for Encoder Z Latch
#define EC45FLAT_ENC_LATCHSLOPE HWLATCH_SLOPE_RISING // Defines the slope of the trigger signal (Default 1)
#define EC45FLAT_CONTROLMODE HWAMP_MODE_POS_CUR // Define control typ
#define EC45FLAT_HALL_ALIGNMENT HALL_ALIGNMENT_120DEGREE // Hall alignment 120° standard
#define EC45FLAT_POLEPAIRS 8 // Number of pole pairs
#define EC45FLAT_CONTCUR 2970 // Nomial continious current allowed in mA
#define EC45FLAT_MAXCUR EC45FLAT_CONTCUR*1.25 // Maximal current allowed in mA
#define EC45FLAT_THERMAL_TIME 13400 // Thermal time constant of the winding
#define EC45FLAT_MAX_RPM 5500 // Maximum velocity in RPM
#define EC45FLAT_CURKPROP 2200 // Proportional factor of current controller

```

Example Documentation

```
#define EC45FLAT_CURKINT          300          // Integral factor of current controller
#define EC45FLAT_CURKILIM        18000        // Integral limit of current controller
// not used in this example → HWAMP_MODE_POS_CUR
#define EC45FLAT_VELKPROP        2000        // Proportional factor of velocity controller
#define EC45FLAT_VELKINT         250         // Integral factor of velocity controller
#define EC45FLAT_VELKILIM        1000        // Integral limit of velocity controller
#define EC45FLAT_VELRES          100         // Velocity resolution, Scaling used for the velocity
and acceleration/deceleration commands
#define EC45FLAT_RAMPTYPE        RAMPTYPE_JERKLIMITED // Defines the ramptype
#define EC45FLAT_RAMPMIN         1000        // Maximum acceleration
#define EC45FLAT_JERKMIN         500         // Minimum time (ms) required before reaching the
maximum acceleration
#define EC45FLAT_POSERR          500         // Maximal track/ position error allowed in qc
#define EC45FLAT_DIRECTION       1           // User units have normal orientation. Increasing
encoder values result in increasing user positions.
#define EC45FLAT_KPROP           1000        // Proportional value for PID position control loop
#define EC45FLAT_KINT             0          // Integral value for PID position control loop
#define EC45FLAT_KDER             2000       // Derivative value for PID position control loop
#define EC45FLAT_KILIM           1000       // Limit value for the integral sum of the PID
position control loop
#define EC45FLAT_KILIMTIME       0           // Time used to increase or decrease the integral
limit
#define EC45FLAT_BANDWIDTH       1000        // Bandwidth within which the PID filter is active.
1000 equals to 100% velocity setpoint
#define EC45FLAT_FFVEL           0           // Velocity Feed forward
#define EC45FLAT_KFFACC          100         // Acceleration Feed forward
#define EC45FLAT_KFFDEC          100         // Deceleration Feed Forward
#define EC45FLAT_POSENCREV       1           // Number of revolutions of the motor
#define EC45FLAT_POSENCQC        EC45FLAT_ENCRES // Number of quadcounts in POSENCREV revolutions
#define EC45FLAT_POSFACT_Z       169        // Number of revolutions of the input shaft
#define EC45FLAT_POSFACT_N       9          // Number of revolutions of the output shaft in
POSFACT_Z revolutions of the input shaft
#define EC45FLAT_FEEDREV         1           // Number of revolutions of the gear box output shaft
#define EC45FLAT_FEEDDIST        36000      // Distance travelled (in user units) in FEEDREV
revolutions of the gear box output shaft
// Function delkratation
long setupEC45Flat_SC_Hall_Inc(long axisNo, long encPort);
long main(void)
{
    ErrorClear();
    DefOrigin(AXIS1_NO);
    // Setup axis & amplifier
    setupEC45Flat_SC_Hall_Inc(AXIS1_NO, AXIS1_ENCPORT);
    // Activate the axis
    AxisControl(AXIS1_NO, ON);
    // Start an endless movement with continuous move control with 50% velocity and 50% acceleration
    sdkStartContinuousMove(AXIS1_NO, 50, 50);
while(1)
{
    // Print axis information all 500 ms
    sdkInfoPrintAxesPos();
    Delay(500);
}
    return(0);
}
long setupEC45Flat_SC_Hall_Inc(long axisNo, long encPort)
{
    // Encoder setup
    sdkSetupIncEncoder(
        axisNo,
        encPort,
        EC45FLAT_ENCRES,
        EC45FLAT_ENC_LATCHTYPE,
        EC45FLAT_ENC_LATCHPARAM,
        EC45FLAT_ENC_LATCHSLOPE
    );

    // Amplifier setup
    sdkSetupAmpHallPmsmMotor(
        axisNo,
        EC45FLAT_CONTROLMODE,
        EC45FLAT_POLEPAIRS,
        EC45FLAT_MAXCUR,
```

```

        EC45FLAT_ENCRESP,
        EC45FLAT_MAX_RPM,
        -1
    );

// Current control setup
sdkSetupCurrentPIControl(    axisNo,
                             EC45FLAT_CURKPROP,
                             EC45FLAT_CURKINT,
                             EC45FLAT_CURKILIM
                             );

// Velocity control setup - not used at the moment
sdkSetupVelocityPIControl(    axisNo,
                             EC45FLAT_VELKPROP,
                             EC45FLAT_VELKINT,
                             EC45FLAT_VELKILIM
                             );

// Movement parameters for the axis
sdkSetupAxisMovementParam(    axisNo,
                             EC45FLAT_VELRES,
                             EC45FLAT_MAX_RPM,
                             EC45FLAT_RAMPTYPE,
                             EC45FLAT_RAMPMIN,
                             EC45FLAT_JERKMIN,
                             EC45FLAT_POSEERR
                             );

// Set the direction of the axis
sdkSetupAxisDirection(        axisNo,
                             EC45FLAT_DIRECTION);

// Position control setup
sdkSetupPositionPIDControlExt(    axisNo,
                                 EC45FLAT_KPROP,
                                 EC45FLAT_KINT,
                                 EC45FLAT_KDER,
                                 EC45FLAT_KILIM,
                                 EC45FLAT_KILIMTIME,
                                 EC45FLAT_BANDWIDTH,
                                 EC45FLAT_FFVEL,
                                 EC45FLAT_KFFACC,
                                 EC45FLAT_KFFDEC
                                 );

// Definition of the user units
sdkSetupAxisUserUnits(        axisNo,
                             EC45FLAT_POSENCREV,
                             EC45FLAT_POSENCQC,
                             EC45FLAT_POSFACT_Z,
                             EC45FLAT_POSFACT_N,
                             EC45FLAT_FEEDREV,
                             EC45FLAT_FEEDDIST
                             );

// Setup virtual I2T
sdkSetupVirtualI2T(axisNo, EC45FLAT_CONTCUR, EC45FLAT_THERMAL_TIME);
return(1);
}

// $X {KPROP,1,1,0,-1,0,-1,0,(-1),-1},0x2300,12,0
// $X {KDER,1,1,0,-1,0,-1,0,(-1),-1},0x2300,13,0
// $X {KINT,1,1,0,-1,0,-1,0,(-1),-1},0x2300,14,0

```

6.26 Maxon_ECi30_2ax_SC_Hall_Inc.mc

```

#include <SysDef.mh>
// Relative path to the main folder ApossC_SDK_Vxx.xx
#include "..\..\..\..\..\SDK\SDK_ApossC.mc"
// Setting the axis number. Hall ports are always bound to the respective axis number.
#define AXIS1_NO          0    // Axis module number
#define AXIS1_ENCPORTR    0    // Encoder port number. Usually, module instance 0 is connected to X1
                             and so on. Please refer to product manual
#define AXIS2_NO          1    // Axis module number

```

Example Documentation

```
#define AXIS2_ENCPORT          1    // Encoder port number.  Usually, module instance 0 is connected to X1
                                // and so on.  Please refer to product manual

// Axe settings
#define ECi30_ENCRES          4*4096    // Resolution of the encoder for position feed back in
                                // increments (quadcounts)
#define ECi30_ENC_LATCHTYPE    0    // Default latchTyp Encoder Z signal
#define ECi30_ENC_LATCHPARAM    0    // Not used for Encoder Z Latch
#define ECi30_ENC_LATCHSLOPE    HWLATCH_SLOPE_RISING// Defines the slope of the trigger signal (Default 1)
#define ECi30_CONTROLMODE      HWAMP_MODE_POS_CUR    // Define control typ
#define ECi30_HALL_ALIGNMENT    HALL_ALIGNMENT_120DEGREE    // Hall alignment 120° standard
#define ECi30_POLEPAIRS        4    // Number of pole pairs
#define ECi30_CONTCUR          2280    // Nomial continious current allowed in mA
#define ECi30_MAXCUR            ECi30_CONTCUR*1.25    // Maximal current allowed in mA
#define ECi30_THERMAL_TIME      27600    // Thermal time constant of the winding
#define ECi30_MAX_RPM          4000    // Maximum velocity in RPM
#define ECi30_CURKPROP          1500    // Proportional factor of current controller
#define ECi30_CURKINT           150    // Integral factor of current controller
#define ECi30_CURKILIM          32767    // Integral limit of current controller
// not used in this example → HWAMP_MODE_POS_CUR
#define ECi30_VELKPROP          0    // Proportional factor of velocity controller
#define ECi30_VELKINT           0    // Integral factor of velocity controller
#define ECi30_VELKILIM          0    // Integral limit of velocity controller
#define ECi30_VELRES            100    // Velocity resolution, Scaling used for the velocity
                                // and acceleration/deceleration commands
#define ECi30_RAMPTYPE          RAMPTYPE_JERKLIMITED    // Defines the ramptype
#define ECi30_RAMPMIN           1000    // Maximum acceleration
#define ECi30_JERKMIN           500    // Minimum time (ms) required before reaching the
                                // maximum acceleration
#define ECi30_POSERR            500    // Maximal track/ position error allowed in qc
#define ECi30_DIRECTION         1    // User units have normal orientation.  Increasing
                                // encoder values result in increasing user positions.
#define ECi30_KPROP             80    // Proportional value for PID position control loop
#define ECi30_KINT              0    // Integral value for PID position control loop
#define ECi30_KDER              200    // Derivative value for PID position control loop
#define ECi30_KILIM             1000    // Limit value for the integral sum of the PID
                                // position control loop
#define ECi30_KILIMTIME         0    // Time used to increase or decrease the integral
                                // limit
#define ECi30_BANDWIDTH         1000    // Bandwidth within which the PID filter is active.
                                // 1000 equals to 100% velocity setpoint
#define ECi30_FFVEL             0    // Velocity Feed forward
#define ECi30_KFFACC            0    // Acceleration Feed forward
#define ECi30_KFFDEC            0    // Deceleration Feed Forward
#define ECi30_POSENCREV         1    // Number of revolutions of the motor
#define ECi30_POSENCQC          ECi30_ENCRES    // Number of quadcounts in POSENCREV revolutions
#define ECi30_POSFACT_Z         1    // Number of revolutions of the input shaft
#define ECi30_POSFACT_N         1    // Number of revolutions of the output shaft in
                                // POSFACT_Z revolutions of the input shaft
#define ECi30_FEEDREV           1    // Number of revolutions of the gear box output shaft
#define ECi30_FEEDDIST          360    // Distance travelled (in user units) in FEEDREV
                                // revolutions of the gear box output shaft
// Function delkratation
long setupECi30_SC_Hall_Inc(long axisNo, long encPort);
long main(void)
{
    long direction = 1;
    long turns = 100;
    ErrorClear();
    DefOrigin(AXALL);
    // Setup axis & amplifier
    setupECi30_SC_Hall_Inc(AXIS1_NO,AXIS1_ENCPORT);
    setupECi30_SC_Hall_Inc(AXIS2_NO,AXIS2_ENCPORT);
    // Activate the axis
    AxisControl(AXIS1_NO, ON, AXIS2_NO, ON);
    // Set movements parameter
    Acc(AXIS1_NO,50,AXIS2_NO,50);    // Set acceleration to 50% of VELMAX
    Dec(AXIS1_NO,50,AXIS2_NO,50);    // Set deceleration to 50% of RAMPMIN
    Vel(AXIS1_NO,50,AXIS2_NO,50);    // Set velocity to 50% of RAMPMIN
    while(1)
    {
```

```
// Do xx turns: direction * distance travelled per revolution * turns p.e. 1 * 36000 1/100
degree * 100 turns
AxisPosAbsStart (AXIS1_NO,direction*ECi30_FEEDDIST*turns,AXIS2_NO,direction*ECi30_FEEDDIST*turns);
AxisWaitReached (AXIS1_NO,AXIS2_NO);
direction = direction*-1;
}
return(0);
}
long setupECi30_SC_Hall_Inc(long axisNo, long encPort)
{
    // Encoder setup
    sdkSetupIncEncoder(
        axisNo,
        encPort,
        ECi30_ENCRESP,
        ECi30_ENC_LATCHTYPE,
        ECi30_ENC_LATCHPARAM,
        ECi30_ENC_LATCHSLOPE
    );

    // Amplifier setup
    sdkSetupAmpHallPmsmMotor(
        axisNo,
        ECi30_CONTROLMODE,
        ECi30_POLEPAIRS,
        ECi30_MAXCUR,
        ECi30_ENCRESP,
        ECi30_MAX_RPM,
        -1
    );

    // Current control setup
    sdkSetupCurrentPIControl(
        axisNo,
        ECi30_CURKPROP,
        ECi30_CURKINT,
        ECi30_CURKILIM
    );

    // Velocity control setup - not used at the moment
    sdkSetupVelocityPIControl(
        axisNo,
        ECi30_VELKPROP,
        ECi30_VELKINT,
        ECi30_VELKILIM
    );

    // Movement parameters for the axis
    sdkSetupAxisMovementParam(
        axisNo,
        ECi30_VELRES,
        ECi30_MAX_RPM,
        ECi30_RAMPTYPE,
        ECi30_RAMPMIN,
        ECi30_JERKMIN,
        ECi30_POSERR
    );

    // Set the direction of the axis
    sdkSetupAxisDirection(
        axisNo,
        ECi30_DIRECTION);

    // Position control setup
    sdkSetupPositionPIDControlExt(
        axisNo,
        ECi30_KPROP,
        ECi30_KINT,
        ECi30_KDER,
        ECi30_KILIM,
        ECi30_KILIMTIME,
        ECi30_BANDWIDTH,
        ECi30_FFVEL,
        ECi30_KFFACC,
        ECi30_KFFDEC
    );

    // Definition of the user units
    sdkSetupAxisUserUnits(
        axisNo,
        ECi30_POSENCREV,
        ECi30_POSENCQC,
        ECi30_POSFACT_Z,
        ECi30_POSFACT_N,
        ECi30_FEEDREV,

```

```

                                ECi30_FEEDDIST
                                );
// Setup virtual I2T
sdkSetupVirtualI2T(axisNo,ECi30_CONTCUR, ECi30_THERMAL_TIME);
return(1);
}
//$X {KPROP,1,1,0,-1,0,-1,0,(-1),-1},0x2300,12,0
//$X {KDER,1,1,0,-1,0,-1,0,(-1),-1},0x2300,13,0
//$X {KINT,1,1,0,-1,0,-1,0,(-1),-1},0x2300,14,0

```

6.27 Maxon_ECi40_1ax_SC_OL_Inc.mc

```

#include <SysDef.mh>
// Relative path to the main folder ApossC_SDK_Vxx.xx
#include "..\..\..\..\..\SDK\SDK_ApossC.mc"
#define C_AXIS1 0 // Axis module number
#define C_AXIS1_ENCPORT 0 // Encoder port number. Usually, module instance
0 is connected to X1 and so on. Please refer to product manual
#define C_AXIS1_ENCRESP 1024 // Resolution of the encoder for position feed
back in increments (quadcounts)
#define C_AXIS1_ENC_LATCHTYPE 1 // Defines the latch type: Digital input
#define C_AXIS1_ENC_LATCHPARAM 1 // Latch on digital input 1
#define C_AXIS1_ENC_LATCHSLOPE HWLATCH_SLOPE_RISING // Defines the slope of the trigger signal
(Default 1)
#define C_MOTOR1_CONTROLMODE HWAMP_MODE_POS_VEL_CUR // Define control typ
#define C_MOTOR1_POLEPAIRS 8 // Number of pole pairs
#define C_MOTOR1_MAXCUR 8670 // Maximal current allowed in mA
#define C_MOTOR1_MAX_RPM 7740 // Maximum velocity in RPM
#define C_MOTOR1_CURKPROP 320 // Proportional factor of current controller
#define C_MOTOR1_CURKINT 90 // Integral factor of current controller
#define C_MOTOR1_CURKILIM 32767 // Integral limit of current controller
#define C_MOTOR1_VELKPROP 2000 // Proportional factor of velocity controller
#define C_MOTOR1_VELKINT 250 // Integral factor of velocity controller
#define C_MOTOR1_VELKILIM 1000 // Integral limit of velocity controller
#define C_MOTOR1_BRUSHLESS 1 // Mode of alignment: For brushless motors
#define C_MOTOR1_ALIGN_CUR 2000 // Current for the alignment function in mA
#define C_AXIS1_VELRES 1000 // Velocity resolution, Scaling used for the
velocity and acceleration/deceleration commands
#define C_AXIS1_RAMPTYPE RAMPTYPE_JERKLIMITED // Defines the ramptype
#define C_AXIS1_RAMPMIN 1000 // Maximum acceleration
#define C_AXIS1_JERKMIN 100 // Minimum time (ms) required before reaching the
maximum acceleration
#define C_AXIS1_TRACKERR C_AXIS1_ENCRESP // Maximal track/ position error allowed in qc
#define C_AXIS1_DIRECTION 1 // User units have normal orientation. Increasing
encoder values result in increasing user positions.
#define C_AXIS1_KPROP 80 // Proportional value for PID position control
loop
#define C_AXIS1_KINT 0 // Integral value for PID position control loop
#define C_AXIS1_KDER 600 // Derivative value for PID position control loop
#define C_AXIS1_KILIM 1000 // Limit value for the integral sum of the PID
position control loop
#define C_AXIS1_KILIMTIME 0 // Time used to increase or decrease the integral
limit
#define C_AXIS1_BANDWIDTH 1000 // Bandwidth within which the PID filter is
active. 1000 equals to 100% velocity setpoint
#define C_AXIS1_FFVEL 1000 // Velocity Feed forward
#define C_AXIS1_KFFAC 0 // Acceleration Feed forward
#define C_AXIS1_KFFDEC 0 // Deceleration Feed Forward
#define C_AXIS1_POSENCREV 1 // Number of revolutions of the motor
#define C_AXIS1_POSENCQC C_AXIS1_ENCRESP // Number of quadcounts in POSENCREV revolutions
#define C_AXIS1_POSFACT_Z 1 // Number of revolutions of the input shaft
#define C_AXIS1_POSFACT_N 1 // Number of revolutions of the output shaft in
POSFACT_Z revolutions of the input shaft
#define C_AXIS1_FEEDREV 1 // Number of revolutions of the gear box output
shaft
#define C_AXIS1_FEEDDIST C_AXIS1_ENCRESP // Distance travelled (in user units) in FEEDREV
revolutions of the gear box output shaft
long main(void)

```

```

{
    long checkMotorAlignment;
    // Encoder setup
    sdkSetupIncEncoder(
        C_AXIS1,
        C_AXIS1_ENCPORT,
        C_AXIS1_ENCREV,
        C_AXIS1_ENC_LATCHTYPE,
        C_AXIS1_ENC_LATCHPARAM,
        C_AXIS1_ENC_LATCHSLOPE
    );

    // Amplifier setup
    sdkSetupAmpPmsmMotor(
        C_AXIS1,
        C_MOTOR1_CONTROLMODE,
        C_MOTOR1_POLEPAIRS,
        C_MOTOR1_MAXCUR,
        C_AXIS1_ENCREV,
        C_MOTOR1_MAX_RPM
    );

    // Current control setup
    sdkSetupCurrentPIControl(
        C_AXIS1,
        C_MOTOR1_CURKPROP,
        C_MOTOR1_CURKINT,
        C_MOTOR1_CURKILIM
    );

    // Velocity control setup
    sdkSetupVelocityPIControl(
        C_AXIS1,
        C_MOTOR1_VELKPROP,
        C_MOTOR1_VELKINT,
        C_MOTOR1_VELKILIM
    );

    // Movement parameters for the axis
    sdkSetupAxisMovementParam(
        C_AXIS1,
        C_AXIS1_VELRES,
        C_MOTOR1_MAX_RPM,
        C_AXIS1_RAMPTYPE,
        C_AXIS1_RAMPMIN,
        C_AXIS1_JERKMIN,
        C_AXIS1_TRACKERR
    );

    // Set the direction of the axis
    sdkSetupAxisDirection(
        C_AXIS1,
        C_AXIS1_DIRECTION);

    // Position control setup
    sdkSetupPositionPIDControlExt(
        C_AXIS1,
        C_AXIS1_KPROP,
        C_AXIS1_KINT,
        C_AXIS1_KDER,
        C_AXIS1_KILIM,
        C_AXIS1_KILIMTIME,
        C_AXIS1_BANDWIDTH,
        C_AXIS1_FFVEL,
        C_AXIS1_KFFAC,
        C_AXIS1_KFFDEC
    );

    // Definition of the user units
    sdkSetupAxisUserUnits(
        C_AXIS1,
        C_AXIS1_POSENCREV,
        C_AXIS1_POSENCQC,
        C_AXIS1_POSFACT_Z,
        C_AXIS1_POSFACT_N,
        C_AXIS1_FEEDREV,
        C_AXIS1_FEEDDIST
    );

    // Motor Alignment
    checkMotorAlignment = sdkMotorAlignment(
        C_AXIS1,
        C_MOTOR1_BRUSHLESS,
        C_MOTOR1_MAXCUR,
        C_MOTOR1_ALIGN_CUR);

    if (checkMotorAlignment < 0)

```

Example Documentation

```
{
    print("Sorry alignment didn't work we exit");
    Exit(0);
}
// Activate the axis
AxisControl(C_AXIS1, ON);
// Start an endless movement with continuous move control
sdkStartContinuousMove(C_AXIS1, 1000, 100);
while(1)
{
    // Print axis information all 500 ms
    sdkInfoPrintAxesPos();
    Delay(500);
}
return(0);
}
```

6.28 Maxon_ECi40_3ax_SC_OL_Inc.mc

```
#include <SysDef.mh>
// Relative path to the main folder ApossC_SDK_Vxx.xx
#include "..\..\..\..\SDK\SDK_ApossC.mc"
#define C_AXIS1 0 // Axis module number
#define C_AXIS1_ENCPORT 0 // Encoder port number. Usually, module instance
0 is connected to X1 and so on. Please refer to product manual
#define C_AXIS2 1 // Axis module number
#define C_AXIS2_ENCPORT 1 // Encoder port number. Usually, module instance
0 is connected to X1 and so on. Please refer to product manual
#define C_AXIS3 2 // Axis module number
#define C_AXIS3_ENCPORT 2 // Encoder port number. Usually, module instance
0 is connected to X1 and so on. Please refer to product manual
#define C_AXIS1_DIRECTION 1 // User units have normal orientation. Increasing
encoder values result in increasing user positions.
#define C_AXIS2_DIRECTION -1 // User units are inverted. Increasing encoder
values result in decreasing user positions.
#define C_AXIS3_DIRECTION -1 // User units are inverted. Increasing encoder
values result in decreasing user positions.
#define C_AXIS_ENCRESP 1024 // Resolution of the encoder for position feed
back in increments (quadcounts)
#define C_AXIS_ENC_LATCHTYPE 1 // Defines the latch type: Digital input
#define C_AXIS_ENC_LATCHPARAM 1 // Latch on digital input 1
#define C_AXIS_ENC_LATCHSLOPE HWLATCH_SLOPE_RISING // Defines the slope of the trigger signal
(Default 1)
#define C_MOTOR_CONTROLMODE HWAMP_MODE_POS_VEL_CUR // Define control typ
#define C_MOTOR_POLEPAIRS 8 // Number of pole pairs
#define C_MOTOR_MAXCUR 8670 // Maximal current allowed in mA
#define C_MOTOR_MAX_RPM 7740 // Maximum velocity in RPM
#define C_MOTOR_CURKPROP 320 // Proportional factor of current controller
#define C_MOTOR_CURKINT 90 // Integral factor of current controller
#define C_MOTOR_CURKILIM 32767 // Integral limit of current controller
#define C_MOTOR_VELKPROP 2000 // Proportional factor of velocity controller
#define C_MOTOR_VELKINT 250 // Integral factor of velocity controller
#define C_MOTOR_VELKILIM 1000 // Integral limit of velocity controller
#define C_MOTOR_BRUSHLESS 1 // Mode of alignment: For brushless motors
#define C_MOTOR_ALIGN_CUR 2000 // Current for the alignment function in mA
#define C_AXIS_VELRES 1000 // Velocity resolution, Scaling used for the
velocity and acceleration/deceleration commands
#define C_AXIS_RAMPTYPE RAMPTYPE_JERKLIMITED // Defines the ramptype
#define C_AXIS_RAMPMIN 1000 // Maximum acceleration
#define C_AXIS_JERKMIN 100 // Minimum time (ms) required before reaching the
maximum acceleration
#define C_AXIS_TRACKERR C_AXIS_ENCRESP // Maximal track/ position error allowed in qc
#define C_AXIS_KPROP 80 // Proportional value for PID position control
loop
#define C_AXIS_KINT 0 // Integral value for PID position control loop
#define C_AXIS_KDER 600 // Derivative value for PID position control loop
#define C_AXIS_KILIM 1000 // Limit value for the integral sum of the PID
position control loop
```

```

#define C_AXIS_KILIMTIME          0           // Time used to increase or decrease the integral
    limit
#define C_AXIS_BANDWIDTH          1000        // Bandwidth within which the PID filter is
    active. 1000 equals to 100% velocity setpoint
#define C_AXIS_FFVEL              1000        // Velocity Feed forward
#define C_AXIS_KFFAC              0           // Acceleration Feed forward
#define C_AXIS_KFFDEC             0           // Deceleration Feed Forward
#define C_AXIS_POSENCREV          1           // Number of revolutions of the motor
#define C_AXIS_POSENCQC           C_AXIS_ENCREV // Number of quadcounts in POSENCREV revolutions
#define C_AXIS_POSFACT_Z          1           // Number of revolutions of the input shaft
#define C_AXIS_POSFACT_N          1           // Number of revolutions of the output shaft in
    POSFACT_Z revolutions of the input shaft
#define C_AXIS_FEEDREV            1           // Number of revolutions of the gear box output
    shaft
#define C_AXIS_FEEDDIST           C_AXIS_ENCREV // Distance travelled (in user units) in FEEDREV
    revolutions of the gear box output shaft
long main(void)
{
    long i, checkMotorAlignment;
    // Encoder setup
    sdkSetupIncEncoder(
        C_AXIS1,
        C_AXIS1_ENCPORT,
        C_AXIS_ENCREV,
        C_AXIS_ENC_LATCHTYPE,
        C_AXIS_ENC_LATCHPARAM,
        C_AXIS_ENC_LATCHSLOPE
    );

    sdkSetupIncEncoder(
        C_AXIS2,
        C_AXIS2_ENCPORT,
        C_AXIS_ENCREV,
        C_AXIS_ENC_LATCHTYPE,
        C_AXIS_ENC_LATCHPARAM,
        C_AXIS_ENC_LATCHSLOPE
    );

    sdkSetupIncEncoder(
        C_AXIS3,
        C_AXIS3_ENCPORT,
        C_AXIS_ENCREV,
        C_AXIS_ENC_LATCHTYPE,
        C_AXIS_ENC_LATCHPARAM,
        C_AXIS_ENC_LATCHSLOPE
    );

    // Set the direction of the axis
    sdkSetupAxisDirection(
        C_AXIS1,
        C_AXIS1_DIRECTION);
    sdkSetupAxisDirection(
        C_AXIS2,
        C_AXIS2_DIRECTION);
    sdkSetupAxisDirection(
        C_AXIS3,
        C_AXIS3_DIRECTION);

    for(i=C_AXIS1;i<=C_AXIS3;i++)
    {
        // Amplifier setup
        sdkSetupAmpPmsmMotor(
            i,
            C_MOTOR_CONTROLMODE,
            C_MOTOR_POLEPAIRS,
            C_MOTOR_MAXCUR,
            C_AXIS_ENCREV,
            C_MOTOR_MAX_RPM
        );

        // Current control setup
        sdkSetupCurrentPIControl(
            i,
            C_MOTOR_CURKPROP,
            C_MOTOR_CURKINT,
            C_MOTOR_CURKILIM
        );

        // Velocity control setup
        sdkSetupVelocityPIControl(
            i,
            C_MOTOR_VELKPROP,
            C_MOTOR_VELKINT,
            C_MOTOR_VELKILIM
        );
    }
}

```

Example Documentation

```
// Movement parameters for the axis
sdkSetupAxisMovementParam( i,
    C_AXIS_VELRES,
    C_MOTOR_MAX_RPM,
    C_AXIS_RAMPTYPE,
    C_AXIS_RAMPMIN,
    C_AXIS_JERKMIN,
    C_AXIS_TRACKERR
);

// Position control setup
sdkSetupPositionPIDControlExt( i,
    C_AXIS_KPROP,
    C_AXIS_KINT,
    C_AXIS_KDER,
    C_AXIS_KILIM,
    C_AXIS_KILIMTIME,
    C_AXIS_BANDWIDTH,
    C_AXIS_FFVEL,
    C_AXIS_KFFAC,
    C_AXIS_KFFDEC
);

// Definition of the user units
sdkSetupAxisUserUnits( i,
    C_AXIS_POSENCREV,
    C_AXIS_POSENCQC,
    C_AXIS_POSFACT_Z,
    C_AXIS_POSFACT_N,
    C_AXIS_FEEDREV,
    C_AXIS_FEEDDIST
);

}

// Motor Alignment
checkMotorAlignment = sdkMotorMultiAlignment(
    C_AXIS1,
    3,
    C_MOTOR_BRUSHLESS,
    C_MOTOR_MAXCUR,
    C_MOTOR_ALIGN_CUR);

if(checkMotorAlignment<0)
{
    print("Sorry alignment didn't work we exit");
    Exit(0);
}

for(i=C_AXIS1;i<=C_AXIS3;i++)
{
    AxisControl(i, ON);
    // Start an endless movement with continuous move control
    sdkStartContinuousMove(i, 1000, 500);
}

while(1)
{
    // Print axis information all 500 ms
    sdkInfoPrintAxesPos();
    Delay(2000);
}

return(0);
}//$X {REG_ACTPOS,1,1,0,-1,0,-1,0,(-1),-1},0x2500,1,0
//$X {User Parameter (1),1,1,0,-1,0,-1,0,(-1),-1},0x2201,1,0
//$X {REG_TRACKERR,1,1,0,-1,0,-1,0,(-1),-1},0x2500,6,0
//$X {REG_TRACKERR,1,1,0,-1,0,-1,0,(-1),-1},0x2501,6,0
//$X {REG_TRACKERR,1,1,0,-1,0,-1,0,(-1),-1},0x2502,6,0
//$X {REG_TRACKERR,1,1,0,-1,0,-1,0,(-1),-1},0x2503,6,0
```

6.29 Maxon_ECi52_1ax_SC_SSI.mc

```
#include <SysDef.mh>
// Relative path to the main folder ApossC_SDK_Vxx.xx
#include "..\..\..\..\..\SDK\SDK_ApossC.mc"
```

```

// Setting the axis number. Hall ports are always bound to the respective axis number.
#define AXIS1_NO 0 // Axis module number
#define AXIS1_ENCPORT 0 // Encoder port number. Usually, module instance 0 is connected to X1
and so on. Please refer to product manual
// Axis settings
#define ECI52_ENCRESP 4096 // Resolution of the encoder for position feed back in
increments (quadcounts)
#define ECI52_SSI_CLK_FREQ 2000000 // Clock frequency of the SSI encoder (Hz)
#define ECI52_FAST_UPDATE 1 // fastUpdate @b 1: The encoder is updated with
the current controller update rate (24 kHz on MiniMACS6)
#define ECI52_DATLEN 12 // Databit length of endat position
#define ECI52_IS_BINARY 0 // 1 if data coding is binary otherwise it is gray
coded.
#define ECI52_I_POS 0 // Define an indexposition
#define ECI52_BRUSHLESS 1 // Mode of alignment: For brushless motors
#define ECI52_ALIGN_CUR 3000 // Current for the alignment function in mA
#define ECI52_ALIGN_POS 463 // Offset of the encoder position to the electrical
position
#define ECI52_CONTROLMODE HWAMP_MODE_POS_CUR // Define control typ
#define ECI52_POLEPAIRS 8 // Number of pole pairs
#define ECI52_ELPOL -1 // Always -1 with a maxon motor
#define ECI52_CONTCUR 9360 // Nomial continious current allowed in mA
#define ECI52_MAXCUR ECI52_CONTCUR+1.25 // Maximal current allowed in mA
#define ECI52_THERMAL_TIME 12800 // Thermal time constant of the winding
#define ECI52_MAX_RPM 2000 // Maximum velocity in RPM
#define ECI52_CURKPROP 1100 // Proportional factor of current controller
#define ECI52_CURKINT 150 // Integral factor of current controller
#define ECI52_CURKILIM 15000 // Integral limit of current controller
// not used in this example + HWAMP_MODE_POS_CUR
#define ECI52_VELKPROP 0 // Proportional factor of velocity controller
#define ECI52_VELKINT 0 // Integral factor of velocity controller
#define ECI52_VELKILIM 0 // Integral limit of velocity controller
#define ECI52_VELRES 100 // Velocity resolution, Scaling used for the velocity
and acceleration/deceleration commands
#define ECI52_RAMPTYPE RAMPTYPE_JERKLIMITED // Defines the ramptype
#define ECI52_RAMPMIN 1000 // Maximum acceleration
#define ECI52_JERKMIN 100 // Minimum time (ms) required before reaching the
maximum acceleration
#define ECI52_POSERR 2000 // Minimum time (ms) required before reaching the
maximum acceleration
#define ECI52_DIRECTION 1 // User units have normal orientation. Increasing
encoder values result in increasing user positions.
#define ECI52_KPROP 100 // Proportional value for PID position control loop
#define ECI52_KINT 0 // Integral value for PID position control loop
#define ECI52_KDER 300 // Derivative value for PID position control loop
#define ECI52_KILIM 1000 // Limit value for the integral sum of the PID
position control loop
#define ECI52_KILIMTIME 0 // Time used to increase or decrease the integral
limit
#define ECI52_BANDWIDTH 1000 // Bandwidth within which the PID filter is active.
1000 equals to 100% velocity setpoint
#define ECI52_FFVEL 0 // Velocity Feed forward
#define ECI52_KFFAC 0 // Acceleration Feed forward
#define ECI52_KFFDEC 0 // Deceleration Feed Forward
#define ECI52_POSENCREV 1 // Number of revolutions of the motor
#define ECI52_POSENCQC ECI52_ENCRESP // Number of quadcounts in POSENCREV revolutions
#define ECI52_POSFACT_Z 1 // Number of revolutions of the input shaft
#define ECI52_POSFACT_N 1 // Number of revolutions of the output shaft in
POSFACZ_Z revolutions of the input shaft
#define ECI52_FEEDREV 1 // Number of revolutions of the gear box output shaft
#define ECI52_FEEDDIST 36000 // Distance travelled (in user units) in FEEDREV
revolutions of the gear box output shaft
// Function delkratation
long setupECi52_SC_SSI(long axisNo, long encPort, long posOffset);
long main(void)
{
    ErrorClear();
    DefOrigin(AXIS1_NO);
    // Setup axis & amplifier
    setupECi52_SC_SSI(AXIS1_NO,AXIS1_ENCPORT,ECI52_ALIGN_POS);

```

Example Documentation

```
// Activate the axis
AxisControl(AXIS1_NO, ON);
// Start an endless movement with continuous move control with 50% velocity and 50% acceleration
sdkStartContinuousMove(AXIS1_NO, 50, 50);
while(1)
{
    // Print axis information all 500 ms
    sdkInfoPrintAxesPos();
    Delay(500);
}
return(0);
}
long setupECi52_SC_SSI(long axisNo, long encPort, long posOffset)
{
    long checkMotorAlignment;
    // Amplifier setup
    sdkSetupAmpHallPmsmMotor(
        axisNo,
        ECI52_CONTROLMODE,
        ECI52_POLEPAIRS,
        ECI52_MAXCUR,
        ECI52_ENCRESP,
        ECI52_MAX_RPM,
        -1
    );

    // Encoder setup
    sdkSetupAbsSSIEncoder(
        axisNo,
        encPort,
        ECI52_ENCRESP,
        ECI52_SSI_CLK_FREQ,
        ECI52_FAST_UPDATE,
        ECI52_DATLEN,
        ECI52_IS_BINARY,
        ECI52_I_POS
    );

    // Current control setup
    sdkSetupCurrentPIControl(
        axisNo,
        ECI52_CURKPROP,
        ECI52_CURKINT,
        ECI52_CURKILIM
    );

    // Velocity control setup
    sdkSetupVelocityPIControl(
        axisNo,
        ECI52_VELKPROP,
        ECI52_VELKINT,
        ECI52_VELKILIM
    );

    // Movement parameters for the axis
    sdkSetupAxisMovementParam(
        axisNo,
        ECI52_VELRES,
        ECI52_MAX_RPM,
        ECI52_RAMPTYPE,
        ECI52_RAMPMIN,
        ECI52_JERKMIN,
        ECI52_POSEERR
    );

    // Set the direction of the axis
    sdkSetupAxisDirection(
        axisNo,
        ECI52_DIRECTION);

    // Position control setup
    sdkSetupPositionPIDControlExt(
        axisNo,
        ECI52_KPROP,
        ECI52_KINT,
        ECI52_KDER,
        ECI52_KILIM,
        ECI52_KILIMTIME,
        ECI52_BANDWIDTH,
        ECI52_FFVEL,
        ECI52_KFFAC,
        ECI52_KFFDEC
    );
}
```

```
// Definition of the user units
sdkSetupAxisUserUnits(    axisNo,
                          ECI52_POSENCREV,
                          ECI52_POSENCQC,
                          ECI52_POSFACT_Z,
                          ECI52_POSFACT_N,
                          ECI52_FEEDREV,
                          ECI52_FEEDDIST
                          );

// Motor Alignment
if(posOffset==0)
{
    print("The first alignment is open loop and is used in commissioning to detect the offset of the
encoder position to the electrical position.");
    print("");
    checkMotorAlignment =    sdkMotorAlignment(
                            axisNo,
                            ECI52_BRUSHLESS,
                            ECI52_MAXCUR,
                            ECI52_ALIGN_CUR);

    if(checkMotorAlignment<0)
    {
        print("Sorry alignment didn't work we exit");
        Exit(0);
    }
    else
    {
        print("");
        print("Alignment position offset: ", HWAMP_PARAM(axisNo, HWAMP_POSEL_OFFSET));
        print("The determined alignment offset can now be entered.");
        print("The value must be specified in the macro: ");
        print("#define ECI52_ALIGN_POS ", HWAMP_PARAM(axisNo, HWAMP_POSEL_OFFSET));
        Exit(0);
    }
}
else
{
    // Motor Alignment
    sdkSetMotorAlignmentOffset( axisNo,
                               -1,
                               posOffset
                               );
}

// Setup virtual I2T
sdkSetupVirtualI2T(axisNo, ECI52_CONTCUR, ECI52_THERMAL_TIME);
return(1);
}
```

6.30 Maxon_RE_40_1ax_Inc.mc

```
#include <SysDef.mh>
// Relative path to the main folder ApossC_SDK_Vxx.xx
#include "..\..\..\..\SDK\SDK_ApossC.mc"
// Setting the axis number.
#define AXIS1_NO          0    // Axis module number
#define AXIS1_ENCPORT     0    // Encoder port number. Usually, module instance 0 is connected to X1
                             and so on. Please refer to product manual
// Axis settings
#define RE_40_ENCRESES    4*500    // Resolution of the encoder for position feed back in
                                   increments (quadcounts)
#define RE_40_ENC_LATCHTYPE 0    // Defines the latch type: Z-Signal
#define RE_40_ENC_LATCHPARAM 0    //
#define RE_40_ENC_LATCHSLOPE HWLATCH_SLOPE_RISING // Defines the slope of the trigger signal (Default 1)
#define RE_40_CONTROLMODE  HWAMP_MODE_POS_CUR    // Define control typ
#define RE_40_POLEPAIRS    1    // Number of pole pairs
#define RE_40_CONTCUR      3200    // Nomial continious current allowed in mA
#define RE_40_MAXCUR       RE_40_CONTCUR*1.25    // Maximal current allowed in mA
#define RE_40_THERMAL_TIME 41500    // Thermal time constant of the winding
```

Example Documentation

```
#define RE_40_MAX_RPM      3500          // Maximum velocity in RPM
#define RE_40_CURKPROP    1000         // Proportional factor of current controller
#define RE_40_CURKINT     100          // Integral factor of current controller
#define RE_40_CURKILIM    32767       // Integral limit of current controller
// not used in this example → HWAMP_MODE_POS_CUR
#define RE_40_VELKPROP    0            // Proportional factor of velocity controller
#define RE_40_VELKINT     1            // Integral factor of velocity controller
#define RE_40_VELKILIM    0           // Integral limit of velocity controller
#define RE_40_VELRES      100         // Velocity resolution, Scaling used for the velocity
    and acceleration/deceleration commands
#define RE_40_RAMPTYPE    RAMPTYPE_JERKLIMITED // Defines the ramptype
#define RE_40_RAMPMIN     100         // Maximum acceleration
#define RE_40_JERKMIN     100         // Minimum time (ms) required before reaching the
    maximum acceleration
#define RE_40_POSERR      2000        // Minimum time (ms) required before reaching the
    maximum acceleration
#define RE_40_DIRECTION   1           // User units have normal orientation. Increasing
    encoder values result in increasing user positions.
#define RE_40_KPROP       500         // Proportional value for PID position control loop
#define RE_40_KINT        0           // Integral value for PID position control loop
#define RE_40_KDER        1000        // Derivative value for PID position control loop
#define RE_40_KILIM       1000        // Limit value for the integral sum of the PID
    position control loop
#define RE_40_KILIMTIME   0           // Time used to increase or decrease the integral
    limit
#define RE_40_BANDWIDTH   1000        // Bandwidth within which the PID filter is active.
    1000 equals to 100% velocity setpoint
#define RE_40_FFVEL       0           // Velocity Feed forward
#define RE_40_KFFAC       0           // Acceleration Feed forward
#define RE_40_KFFDEC      0           // Deceleration Feed Forward
#define RE_40_POSENCREV   1           // Number of revolutions of the motor
#define RE_40_POSENCQC    RE_40_ENCRES // Number of quadcounts in POSENCREV revolutions
#define RE_40_POSFACT_Z   1           // Number of revolutions of the input shaft
#define RE_40_POSFACT_N   1           // Number of revolutions of the output shaft in
    POSFACT_Z revolutions of the input shaft
#define RE_40_FEEDREV     1           // Number of revolutions of the gear box output shaft
#define RE_40_FEEDDIST    36000       // Distance travelled (in user units) in FEEDREV
    revolutions of the gear box output shaft
// Function delkratation
long setup_RE_40_Inc(long axisNo, long encPort);
long main(void)
{
    ErrorClear();
    DefOrigin(AXIS1_NO);
    // Setup axis & amplifier
    setup_RE_40_Inc(AXIS1_NO, AXIS1_ENCPORT);
    // Activate the axis
    AxisControl(AXIS1_NO, ON);
    // Start an endless movement with continuous move control with 50% velocity and 50% acceleration
    sdkStartContinuousMove(AXIS1_NO, 50, 50);
    while(1)
    {
        // Print axis information all 500 ms
        sdkInfoPrintAxesPos();
        Delay(500);
    }
    return(0);
}
long setup_RE_40_Inc(long axisNo, long encPort)
{
    // Encoder setup
    sdkSetupIncEncoder(
        axisNo,
        encPort,
        RE_40_ENCRES,
        RE_40_ENC_LATCHTYPE,
        RE_40_ENC_LATCHPARAM,
        RE_40_ENC_LATCHSLOPE
    );

    // Amplifier setup
    sdkSetupAmpDcMotor(
        axisNo,
```

```

RE_40_CONTROLMODE,
RE_40_POLEPAIRS,
RE_40_MAXCUR,
RE_40_ENCRESP,
RE_40_MAX_RPM);

// Current control setup
sdkSetupCurrentPIControl( axisNo,
RE_40_CURKPROP,
RE_40_CURKINT,
RE_40_CURKILIM
);

// Velocity control setup
sdkSetupVelocityPIControl( axisNo,
RE_40_VELKPROP,
RE_40_VELKINT,
RE_40_VELKILIM
);

// Movement parameters for the axis
sdkSetupAxisMovementParam( axisNo,
RE_40_VELRES,
RE_40_MAX_RPM,
RE_40_RAMPTYPE,
RE_40_RAMPMIN,
RE_40_JERKMIN,
RE_40_POSERR
);

// Set the direction of the axis
sdkSetupAxisDirection( axisNo,
RE_40_DIRECTION);

// Position control setup
sdkSetupPositionPIDControlExt( axisNo,
RE_40_KPROP,
RE_40_KINT,
RE_40_KDER,
RE_40_KILIM,
RE_40_KILIMTIME,
RE_40_BANDWIDTH,
RE_40_FFVEL,
RE_40_KFFAC,
RE_40_KFFDEC
);

// Definition of the user units
sdkSetupAxisUserUnits( axisNo,
RE_40_POSENCREV,
RE_40_POSENCQC,
RE_40_POSFACT_Z,
RE_40_POSFACT_N,
RE_40_FEEDREV,
RE_40_FEEDDIST
);

// Setup virtual I2T
sdkSetupVirtualI2T(axisNo, RE_40_CONTCUR, RE_40_THERMAL_TIME);
return(1);
}

```

6.31 Maxon_RE_40_1ax_OL.mc

```

#include <SysDef.mh>
// Relative path to the main folder ApossC_SDK_Vxx.xx
#include "..\..\..\..\SDK\SDK_ApossC.mc"
// Setting the axis number.
#define AXIS1_NO 0 // Axis module number
#define AXIS1_ENCPORT 0 // Encoder port number. Usually, module instance 0 is connected to X1
and so on. Please refer to product manual
// Axis settings
#define RE_40_CONTROLMODE HWAMP_MODE_POS_CUR // Define control typ
#define RE_40_POLEPAIRS 1 // Number of pole pairs
#define RE_40_CONTCUR 3200 // Nomial continious current allowed in mA

```

Example Documentation

```
#define RE_40_MAXCUR          RE_40_CONTCUR*1.25          // Maximal current allowed in mA
#define RE_40_THERMAL_TIME    41500                      // Thermal time constant of the winding
#define RE_40_MAX_RPM         3500                       // Maximum velocity in RPM
#define RE_40_CURKPROP        1000                      // Proportional factor of current controller
#define RE_40_CURKINT         100                       // Integral factor of current controller
#define RE_40_CURKILIM        32767                     // Integral limit of current controller
// not used in this example → HWAMP_MODE_POS_CUR
#define RE_40_VELKPROP        0                         // Proportional factor of velocity controller
#define RE_40_VELKINT         1                         // Integral factor of velocity controller
#define RE_40_VELKILIM        0                         // Integral limit of velocity controller
#define RE_40_VELRES          100                       // Velocity resolution, Scaling used for the velocity
and acceleration/deceleration commands
#define RE_40_RAMPTYPE        RAMPTYPE_JERKLIMITED       // Defines the ramptype
#define RE_40_RAMPMIN         100                       // Maximum acceleration
#define RE_40_JERKMIN         100                       // Minimum time (ms) required before reaching the
maximum acceleration
#define RE_40_POSERR          0                         // Disable following error
#define RE_40_DIRECTION       1                         // User units have normal orientation. Increasing
encoder values result in increasing user positions.
#define RE_40_KPROP           500                       // Proportional value for PID position control loop
#define RE_40_KINT            0                         // Integral value for PID position control loop
#define RE_40_KDER            1000                      // Derivative value for PID position control loop
#define RE_40_KILIM           1000                     // Limit value for the integral sum of the PID
position control loop
#define RE_40_KILIMTIME       0                         // Time used to increase or decrease the integral
limit
#define RE_40_BANDWIDTH       1000                      // Bandwidth within which the PID filter is active.
1000 equals to 100% velocity setpoint
#define RE_40_FFVEL           0                         // Velocity Feed forward
#define RE_40_KFFAC           0                         // Acceleration Feed forward
#define RE_40_KFFDEC          0                         // Deceleration Feed Forward
// Function delkratation
long setup_RE_40_OL(long axisNo, long encPort);
// Command current in mA, adaption while running possible (watch window)
long current = 1000;
long main(void)
{
    ErrorClear();
    DefOrigin(AXIS1_NO);
    // Setup axis & amplifier
    setup_RE_40_OL(AXIS1_NO, AXIS1_ENCPORT);
    // Activate the axis
    AxisControl(AXIS1_NO, USERREFCUR);
    while(1)
    {
        // Set current in mA. If the current is to be regulated to 0, sufficient torque must be applied to
        the motor shaft.
        // Alternatively, the axis can also be switched off with AxisControl(AXIS1_NO, OFF);
        AXE_PROCESS(AXIS1_NO, REG_USERREFCUR)=current;
    }
    return(0);
}
long setup_RE_40_OL(long axisNo, long encPort)
{
    // Disable following error and set compos to actpos
    VIRTCONTIN_PARAM(axisNo, VIRTCONTIN_PISRC_COUNTER) = AXE_PROCESS_SRCINDEX(axisNo, REG_COMPOS);
    // Amplifier setup
    sdkSetupAmpDcMotor(
        axisNo,
        RE_40_CONTROLMODE,
        RE_40_POLEPAIRS,
        RE_40_MAXCUR,
        2000, //Default value
        RE_40_MAX_RPM);

    // Current control setup
    sdkSetupCurrentPIControl(
        axisNo,
        RE_40_CURKPROP,
        RE_40_CURKINT,
        RE_40_CURKILIM
    );

    // Velocity control setup
```

```

sdkSetupVelocityPIControl(      axisNo,
                                RE_40_VELKPROP,
                                RE_40_VELKINT,
                                RE_40_VELKILIM
                                );

// Movement parameters for the axis
sdkSetupAxisMovementParam( axisNo,
                             RE_40_VELRES,
                             RE_40_MAX_RPM,
                             RE_40_RAMPTYPE,
                             RE_40_RAMPMIN,
                             RE_40_JERKMIN,
                             RE_40_POSERR
                             );

// Set the direction of the axis
sdkSetupAxisDirection(      axisNo,
                             RE_40_DIRECTION);

// Position control setup
sdkSetupPositionPIDControlExt(      axisNo,
                                     RE_40_KPROP,
                                     RE_40_KINT,
                                     RE_40_KDER,
                                     RE_40_KILIM,
                                     RE_40_KILIMTIME,
                                     RE_40_BANDWIDTH,
                                     RE_40_FFVEL,
                                     RE_40_KFFAC,
                                     RE_40_KFFDEC
                                     );

// Setup virtual I2T
sdkSetupVirtualI2T(axisNo, RE_40_CONTCUR, RE_40_THERMAL_TIME);
return(1);
}

```

6.32 MotorCommissioning_MaxonECi30.mc

```

#include <SysDef.mh>
#include "..\..\..\SDK\SDK_ApossC.mc"
// Axis module number
#define C_AXIS1 0 // Axis module number
#define C_AXIS1_ENCPORT 0 // Encoder port number
// Definition of the methods
#define C_METHODE_ENCODER_CHECK 0
#define C_METHODE_HALL_CHECK 1
#define C_METHODE_CURRENT_STEP 2
#define C_METHODE_VELOCITY_STEP 3
#define C_METHODE_POSITION_RAMP 4
// Set methode and enable/ disable recording
#define C_COMM_METHODE C_METHODE_ENCODER_CHECK
#define C_RECORD_ENABLE 1
// Settings for the current step
#define C_TIME_CURRENT_STEP 50 // [ms]
#define C_CURRENT_STEP 2000 // [mA]
// Settings for the velocity step
#define C_TIME_VELOCITY_STEP 2500 // [ms]
#define C_VELOCITY_STEP 0X2000 // scaled to the maximum speed with -0x4000 to 0x4000
internal units
// Settings for the position ramp
#define C_POSITION_RAMP 10*360 // [uu] → 5 turns
#define C_CONTROLLER_CASCADING HWAMP_MODE_POS_VEL_CUR // SDO Dictionary: HWAMP_MODE
// Axis setup
#define AXIS_VELRES 100 // Velocity resolution, Scaling used for the
velocity and acceleration/deceleration commands
#define AXIS_RAMPTYPE RAMPTYPE_JERKLIMITED // Defines the ramptype
#define AXIS_RAMPMIN 1000 // Maximum acceleration
#define AXIS_JERKMIN 500 // Minimum time (ms) required before reaching the
maximum acceleration
#define AXIS_MAX_RPM 4000 // Maximum velocity in RPM

```

Example Documentation

```
#define AXIS_POSERR          0          // Maximal track/ position error allowed in qc →
    disable for testing
#define AXIS_POSENCREV      1          // Number of revolutions of the motor
#define AXIS_POSENCQC       ECi30_621403_ENCREV // Number of quadcounts in POSENCREV revolutions
#define AXIS_POSFACT_Z      1          // Number of revolutions of the input shaft
#define AXIS_POSFACT_N      1          // Number of revolutions of the output shaft in
    POSFACT_Z revolutions of the input shaft
#define AXIS_FEEDREV        1          // Number of revolutions of the gear box output
    shaft
#define AXIS_FEEDDIST       360         // Distance travelled (in user units) in FEEDREV
    revolutions of the gear box output shaft
// Encoder settings
#define ECi30_621403_ENCREV 4*4096     // Resolution of the encoder for position feed
    back in increments (quadcounts)
#define ECi30_621403_ENC_LATCHTYPE 0    // Defines the latch type: Index Line
#define ECi30_621403_ENC_LATCHPARAM 0   // -
#define ECi30_621403_ENC_LATCHSLOPE   HWLATCH_SLOPE_RISING // Defines the slope of the trigger signal
    (Default 1)
// Motor settings
#define ECi30_621403_CONTROLMODE   HWAMP_MODE_POS_VEL_CUR // Define control typ
#define ECi30_621403_POLEPAIRS     4          // Number of pole pairs
#define ECi30_621403_CONTCUR       2280       // Nomial continious current allowed in mA
#define ECi30_621403_MAXCUR        ECi30_621403_CONTCUR*1.25 // Maximal current allowed in mA
#define ECi30_621403_THERMAL_TIME   27600      // Thermal time constant of the winding
#define ECi30_621403_MAX_RPM       AXIS_MAX_RPM // Maximum velocity in RPM
#define ECi30_621403_DIRECTION     1          // Direction of the drive
// Parameter for the current controller
// The values were calculated with the Current Calculator Tool
#define ECi30_621403_CURKPROP      1500       // Proportional factor of current controller
#define ECi30_621403_CURKINT       150        // Integral factor of current controller
#define ECi30_621403_CURKILIM      32767      // Integral limit of current controller
// Parameter for the velocity controller
#define ECi30_621403_VELKPROP      2000       // Proportional factor of velocity controller
#define ECi30_621403_VELKINT       1000       // Integral factor of velocity controller
#define ECi30_621403_VELKILIM      0          // Integral limit of velocity controller
// Parameter for the position controller
#define ECi30_621403_KP            500        // Proportional value for PID position control
    loop
#define ECi30_621403_KINT          0          // Integral value for PID position control
    loop
#define ECi30_621403_KDER          1000       // Derivative value for PID position
    control loop
#define ECi30_621403_KILIM         1000       // Limit value for the integral sum of the PID
    position control loop
#define ECi30_621403_KILIMTIME     0          // Time used to increase or decrease the
    integral limit
#define ECi30_621403_BANDWIDTH     1000       // Bandwidth within which the PID filter is
    active. 1000 equals to 100% velocity setpoint
#define ECi30_621403_FFVEL         1000       // Velocity Feed forward
#define ECi30_621403_KFFACC        0          // Acceleration Feed forward
#define ECi30_621403_KFFDEC        0          // Deceleration Feed Forward
long setupECi30_621403(long axisNo);
long main(void)
{
    long waitingTime = 5000;
    long direction = 1;
    setupECi30_621403(C_AXIS1);
    DefOrigin(C_AXIS1);
    while(1)
    {
        #if C_COMM_METHODE == C_METHODE_ENCODER_CHECK
            waitingTime=250;
            sdkMotorCommEncoderUserUnitCheck(C_AXIS1);
        #elif C_COMM_METHODE == C_METHODE_HALL_CHECK
            waitingTime=250;
            sdkMotorCommHallSignalCheck(C_AXIS1);
        #elif C_COMM_METHODE == C_METHODE_CURRENT_STEP
            sdkMotorCommCurrentStep( C_AXIS1,
                                    C_CURRENT_STEP,
                                    C_TIME_CURRENT_STEP,
```

```

        C_RECORD_ENABLE);
#elif C_COMM_METHODE == C_METHODE_VELOCITY_STEP
    sdkMotorCommVelStep( C_AXIS1,
                        direction*C_VELOCITY_STEP,
                        C_TIME_VELOCITY_STEP,
                        C_RECORD_ENABLE);

    direction = direction*-1;
#elif C_COMM_METHODE == C_METHODE_POSITION_RAMP
    sdkMotorCommPositionRamp( C_AXIS1,
                            direction*C_POSITION_RAMP,
                            C_CONTROLLER_CASCADING,
                            C_RECORD_ENABLE);

    direction = direction*-1;
#endif

    if(waitingTime>1000)
        print("Wait for ", waitingTime, " ms");
    Delay(waitingTime);
}
return(0);
}
// Axis, motor and encoder setup
long setupECi30_621403(long axisNo)
{
    print("ECi30 MiniMACS6");
    // Encoder setup
    sdkSetupIncEncoder(
        axisNo,
        axisNo,
        ECi30_621403_ENCRESP,
        ECi30_621403_ENC_LATCHTYPE,
        ECi30_621403_ENC_LATCHPARAM,
        ECi30_621403_ENC_LATCHSLOPE
    );

    // Amplifier setup
    #if 1
        sdkSetupAmpHallPmsmMotor(
            axisNo,
            ECi30_621403_CONTROLMODE,
            ECi30_621403_POLEPAIRS,
            ECi30_621403_MAXCUR,
            ECi30_621403_ENCRESP,
            ECi30_621403_MAX_RPM,
            -1
        );
    #else
        sdkSetupAmpBldcMotor(
            axisNo,
            ECi30_621403_CONTROLMODE,
            ECi30_621403_POLEPAIRS,
            ECi30_621403_MAXCUR,
            ECi30_621403_ENCRESP,
            ECi30_621403_MAX_RPM
        );
    #endif

    // Current control setup
    sdkSetupCurrentPIControl(
        axisNo,
        ECi30_621403_CURKPROP,
        ECi30_621403_CURKINT,
        ECi30_621403_CURKILIM
    );

    // Velocity control setup
    sdkSetupVelocityPIControl(
        axisNo,
        ECi30_621403_VELKPROP,
        ECi30_621403_VELKINT,
        ECi30_621403_VELKILIM
    );

    // Set the direction of the axis
    sdkSetupAxisDirection(
        axisNo,
        ECi30_621403_DIRECTION);

    // Position control setup
    sdkSetupPositionPIDControlExt(
        axisNo,
        ECi30_621403_KPROP,
        ECi30_621403_KINT,

```

```

        ECi30_621403_KDER,
        ECi30_621403_KILIM,
        ECi30_621403_KILIMTIME,
        ECi30_621403_BANDWIDTH,
        ECi30_621403_FFVEL,
        ECi30_621403_KFFACC,
        ECi30_621403_KFFDEC);

// Set the virtual I2T
sdkSetupVirtualI2T(axisNo, ECi30_621403_CONTCUR, ECi30_621403_THERMAL_TIME);
// Movement parameters for the axis
sdkSetupAxisMovementParam( axisNo,
        AXIS_VELRES,
        AXIS_MAX_RPM,
        AXIS_RAMPTYPE,
        AXIS_RAMPMIN,
        AXIS_JERKMIN,
        AXIS_POSERR
    );

// Definition of the user units
sdkSetupAxisUserUnits(    axisNo,
        AXIS_POSENCREV,
        AXIS_POSENCQC,
        AXIS_POSFACT_Z,
        AXIS_POSFACT_N,
        AXIS_FEEDREV,
        AXIS_FEEDDIST
    );

return(1);
}

```

6.33 PWM_1Ax_ESCON.mc

```

#include "..\..\..\SDK\SDK_ApossC.mc"
// Application settings to change the mode
#define PWM_AXIS_MODE    0
#define PWM_USER_MODE    1
#define PWM_MODE    PWM_AXIS_MODE
// #define PWM_MODE    PWM_USER_MODE
// ESCON control settings
#define ESCON_AXIS_NO        0            // Axis number (zero based) → PWM_AXIS_MODE
#define ESCON_PWM_USER_PARAM    1        // User parameter number → PWM_USER_MODE
#define ESCON_PWM_DIGOUTPUT_NO    0        // Digital output for PWM cmd value (zero based)
#define ESCON_ENABLE_DIGOUTPUT_NO    1    // Digital output for ESCON enable (zero based)
#define ESCON_PWM_FREQUENCY    50        // PWM frequency [Hz]
#define ESCON_PWM_CYCLE_RANGE    80        // PWM cycle range 10-90%
#define ESCON_PWM_POLARITY    HWPWMGEN_POLARITY_POSITIVE
// Analog feedback defines
#define ESCON_AI_FEEDBACK_NO    0            // Analog input number (zero based)
#define ESCON_AI_SPEED_MIN_VOLTAGE    0    // 0 voltage → -5000 rpm
#define ESCON_AI_SPEED_OFF_VOLTAGE    2000 // offset: 2 voltage → 0 rpm
#define ESCON_AI_SPEED_MAX_VOLTAGE    4000 // 4 voltage → 5000 rpm
#define ESCON_AI_SPEED_MIN_RPM    -5000    // 0 voltage → -5000 rpm
#define ESCON_AI_SPEED_MAX_RPM    5000     // 4 voltage → 5000 rpm
// Set velocity for pwm axis mode (0 → +/- 100)
long cvel    =    0;
// Set duty cycle for pwm user parameter mode (10 → 90%)
long dutyCycle    = 0x7FFF; // 50%
long main(void)
{
    if PWM_MODE == PWM_USER_MODE
        sdkSetupPwmGenerator_UserParamMode(    ESCON_PWM_USER_PARAM,
                                                ESCON_PWM_DIGOUTPUT_NO,
                                                ESCON_PWM_FREQUENCY,
                                                ESCON_PWM_CYCLE_RANGE,
                                                ESCON_PWM_POLARITY);
    else
        sdkSetupPwmGenerator_AxisModeVel(    ESCON_AXIS_NO,
                                                ESCON_ENABLE_DIGOUTPUT_NO,

```

```

        ESCON_PWM_DIGOUTPUT_NO,
        ESCON_PWM_FREQUENCY,
        ESCON_PWM_CYCLE_RANGE,
        ESCON_PWM_POLARITY);
#endifif
    sdkScaleAnalogInput( ESCON_AI_FEEDBACK_NO,
        ESCON_AI_SPEED_MIN_VOLTAGE,
        ESCON_AI_SPEED_MAX_VOLTAGE,
        ESCON_AI_SPEED_OFF_VOLTAGE,
        ESCON_AI_SPEED_MIN_RPM,
        ESCON_AI_SPEED_MAX_RPM);
#if PWM_MODE == PWM_USER_MODE
    // Automatic disable if there is an error
    GLB_PARAM(ESCONDGLB)=GLB_PARAM_ESCONDGLB_OUT_OFF;
    // Reset enable output if there was an error
    DigOutput(ESCON_ENABLE_DIGOUTPUT_NO+1,0);
    Delay(50);
    // Enable ESCON Controller
    DigOutput(ESCON_ENABLE_DIGOUTPUT_NO+1,1);
#else
    // Enable axis
    AxisControl(ESCON_AXIS_NO, ON);
    // Set profil
    Acc(ESCON_AXIS_NO,100);
    Dec(ESCON_AXIS_NO,100);
    // Set constant velocity and start axis
    Cvel(ESCON_AXIS_NO,cvel);
    AxisCvelStart(ESCON_AXIS_NO);
#endifif
    while(1)
    {
        #if PWM_MODE == PWM_USER_MODE
            // reset dutycycle
            USER_PARAM(ESCON_PWM_USER_PARAM)=dutyCycle;
            print("HWPWMGEN_MODE_UNSIGNED, PWM DuryCycle:
",HWPWMGEN_PROCESS(ESCON_PWM_DIGOUTPUT_NO,PO_HWPWMGEN_VALUE));
        #else
            // Set constant velocity and restart axis
            Cvel(ESCON_AXIS_NO,cvel);
            AxisCvelStart(ESCON_AXIS_NO);
            print("HWPWMGEN_MODE_SIGNED, PWM DuryCycle:
",HWPWMGEN_PROCESS(ESCON_PWM_DIGOUTPUT_NO,PO_HWPWMGEN_VALUE));
        #endifif
        print("ESCON Feedback: ",AnalogInput(ESCON_AI_FEEDBACK_NO + 1)," Rpm");
        print("");
        Delay(2000);
    }
}

```

6.34 ScaraRobot_SM.mc

```

#include <SysDef.mh>
//Define encoder simulation or real motors on a MiniMACS6 → The file "Motors\Maxon_XYZ.mc" must be
    adapted before start
#define MINIMACS6_MOTORS      1
#define SIMULATION            2
#define MACHINE                SIMULATION
#include "include\UserInterface.mh"
#include "include\MachineSpecification.mh"
#include "include\Global_Variables.mh"
#include "include\General_Functions.mc"
#include "include\AxisSetup.mc"
/*
** Load the path arrays from the ".zbc" configuration file.
*/
#config "include\scara_Show.zbc", CONFIGFLG_DIM
#include "include\StateMachine_Main.mc"
long main(void)

```

Example Documentation

```
{
    /*
    ** Start by clearing any previous errors.
    */
    ErrorClear();
    /*
    ** Initialize the user parameters.      These are used to communicate
    ** with the Monitor dialog.
    */
    UserParamInit();
    /*
    ** Start the state machine.
    */
    print("Starting...");
    SmRun(Scara);
    print("Done.");
    return(0);
} // $X {Actual Pos.,1,1,0,-1,0,-1,0,(-1),-1},0x6064,0,0
// $X {Actual Pos.,1,1,0,-1,0,-1,0,(-1),-1},0x6864,0,0
// $X {Actual Pos.,1,1,0,-1,0,-1,0,(-1),-1},0x7064,0,0
```

6.35 SDK_Amplifier_DS402_StateMachine.mc

6.36 SetupAbsSSIEncoder.mc

```
#include <SysDef.mh>
#include "..\..\SDK\SDK_ApossC.mc"
#define AXIS 0 // Axis module number
#define AXIS_ENCPORT 0 // Encoder port number. Usually, module instance
0 is connected to X1 and so on. Please refer to product manual
#define AXIS_ENCRECRES 4096 // Resolution of the encoder for position feed
back in increments (quadcounts)
#define AXIS_ENC_CLOCK_FREQ 2000000 // Depending on the requirements of the sensor
#define AXIS_ENC_DATALENGTH 13 // Databit length of endat position
#define AXIS_ENC_FAST_UPDATE 0 // 1 if SSI is used for commutation
#define AXIS_ENC_CODED 0 // 1 if data coding is binary otherwise it is gray
coded.
#define AXIS_ENC_LATCH_BITMASK 1 // Bitmask to check the latchvalue
long main(void)
{
    sdkSetupAbsSSIEncoder( AXIS,
        AXIS_ENCPORT,
        AXIS_ENCRECRES,
        AXIS_ENC_CLOCK_FREQ,
        AXIS_ENC_FAST_UPDATE,
        AXIS_ENC_DATALENGTH,
        AXIS_ENC_CODED,
        AXIS_ENC_LATCH_BITMASK
    );

    while(1)
    {
        sdkInfoPrintAxesPos();
        Delay(500);
    }
    return(0);
}
```

6.37 SetupIncEncoder.mc

```
#include <SysDef.mh>
#include "..\..\SDK\SDK_ApossC.mc"
#define AXIS 0 // Axis module number
#define AXIS_ENCPORT 0 // Encoder port number. Usually, module instance
0 is connected to X1 and so on. Please refer to product manual
```

```

#define AXIS_ENCRES          1024          // Resolution of the encoder for position feed
    back in increments (quadcounts)
#define AXIS_ENC_LATCHTYPE    1            // Defines the latch type: Digital input
#define AXIS_ENC_LATCHPARAM    1          // Latch on digital input 1
#define AXIS_ENC_LATCHSLOPE    HWLATCH_SLOPE_RISING // Defines the slope of the trigger signal
    (Default 1)
long main(void)
{
    sdkSetupIncEncoder(    AXIS,
                          AXIS_ENCPORT,
                          AXIS_ENCRES,
                          AXIS_ENC_LATCHTYPE,
                          AXIS_ENC_LATCHPARAM,
                          AXIS_ENC_LATCHSLOPE
                          );

    while(1)
    {
        sdkInfoPrintAxesPos();
        Delay(2000);
    }
    return(0);
}

```

6.38 SetupSignalGeneratorAxis.mc

```

#include <SysDef.mh>
#include "..\..\SDK\SDK_ApossC.mc"
#include "..\..\SDK\Encoder\SDK_SignalGenerator.mc"
#define AXIS          0          // Axis module number
#define ENCPORNTX      0          // Encoder port number used for signal generator
#define ENCPORTRX      1          // Encoder port number used for reception of the signal
#define INDEX_DISTANCE 10000
long startRecording(long axis);
long main(void)
{
    sdkSetupIncEncoder(AXIS,
                      ENCPORTRX,
                      2000,
                      1,          //digital input latch type
                      1,          //digital input no
                      HWLATCH_SLOPE_RISING);
    sdkSigGenAxisSetupMiniMACS6(AXIS,
                                ENCPORNTX,
                                INDEX_DISTANCE
                                );

    Delay(1);
    DefOrigin(AXIS);
    startRecording(AXIS);
    AxisControl(AXIS, ON);
    AxisPosAbsStart(AXIS, 100000);
    AxisWaitReached(AXIS);
    Delay(100);
    RecordStop(0, 0);
    while(1)
    {
        sdkInfoPrintAxesPos();
        Delay(500);
    }
    return(0);
}
long startRecording(long axis)
{
    //set up oscilloscope
    RecordTime(1);
    RecordDest(DYNMEM);
    RecordIndex(AXE_PROCESS_INDEX(axis, REG_ACTPOS),
               AXE_PROCESS_INDEX(axis, REG_COMPOS),
               AXE_PROCESS_INDEX(axis, REG_ZERO),

```

Example Documentation

```

        AXE_PROCESS_INDEX(axis, REG_IAVEL),
        VIRTMAST_PROCESS_INDEX(axis, PO_VIRTMAST_VEL));
RecordType(0);
RecordStart(0);
}

```

6.39 SetupSignalGeneratorOpenloop.mc

```

#include <SysDef.mh>
#include "..\..\SDK\SDK_ApossC.mc"
#include "..\..\SDK\Encoder\SDK_SignalGenerator.mc"
#define AXIS 0 // Axis module number
#define ENCPOR_TX 0 // Encoder port number used for signal generator
#define ENCPOR_RX 1 // Encoder port number used for reception of the signal
#define INDEX_DISTANCE 10000
long startRecording(long axis);
long main(void)
{
    sdkSetupIncEncoder(AXIS,
                      ENCPOR_RX,
                      2000,
                      1, //digital input latch type
                      1, //digital input no
                      HWLATCH_SLOPE_RISING);
    sdkSigGenOpenloopSetupMiniMACS6(ENCPOR_TX,
                                     INDEX_DISTANCE
                                     );
    sdkSigGenOpenloopAcceleration(ENCPOR_TX, 100000);
    sdkSigGenOpenloopDeceleration(ENCPOR_TX, 50000);
    Delay(1);
    DefOrigin(AXIS);
    startRecording(AXIS);
    //start master and ramp up
    sdkSigGenOpenloopVelocity(ENCPOR_TX, 100000);
    Delay(2500);
    sdkSigGenEnable(ENCPOR_TX, 0);
    Delay(100);
    sdkSigGenEnable(ENCPOR_TX, 1);
    //ramp down to 0
    sdkSigGenOpenloopVelocity(ENCPOR_TX, 0);
    Delay(3000);
    //reset index generator
    sdkSigGenReset(ENCPOR_TX);
    //start master
    sdkSigGenOpenloopVelocity(ENCPOR_TX, -100000);
    Delay(2500);
    sdkSigGenEnable(ENCPOR_TX, 0);
    Delay(100);
    sdkSigGenEnable(ENCPOR_TX, 1);
    //stop master
    sdkSigGenOpenloopVelocity(ENCPOR_TX, 0);
    Delay(2500);
    sdkSigGenEnable(ENCPOR_TX, 0);
    Delay(100);
    RecordStop(0, 0);
    while(1)
    {
        sdkInfoPrintAxesPos();
        Delay(500);
    }
    return(0);
}
long startRecording(long axis)
{
    //set up oscilloscope
    RecordTime(1);
    RecordDest(DYNMEM);
    RecordIndex(AXE_PROCESS_INDEX(axis, REG_ACTPOS),

```

```

    AXE_PROCESS_INDEX(axis, REG_ZERO),
    AXE_PROCESS_INDEX(axis, REG_IAVEL),
    VIRTMAST_PROCESS_INDEX(axis, PO_VIRTMAST_VEL));
RecordType(0);
RecordStart(0);
}

```

6.40 SetupSinCosEncoder.mc

```

#include <SysDef.mh>
#include "..\..\SDK\SDK_ApossC.mc"
#define AXIS 0 // Axis module number
#define AXIS_ENCPORT 3 // Encoder port number. Usually, module instance
0 is connected to X1 and so on. Please refer to product manual
#define AXIS_ENCRESP 5000 // Resolution of the encoder for position feed
back in increments (quadcounts)
#define AXIS_ENC_LATCHTYPE 1 // Defines the latch type: Digital input
#define AXIS_ENC_LATCHPARAM 1 // Latch on digital input 1
#define AXIS_ENC_LATCHSLOPE HWLATCH_SLOPE_RISING // Defines the slope of the trigger signal
(Default 1)
long main(void)
{
    sdkSetupSinCosEncoder( AXIS,
        AXIS_ENCPORT,
        AXIS_ENCRESP*256,
        AXIS_ENC_LATCHTYPE,
        AXIS_ENC_LATCHPARAM,
        AXIS_ENC_LATCHSLOPE
    );

    while(1)
    {
        sdkInfoPrintAxesPos();
        Delay(2000);
    }
    return(0);
}

```

6.41 Stepper_XY_CL.mc

```

#include <SysDef.mh>
// Relative path to the main folder ApossC_SDK_Vxx.xx
#include "..\..\..\..\SDK\SDK_ApossC.mc"
// Setting the axis number.
#define AXIS1_NO 0 // Axis module number
#define AXIS1_ENCPORT 0 // Encoder port number.
// Axis settings
#define STEPPER_CL_STEPS 200 // Resolution of the stepper → 200 Steps per rev
→ 1.8 deg/ step
#define STEPPER_CL_ENCRESP 4000 // Resolution of the encoder for position feed
back in increments (quadcounts)
#define STEPPER_CL_CONTROLMODE HWAMP_MODE_POS_CUR // Define control typ
#define STEPPER_CL_CONTCUR 4000 // Nomial continious current allowed in mA
#define STEPPER_CL_MAXCUR STEPPER_CL_CONTCUR * 1.25 // Maximal current allowed in mA
#define STEPPER_CL_THERMAL_TIME 25600 // Thermal time constant of the winding
#define STEPPER_CL_MAX_RPM 400 // Maximum velocity in RPM
// Portescap P532 258 0.7 10 (parallel): 0.350hm / 0.7mH / Tools → Current Regulator Calculator → if
there is noise the values are to high
#define STEPPER_CL_CURKPROP 20000 // Proportional factor of current controller
#define STEPPER_CL_CURKINT 250 // Integral factor of current controller
#define STEPPER_CL_CURKILIM 32767 // Integral limit of current controller
#define STEPPER_CL_VELRES 100 // Velocity resolution, Scaling used for the
velocity and acceleration/deceleration commands
#define STEPPER_CL_RAMPTYPE RAMPTYPE_JERKLIMITED // Defines the ramptype
#define STEPPER_CL_RAMPMIN 100 // Maximum acceleration
#define STEPPER_CL_JERKMIN 100 // Minimum time (ms) required before reaching
the maximum acceleration

```

Example Documentation

```
#define STEPPER_CL_POSERR      2000          // Maximal track/ position error allowed in qc
#define STEPPER_CL_DIRECTION  1             // User units have normal orientation.
    Increasing encoder values result in increasing user positions.
#define STEPPER_CL_POSENCREV  1             // Number of revolutions of the motor
#define STEPPER_CL_POSENCQC   STEPPER_CL_ENCRES // Number of quadcounts in POSENCREV
    revolutions
#define STEPPER_CL_POSFACT_Z   1             // Number of revolutions of the input shaft
#define STEPPER_CL_POSFACT_N   1             // Number of revolutions of the output shaft
    in POSFACT_Z revolutions of the input shaft
#define STEPPER_CL_FEEDREV    1             // Number of revolutions of the gear box
    output shaft
#define STEPPER_CL_FEEDDIST    36000         // Distance travelled (in user units) in FEEDREV
    revolutions of the gear box output shaft
#define STEPPER_CL_KPROP      60             // Proportional value for PID position control
    loop
#define STEPPER_CL_KINT       0             // Integral value for PID position control
    loop
#define STEPPER_CL_KDER       3000           // Derivative value for PID position
    control loop
#define STEPPER_CL_KILIM      1000          // Limit value for the integral sum of the PID
    position control loop
#define STEPPER_CL_KILIMTIME  0             // Time used to increase or decrease the
    integral limit
#define STEPPER_CL_BANDWIDTH  1000          // Bandwidth within which the PID filter is
    active. 1000 equals to 100% velocity setpoint
#define STEPPER_CL_FFVEL      0             // Velocity Feed forward
#define STEPPER_CL_KFFAC      0             // Acceleration Feed forward
#define STEPPER_CL_KFFDEC     0             // Deceleration Feed Forward
// Function delkration
long setupStepper_CL(long axisNo, long encPort);
long main(void)
{
    ErrorClear();
    DefOrigin(AXIS1_NO);
    // Setup axis & amplifier
    setupStepper_CL(AXIS1_NO,AXIS1_ENCPORT);
    // Activate the axis
    AxisControl(AXIS1_NO, ON);
    // Start an endless movement with continuous move control with 75% velocity and 75% acceleration
    sdkStartContinuousMove(AXIS1_NO, 75, 75);
while(1)
{
    // Print axis information all 500 ms
    sdkInfoPrintAxesPos();
    Delay(500);
}
return(0);
}
long setupStepper_CL(long axisNo, long encPort)
{
    long checkMotorAlignment=0;
    // Amplifier setup
    sdkSetupAmpStepMotor_CL( axisNo,
                            STEPPER_CL_CONTROLMODE,
                            STEPPER_CL_STEPS,
                            STEPPER_CL_MAXCUR,
                            STEPPER_CL_ENCRES,
                            STEPPER_CL_MAX_RPM
                            );
    // At high speeds, the lag value can be determined empirically. Otherwise this line can be commented
    out
    HWAMP_PARAM(axisNo,HWAMP_ELPOS_LAGMULT) = 115;
    // Encoder setup
    sdkSetupIncEncoder(
        axisNo,
        encPort,
        STEPPER_CL_ENCRES,
        0,
        0,
        0);

    // Current control setup
```

```

sdkSetupCurrentPIControl(  axisNo,
                           STEPPER_CL_CURKPROP,
                           STEPPER_CL_CURKINT,
                           STEPPER_CL_CURKILIM
                           );

// Movement parameters for the axis
sdkSetupAxisMovementParam( axisNo,
                           STEPPER_CL_VELRES,
                           STEPPER_CL_MAX_RPM,
                           STEPPER_CL_RAMPTYPE,
                           STEPPER_CL_RAMPMIN,
                           STEPPER_CL_JERKMIN,
                           STEPPER_CL_POSERR
                           );

// Definition of the user units
sdkSetupAxisUserUnits(    axisNo,
                           STEPPER_CL_POSENCREV,
                           STEPPER_CL_POSENCQC,
                           STEPPER_CL_POSFACT_Z,
                           STEPPER_CL_POSFACT_N,
                           STEPPER_CL_FEEDREV,
                           STEPPER_CL_FEEDDIST
                           );

sdkSetupAxisDirection(axisNo, STEPPER_CL_DIRECTION);
// Position control setup
sdkSetupPositionPIDControlExt(  axisNo,
                                STEPPER_CL_KPROP,
                                STEPPER_CL_KINT,
                                STEPPER_CL_KDER,
                                STEPPER_CL_KILIM,
                                STEPPER_CL_KILIMTIME,
                                STEPPER_CL_BANDWIDTH,
                                STEPPER_CL_FFVEL,
                                STEPPER_CL_KFFAC,
                                STEPPER_CL_KFFDEC
                                );

// Motor Alignment
checkMotorAlignment =  sdkMotorAlignment(
                        axisNo,
                        3,
                        STEPPER_CL_MAXCUR,
                        STEPPER_CL_CONTCUR);

if(checkMotorAlignment<0)
{
    print("Sorry alignement didn't work we exit");
    Exit(0);
}
// Setup virtual I2T
sdkSetupVirtualI2T(axisNo,STEPPER_CL_CONTCUR, STEPPER_CL_THERMAL_TIME);
return(1);
}

```

6.42 Stepper_XY_OL.mc

```

#include <SysDef.mh>
// Relative path to the main folder ApossC_SDK_Vxx.xx
#include "..\..\..\..\SDK\SDK_ApossC.mc"
// Setting the axis number.
#define AXIS1_NO          0    // Axis module number
// Axis settings
#define STEPPER_OL_STEPS      200    // Resolution of the stepper → 200 Steps per rev
    → 1.8 deg/ step
#define STEPPER_OL_CONTCUR    4000    // Nomial continious current allowed in mA
#define STEPPER_OL_MAXCUR     STEPPER_OL_CONTCUR * 1.25 // Maximal current allowed in mA
#define STEPPER_OL_THERMAL_TIME 25600 // Thermal time constant of the winding
#define STEPPER_OL_MAX_RPM    400    // Maximum velocity in RPM
// Portescap P532 258 0.7 10 (parallel): 0.350hm / 0.7mH / Tools → Current Regulator Calculator → if
    there is noise the values are to high

```

Example Documentation

```
#define STEPPER_OL_CURKPROP      20000
#define STEPPER_OL_CURKINT      250
#define STEPPER_OL_CURKILIM     32767
#define STEPPER_OL_VELRES      100
    velocity and acceleration/deceleration commands
#define STEPPER_OL_RAMPTYPE      RAMPTYPE_JERKLIMITED
#define STEPPER_OL_RAMPMIN      100
#define STEPPER_OL_JERKMIN      100
    the maximum acceleration
#define STEPPER_OL_POSERR      0
#define STEPPER_OL_DIRECTION    1
#define STEPPER_OL_POSENCREV    1
#define STEPPER_OL_POSFACT_Z    1
#define STEPPER_OL_POSFACT_N    1
    in POSFACT_Z revolutions of the input shaft
#define STEPPER_OL_FEEDREV      1
    output shaft
#define STEPPER_OL_FEEDDIST      36000
    FEEDREV revolutions of the gear box output shaft
#define STEPPER_OL_KPROP        0
    loop
#define STEPPER_OL_KINT         0
    loop
#define STEPPER_OL_KDER         0
    loop
#define STEPPER_OL_KILIM        0
    position control loop
#define STEPPER_OL_KILIMTIME    0
    integral limit
#define STEPPER_OL_BANDWIDTH    0
    active. 1000 equals to 100% velocity setpoint
#define STEPPER_OL_FFVEL        0
#define STEPPER_OL_KFFAC        0
#define STEPPER_OL_KFFDEC       0
// Function delkratation
long setupStepper_OL(long axisNo);
long main(void)
{
    ErrorClear();
    DefOrigin(AXIS1_NO);
    // Setup axis & amplifier
    setupStepper_OL(AXIS1_NO);
    // Activate the axis
    AxisControl(AXIS1_NO, ON);
    // Start an endless movement with continuous move control with 75% velocity and 75% acceleration
    sdkStartContinuousMove(AXIS1_NO, 75, 75);
    while(1)
    {
        // Print axis information all 500 ms
        sdkInfoPrintAxesPos();
        Delay(500);
    }
    return(0);
}
long setupStepper_OL(long axisNo)
{
    // The current position is set equal to the target position as there is no feedback
    sdkSetupAxisSimulation(axisNo);
    // The following error monitoring is switched off
    // If the FW is newer than 9.4.00 the parameter can also be set to 0 to switch off the monitoring
    AXE_PARAM(axisNo, POSERR) = 0x7FFFFFFF;
    // Amplifier setup
    sdkSetupAmpStepMotor_OL( axisNo,
                            STEPPER_OL_STEPS,
                            STEPPER_OL_MAXCUR,
                            STEPPER_OL_MAX_RPM
                            );

    // Current control setup
    sdkSetupCurrentPIControl( axisNo,
                             STEPPER_OL_CURKPROP,
```

```
// Proportional factor of current controller
// Integral factor of current controller
// Integral limit of current controller
// Velocity resolution, Scaling used for the

// Defines the ramptype
// Maximum acceleration
// Minimum time (ms) required before reaching

// Maximal track/ position error allowed in qc
// User units have normal orientation.
// Number of revolutions of the motor
// Number of revolutions of the input shaft
// Number of revolutions of the output shaft

// Number of revolutions of the gear box

// Distance travelled (in user units) in

// Proportional value for PID position control
// Integral value for PID position control
// Derivative value for PID position control
// Limit value for the integral sum of the PID
// Time used to increase or decrease the

// Bandwidth within which the PID filter is

// Velocity Feed forward
// Acceleration Feed forward
// Deceleration Feed Forward
```

```

        STEPPER_OL_CURKINT,
        STEPPER_OL_CURKILIM
    );

// Movement parameters for the axis
sdkSetupAxisMovementParam( axisNo,
    STEPPER_OL_VELRES,
    STEPPER_OL_MAX_RPM,
    STEPPER_OL_RAMPTYPE,
    STEPPER_OL_RAMPMIN,
    STEPPER_OL_JERKMIN,
    STEPPER_OL_POSERR
);

// Definition of the user units
sdkSetupAxisUserUnits(    axisNo,
    STEPPER_OL_POSENCREV,
    HWAMP_PARAM(axisNo, HWAMP_POLES) * HWAMP_ENCRESES_MICROSTEP_RES,
    STEPPER_OL_POSFACT_Z,
    STEPPER_OL_POSFACT_N,
    STEPPER_OL_FEEDREV,
    STEPPER_OL_FEEDDIST
);

sdkSetupAxisDirection(axisNo, STEPPER_OL_DIRECTION);
// Position control setup
sdkSetupPositionPIDControlExt(    axisNo,
    STEPPER_OL_KPROP,
    STEPPER_OL_KINT,
    STEPPER_OL_KDER,
    STEPPER_OL_KILIM,
    STEPPER_OL_KILIMTIME,
    STEPPER_OL_BANDWIDTH,
    STEPPER_OL_FFVEL,
    STEPPER_OL_KFFAC,
    STEPPER_OL_KFFDEC
);

// Setup virtual I2T
sdkSetupVirtualI2T(axisNo, STEPPER_OL_CONTCUR, STEPPER_OL_THERMAL_TIME);
return(1);
}

```

6.43 VirtualMaster_ProfileMode.mc

```

#include <SysDef.mh>
#include "..\..\SDK\SDK_ApossC.mc"
#define VIRTUAL_MASTER 0
#define VIRTUAL_AXIS 0
long main(void)
{
    // Setup an virtual master in profile mode
    sdkSetupVirtualMasterMode(VIRTUAL_MASTER, VIRTMAST_MODE_PROFILE);
    // Setup an virtual axis for simulation
    sdkSetupAxisSimulation(VIRTUAL_AXIS);
    // Set the virtual master VIRTUAL_MASTER as master of VIRTUAL_AXIS
    sdkSetupVirtualMasterAxisLink(VIRTUAL_MASTER, VIRTUAL_AXIS);
    // The "inputs" are converted from increments to quadcounts.
    sdkSetupVirtualMasterScale(VIRTUAL_MASTER, 4, 1);
    // Set the profile of the virtual master
    // Acc/ Dec = 10 qc / ms
    sdkSetVirtualMasterProfile(VIRTUAL_MASTER, 10, 10);
    // The axis must be on and ready for synchronisation
    AxisControl(VIRTUAL_AXIS, ON);
    SyncPos(VIRTUAL_AXIS);
    // Start the virtual master with a velocity of 10
    sdkStartVirtualMasterProfile(VIRTUAL_MASTER, 10);
    while(1)
    {
        sdkInfoPrintAxesPos();
        Delay(2000);
    }
}

```

Example Documentation

```
// Stops the virtual master -> Set velocity to zero
sdkStopVirtualMasterProfile(VIRTUAL_MASTER);
// Disable the virtual master module
sdkSetupVirtualMasterMode(VIRTUAL_MASTER, VIRTMAST_MODE_DISABLED);
return(0);
}
```

