

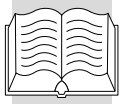
# **BONFIGLIOLI AXIS MANAGER**

Siemens

Installer manual







## TABLE OF CONTENTS

<b>1</b>	<b>General information about the Documentation .....</b>	<b>5</b>
1.1	This document .....	5
1.2	Prerequisites .....	5
1.3	Warranty and liability .....	5
1.4	Copyright .....	6
1.5	Storage .....	6
<b>2</b>	<b>General safety instructions and information on use .....</b>	<b>7</b>
2.1	Warning information and symbols used in the user manual .....	7
2.1.1	Hazard classes .....	7
2.1.2	Hazard symbols .....	7
2.1.3	Prohibition signs .....	7
2.1.4	Personal safety equipment .....	8
2.1.5	Recycling .....	8
2.1.6	Grounding symbol .....	8
2.1.7	ESD symbol .....	8
2.1.8	Information sign .....	8
<b>3</b>	<b>Introduction .....</b>	<b>9</b>
3.1	Information about the product .....	9
<b>4</b>	<b>Library Overview .....</b>	<b>10</b>
4.1	Example Project .....	10
4.2	Overview of the FBs .....	11
4.3	Overview of the Constants .....	12
4.4	Overview of the Structures .....	12
4.5	Naming convention .....	13
<b>5</b>	<b>Overview of Axis Manager .....</b>	<b>14</b>
5.1	State diagram .....	14
5.2	AxisMgr Interface .....	15
5.2.1	State transitions .....	19
5.2.1.1	<i>i_xAbort</i> .....	19
5.2.1.2	<i>From Disabled to Standstill</i> .....	19
5.2.1.3	<i>Launching Commands</i> .....	22
5.2.1.4	<i>ErrorStop: possible causes, q_xFault and i_xReset</i> .....	23
5.2.2	Fieldbus management .....	24
5.2.2.1	<i>PROFINet®</i> .....	25
5.2.2.2	<i>PROFibus®</i> .....	29
5.3	Overview of AxisMgr Functions .....	33
5.3.1	FB_ETrigA .....	35
5.3.1.1	<i>State Machine Description</i> .....	36
5.3.1.2	<i>State transition</i> .....	36
5.3.1.3	<i>Operational Behaviour</i> .....	37
5.3.2	Input/Output data .....	39
5.3.2.1	<i>i_stXXX structure</i> .....	39



5.3.2.2	Values conversion factor, user units definition .....	39
5.3.3	Hardware error (FB_MCHardwareError) .....	41
5.3.4	Homing function (FB_MCHoming).....	43
5.3.5	Power (FB_MCPower) .....	46
5.3.6	Profile Position (FB_MCProfilePosition) .....	48
5.3.7	Profile Torque (FB_MCProfileTorque) .....	52
5.3.8	Profile Velocity (FB_MCProfileVelocity) .....	54
5.3.9	Reset (FB_MCReset).....	56
5.3.10	Touch Probe (FB_MCTouchProbe) .....	57
5.3.11	Velocity mode (FB_MCVelocity) .....	59
5.3.12	Object Access – Access Request (FB_Mutex_ObjAccess) .....	62
5.3.13	Object Access (FB_ObjAccess) .....	63
5.3.14	Stop .....	65
<b>5.4</b>	<b>Support FBs and FCs.....</b>	<b>67</b>
5.4.1	FB_ObjAccessMap .....	67
5.4.2	FB_PROFINetMap .....	68
5.4.3	FB_PROFIBusMap.....	70
5.4.4	FC_GetFaultMessage .....	72
5.4.5	FC_GetWarningMessage .....	72
5.4.6	FB_B_AxisMgr_Constants.....	72
<b>6</b>	<b>Data Unit Types.....</b>	<b>73</b>
<b>6.1</b>	<b>Constants.....</b>	<b>73</b>
6.1.1	AxisMgr Status .....	73
6.1.2	Object Access .....	74
6.1.3	Shutdown Procedure .....	74
6.1.4	Homing Method .....	75
6.1.5	Error strings.....	78
<b>6.2</b>	<b>Structures.....</b>	<b>79</b>
6.2.1	st_Axis .....	79
6.2.2	st_Homing .....	80
6.2.3	st_ProfilePosition.....	80
6.2.4	st_ProfileTorque.....	82
6.2.5	st_ProfileVelocity.....	82
6.2.6	st_ObjectAccess .....	83
6.2.7	st_ObjectAccessData .....	84
6.2.8	st_TouchProbe.....	85
6.2.9	st_TouchProbeChannel .....	85
6.2.10	st_TouchProbeChannelStatus .....	86
6.2.11	st_Velocity .....	87
<b>7</b>	<b>Example.....</b>	<b>88</b>
<b>7.1</b>	<b>Variable definition .....</b>	<b>88</b>
<b>7.2</b>	<b>FB_AxisMgr instance .....</b>	<b>90</b>
<b>7.3</b>	<b>SCL Program – Velocity Mode .....</b>	<b>92</b>
<b>7.1</b>	<b>SCL Program – Object Access.....</b>	<b>94</b>
<b>8</b>	<b>List of Figures.....</b>	<b>96</b>
<b>9</b>	<b>References .....</b>	<b>97</b>
<b>10</b>	<b>Revision Index (R) .....</b>	<b>98</b>



## 1 General information about the Documentation

For better clarity, the documentation of the library is structured according to the customer-specific requirements.

This documentation was written in English language. The English documentation is the original one. Other language versions are translated.

### 1.1 This document

The user manual contains important information about the implementation of the *Bonfiglioli Axis Manager* Library and use of its functionalities. Compliance with this user manual contributes to avoiding risks, minimizing repair cost and downtimes.

For this reason, make sure to read the user manual carefully.



#### **WARNING**

Bonfiglioli S.p.A. shall not be held liable for any damage caused by any non-compliance with the documentation.



In case any problems occur which are not covered by the documentation sufficiently, please contact the manufacturer.

### 1.2 Prerequisites

In order to fully understand the content of this document, the reader must be familiar with the following topics:

- ➔ Siemens TIA Portal programming languages;
- ➔ Bonfiglioli's inverter series;
- ➔ Fieldbus technologies;
- ➔ Object Oriented Programming;
- ➔ DSP CiA402 specifications and motion kinematics.

### 1.3 Warranty and liability

Bonfiglioli S.p.A. would like to point out that the contents of this user manual do not form part of any previous or existing agreement, assurance or legal relationship. Neither are they intended to supplement or replace such agreements, assurances or legal relationships. Any obligations of the manufacturer shall solely be based on the relevant purchase agreement which also includes the complete and solely valid warranty stipulations. These contractual warranty provisions are neither extended nor limited by the specifications contained in this documentation.

The manufacturer reserves the right to correct or amend the specifications, product information and omissions in these operating instructions without notice. The manufacturer shall not be liable for any damage, injuries or costs which may be caused by the aforementioned reasons.

In addition to that, Bonfiglioli S.p.A. excludes any warranty/liability claims for any personal and/or material damage if such damage is due to one or more of the following causes:

- inappropriate use of the library,
- non-compliance with the instructions, warnings and prohibitions contained in the documentation,
- unauthorized modifications of library,
- insufficient monitoring of parts of the machine/plant which are subject to wear,
- repair work at the machine/plant not carried out properly or in time,
- catastrophes by external impact and Force Majeure.



---

## 1.4 Copyright

In accordance with applicable law against unfair competition, this user manual is a certificate. Any copyrights relating to it shall remain with

### **Bonfiglioli S.p.A.**

Via Cav. Clementino Bonfiglioli 1 - 40012 Calderara di Reno (BO)

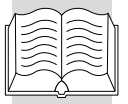
Tel. +39 (0)51 6473111

This user manual is intended for the user. Any disclosure or copying of this document, exploitation, and communication of its contents (as hardcopy or electronically) shall be forbidden, unless permitted expressly.

Any non-compliance will constitute an offense against the copyright law dated 09 September 1965, the law against unfair competition and the Civil Code and may result in claims for damages. All rights relating to patent, utility model or design registration reserved.

## 1.5 Storage

The documentation forms an integral part of the library.



## 2 General safety instructions and information on use

The chapter "General safety instructions and information on use" contains general safety instructions for the Operator and the Operating Staff. At the beginning of certain main chapters, some safety instructions are included which apply to all work described in the relevant chapter. Special work-specific safety instructions are provided before each safety-relevant work step.

### 2.1 Warning information and symbols used in the user manual

#### 2.1.1 Hazard classes

The following hazard identifications and symbols are used to mark particularly important information:



#### **DANGER**

Identification of immediate threat holding a **high** risk of death or serious injury if not avoided.



#### **WARNING**

Identification of immediate threat holding a **medium** risk of death or serious injury if not avoided.







#### **CAUTION**

Identification of immediate threat holding a **low** risk of minor or moderate physical injury if not avoided.


#### **NOTE**

Identification of a threat holding a risk of material damage if not avoided.

#### 2.1.2 Hazard symbols


Symbol	Meaning	Symbol	Meaning
	General hazard		Suspended load
	Electrical voltage		Hot surfaces

#### 2.1.3 Prohibition signs


Symbol	Meaning
	No switching; it is forbidden to switch the machine/plant, assembly on




### 2.1.4 Personal safety equipment

Symbol	Meaning
	Wear body protection


### 2.1.5 Recycling

Symbol	Meaning
	Recycling, to avoid waste, collect all materials for reuse


### 2.1.6 Grounding symbol

Symbol	Meaning
	Ground connection

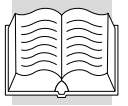
### 2.1.7 ESD symbol

Symbol	Meaning
	ESD: Electrostatic Discharge (can damage components and assemblies)

### 2.1.8 Information sign

Symbol	Meaning
	Tips and information making using the library easier.





### 3 Introduction

The library provides function and function blocks as tools to manage and control the Bonfiglioli's inverter and drive series.

#### 3.1 Information about the product

Before to attempt a solution (machine or process) for a specific application using the FB in the library, it is required to consider the usage of right procedure which includes, besides the rest, risks analysis, functional safety, components compatibility, test and validation of the system related to this library.



#### **WARNING**

##### **WRONG USAGE OF FBs**

- ➔ Do a safety analysis for the application and for the installed devices.
- ➔ Verify that the FBs are compatible with the devices in the system and that they do not have unwanted behavior on the system work.
- ➔ Use appropriate parameters, limit the values in particular, and observe the usage of the machine and the stop behavior.
- ➔ Verify that all the sensor and the actuators are compatible with the selected FB .
- ➔ Test deeply all the functions during the validation and the commissioning in each operating mode.
- ➔ Provide independent methods for the critical control functions (emergency stop, etc...) after an analysis about safety and the related rules.

**The inobservance of these instructions can lead to death, injuries or damage to the equipment.**



#### **WARNING**

##### **UNEXPECTED OPERATION OF THE EQUIPMENT**

Update the application software according to the necessities, paying attention mostly to the modification of the I/O addresses every time that the hardware configuration changes. Pay also attention performing online changes in the software, variable address areas and pointers in use may have unexpected consequences.



## 4 Library Overview

The library provides functions and function blocks as tools to manage and control the Bonfiglioli's inverter and drive series. This library has been implemented with **Siemens TIA Portal version 20**. Compatibility with other versions is not guaranteed.

The library has been tested in combination with Siemens CPU 1511T-1 PN. Compatibility with other devices is not guaranteed.

Features	Value
Library title	Bonfiglioli
Company	Bonfiglioli
Author	Motion Control Development

### 4.1 Example Project

In conjunction with the library, the example project available in the library is provided. The example project demonstrates how to implement the components from the Bonfiglioli Axis Manager.

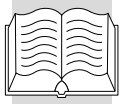
The specific application requirements may be different from those assumed for any related examples, templates or architectures described herein. In that case, it is needed to adapt the information provided in this and other related documents to the particular needs. To do so, the user need to consult the specific product documentation of the hardware and/or software components that you may add or substitute for any information specified in this documentation. Pay particular attention and conform to any safety information, different electrical requirements and normative standards that would apply to the user's adaptation.



#### **WARNING**

##### **REGULATORY INCOMPATIBILITY**

Be sure that all equipment applied and systems designed comply with all application local, regional and national regulations and standards.  
Failure to follow these instructions can result in death, serious injury, or equipment damage.



## 4.2 Overview of the FBs

The library contains the following function block:

Function Block	Description
FB_AxisMgr ( <a href="#">5.2</a> )	<p>This FB offer an easy and intuitive interface to manage the control of AxiaVert drive via PROFINet® or PROFibus®. It gives the possibility to easily manage the non interpolated modes of operation, such as Profile Position Mode, Homing Mode, Profile Velocity Mode...</p> <p>This is the <u>only</u> FB that must be instantiated.</p>
FB_MCHardwareError ( <a href="#">5.3.3</a> )	Monitor AxiaVert drive errors
FB_MCHoming ( <a href="#">5.3.4</a> )	Activate the Homing mode (Modes Of Operation = 6)
FB_MCPower ( <a href="#">5.3.5</a> )	Power on and off AxiaVert drive
FB_MCProfilePosition ( <a href="#">5.3.6</a> )	Activate the Profile Position mode (Modes Of Operation = 1)
FB_MCProfileTorque ( <a href="#">5.3.7</a> )	Activate the Profile Torque mode (Modes Of Operation = 4)
FB_MCProfileVelocity ( <a href="#">5.3.8</a> )	Activate the Profile Velocity mode (Modes Of Operation = 3)
FB_MCReset ( <a href="#">5.3.9</a> )	Acknowledge AxiaVert drive errors
FB_MCTouchProbe ( <a href="#">0</a> )	Activate the Touch Probe functionality
FB_MCVelocity ( <a href="#">5.3.11</a> )	Activate the Velocity mode (Modes Of Operation = 2)
FB_Mutex_ObjAccess ( <a href="#">5.3.12</a> )	Support utility designed to manage and coordinate access requests to the Object Access
FB_ObjAccess ( <a href="#">5.3.13</a> )	Reading and writing operations on Object Access

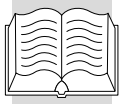


## 4.3 Overview of the Constants

Constant	Description
Axis Manager State ( <a href="#">6.1.1</a> )	Constants for the definition of the actual state of the Axis Manager
Object Access ( <a href="#">6.1.2</a> )	Constants for Object Access data type
Shutdown procedure ( <a href="#">6.1.3</a> )	Constants for set the shutdown procedure
Homing method ( <a href="#">6.1.4</a> )	Constant to set the Homing method
Error strings ( <a href="#">6.1.5</a> )	Error strings definition

## 4.4 Overview of the Structures

Structure	Description
st_Axis ( <a href="#">6.2.1</a> )	Parameters that must be connected to AxiaVert Telegrams
st_Homing ( <a href="#">6.2.2</a> )	Parameters for the Homing mode of operation
st_ProfilePosition ( <a href="#">6.2.3</a> )	Parameters for the Profile Position mode of operation
st_ProfileTorque ( <a href="#">6.2.4</a> )	Parameters for the Profile Torque mode of operation
st_ProfileVelocity ( <a href="#">6.2.5</a> )	Parameters for the Profile Torque mode of operation
st_ObjAccess ( <a href="#">6.2.6</a> )	Parameters that must be connected to Telegram 120 (PROFINet®) or to the Object Access Channel (PROFIBus®)
st_ObjAccessData ( <a href="#">6.2.7</a> )	Parameters for Object Access
st_TouchProbe ( <a href="#">6.2.8</a> )	Parameters for Touch Probe
st_TouchProbeChannel ( <a href="#">6.2.9</a> )	Parameters to enable Touch Probe functionality on each channel
st_TouchProbeChannelStatus ( <a href="#">6.2.10</a> )	Touch Probe parameters read from the AxiaVert drive
st_Velocity ( <a href="#">6.2.11</a> )	Parameters for the Velocity mode of operation



## 4.5 Naming convention

The library follows a systematic naming convention based on

*scope prefix + data type indicator + descriptive name.*

### Scope Prefix

Scope Prefix	Format	Description
<i>i_</i>	i_yyyName	Function block input parameter. Example: <i>i_xPower</i> (boolean input)
<i>q_</i>	q_yyyName	Function block output parameter Example: <i>q_xDone</i> (boolean output)
<i>l_</i>	l_yyyName	Internal function block variable Example: <i>l_iCounter</i> (integer)
<i>No Prefix</i>	yyyName	Input-output function block variable Example: stAxis (structure)
<i>NAME</i>	All Capital Letters	Fixed values and enumeration constants Example: OBJACCESS_UINT32

### Data Type Indicator

Indicator	Data Type	Example
<i>x</i>	Bool	<i>i_xEnable, q_xReady</i>
<i>i</i>	Int	<i>i_iPosition, l_iState</i>
<i>ui</i>	UInt	<i>i_uiTimeout, q_uiStatus</i>
<i>r</i>	Real	<i>i_rStopDeceleration, i_rFactorPos</i>
<i>st</i>	Structure	<i>i_stHoming, q_stTouchProbeCh1Status</i>

This convention ensures consistent, readable code with clear variable purpose and data type identification throughout the library.



## 5 Overview of Axis Manager

The main goal of the Axis Manager, described in this document, is to help the user to easily implement an effective control of an axis using Bonfiglioli's AxiaVert. The FBs provided by the library uses a fieldbus to give commands and to check the status of the inverter (control must be set to State Machine). Supported fieldbuses are PROFINet® and PROFIBus®.

The FB are designed in order to be as more compliant with CiA402 as possible, but they are specifically developed to work in combination with Bonfiglioli's inverter only.

### 5.1 State diagram

Axis Manager, implemented in the Function Block FB\_AxisMgr, follow the logic behind the state machine defined by PLCOpen, but with some differences. Each function can be called in certain state, and some of them lead to a state transition which abort the actual running function. The picture below represents the possible states and the possible transitions between them.

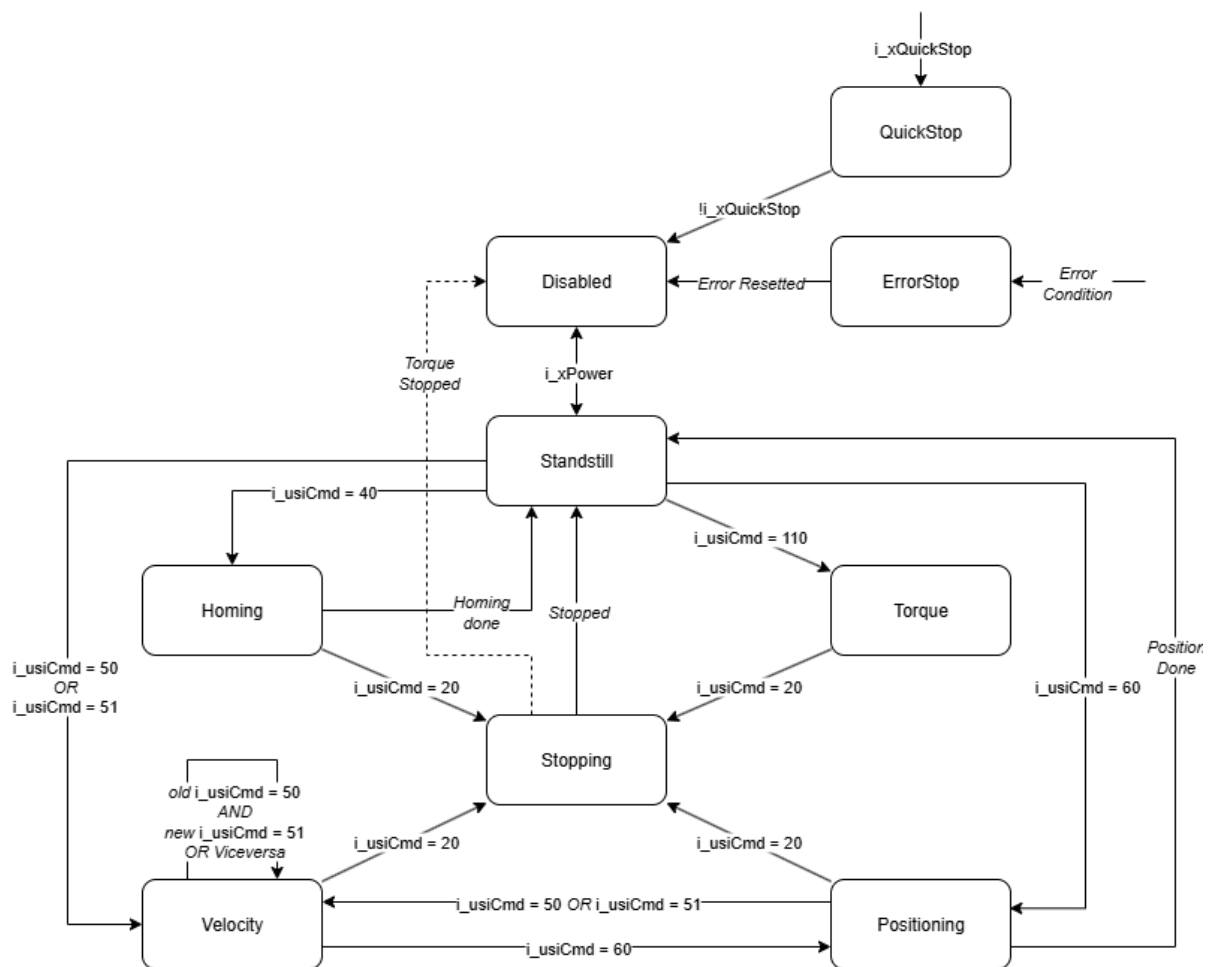
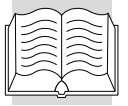


Figure 1: Axis Manager State Diagram



---

## 5.2 AxisMgr Interface

The Axis Manager function block, *FB\_AxisMgr*, serves as a centralized control interface for managing servo drive operations and motion control functions.

It acts as the primary interface between higher-level control logic and individual servo drive systems, simplifying complex motion control applications while ensuring safe and reliable operation.

### Core Functionality

- ➔ Drive State Management: Controls power-up, shutdown, and operational state transitions
- ➔ Motion Control Integration: Coordinates various motion function blocks (homing, positioning, velocity control)
- ➔ Error Handling: Manages fault detection, error recovery, and safety operations
- ➔ Communication Interface: Handles Object Access protocol communications with servo drives

### Key Features

- ➔ Unified Control: Single point of control for multiple motion control operations
- ➔ Resource Arbitration: Manages access to shared communication protocols
- ➔ Safety Integration: Implements proper shutdown sequences and emergency stops
- ➔ Status Monitoring: Provides comprehensive feedback on drive and motion states

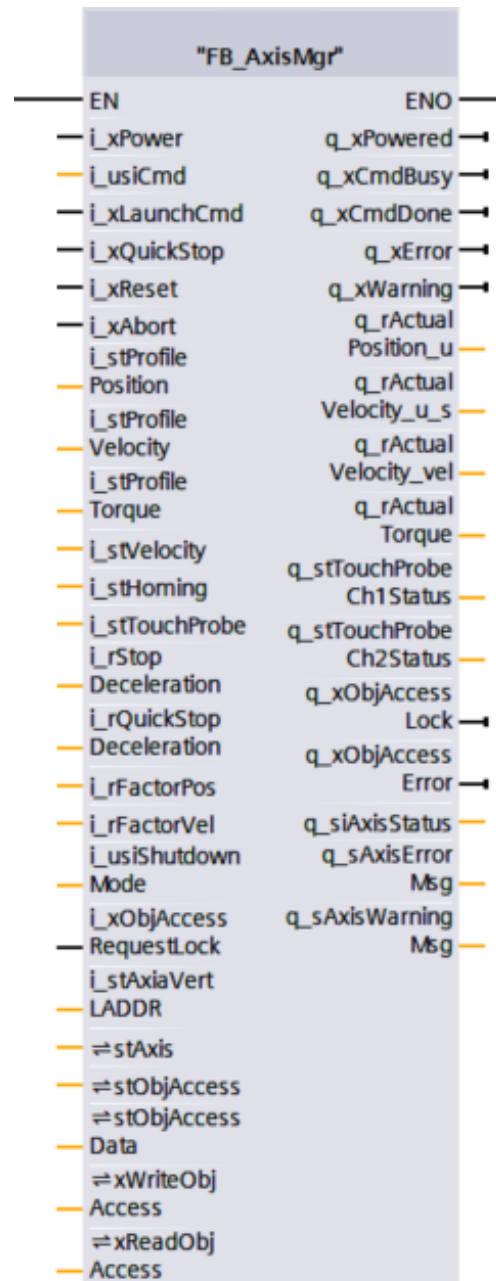


Figure 2: FB\_AxisMgr

This FB allows the user to easily manage AxiaVert drive series in PROFINet® and PROFIBus®. The motion profiles defined by the CiA402 specifications has been implemented, in combination with the manufacturer specific features. The controller does not calculate and send the trajectory to the drive, but the drive's motion profile generator is used. This also means that the controller is not intended to have demanding real-time capabilities, since the control of the drive may not be deterministic.

This FB only works in combination with AxiaVert drive. The FB is designed to work in State Machine control only (*obj. 0x2200 = 3 - State Machine*). Depending on the actual drive configuration, some functions may not be supported.





Input	Data Type	Default	Description
<i>i_xPower</i>	Bool	false	Rising edge: power on the axis. Falling edge: disable the axis.
<i>i_usiCmd</i>	USInt	0	Command selection input. It defines which functions should be executed. The possible values and the commands supported are listed in the FB description. The command will be taken into account only after a rising edge on <i>i_xLaunchCmd</i> .
<i>i_xLaunchCmd</i>	Bool	false	Rising Edge: launch the selected command in <i>i_usiCmd</i> .
<i>i_xQuickStop</i>	Bool	false	Rising Edge: put the axis in EmergencyStop state and trigger the emergency ramp of the drive. As long as it remains TRUE, it prevents the axis to get powered.
<i>i_xReset</i>	Bool	false	Rising edge: clear drive's fault and FB's diagnostic
<i>i_xAbort</i>	Bool	false	Rising edge: abort all Motion Control FB's
<i>i_stProfilePosition</i>	st_ProfilePosition		Input parameters for the Profile Position function
<i>i_stProfileVelocity</i>	st_ProfileVelocity		Input parameters for the Profile Velocity function
<i>i_stProfileTorque</i>	st_ProfileTorque		Input parameters for the Profile Torque function
<i>i_stVelocity</i>	st_Velocity		Input parameters for the Velocity Mode function
<i>i_stHoming</i>	st_Homing		Input parameters for the Homing Mode function
<i>i_stTouchProbe</i>	st_TouchProbe		Input parameters for the Touch Probe function
<i>i_rStopDeceleration</i>	Real	0.0	Deceleration ramp for STOP command
<i>i_rQuickStopDeceleration</i>	Real	0.0	Deceleration ramp for <i>i_xQuickStop</i> command
<i>i_rFactorPos</i>	Real	182.04	Scaling factor [increments/units]. It defines the amount of the drive's increment that represent 1u displacement in the user program. This value is relevant for the position-controlled functions (Profile Position, Profile Velocity, Homing, ecc...). Default 182.04 = 65535.0/360.0 See <a href="#">5.3.2.2</a>
<i>i_rFactorVel</i>	Real	1.0	Scaling factor [VL Units/vel. units]. It defines the amount of drive's VL units that represent 1u speed in the user program. This value is relevant for the speed-controlled functions (i.e. Velocity Mode). See <a href="#">5.3.2.2</a>
<i>i_usiShutdownMode</i>	USInt	0	Shutdown mode where the axis is disabled See <a href="#">6.1.3</a>
<i>i_xObjAccessRequest</i>	Bool	false	TRUE: External request for Object Access usage
<i>i_stAxiaVertLADDR</i>	HW_DEVICE	0	DP slave hardware identifier



Output	Data Type	Default	Description
<i>q_xPowered</i>	Bool	false	TRUE: axis is powered on.
<i>q_xCmdBusy</i>	Bool	false	TRUE: the function indicated by <i>i_usiCmd</i> is still busy or the axis is still powering on or off.
<i>q_xCmdDone</i>	Bool	false	TRUE: the function indicated by <i>i_usiCmd</i> has been finished. Relevant only in commands that can reach a finish (i.e.: Positioning, Stop, Home, Power on and off). It remains TRUE as long as a new command is triggered (rising edge of <i>i_xLaunchCmd</i> ) or <i>i_xPower</i> change value
<i>q_xError</i>	Bool	false	TRUE: an error arise
<i>q_xWarning</i>	Bool	false	TRUE: at least a warning arise
<i>q_rActualPosition_u</i>	Real	0.0	Actual position of the axis [u], referred to the last home referencing. The value is updated only if <i>stAxis.i_diActualPos</i> has been provided, and it is scaled by <i>i_rFactorPos</i>
<i>q_rActualVelocity_u_s</i>	Real	0.0	Actual velocity of the axis [u/s]. The value is updated only if <i>stAxis.i_diActualVelocity_u_s</i> has been provided, and it is scaled by <i>i_rFactorPos</i>
<i>q_rActualVelocity_vel</i>	Real	0.0	Actual velocity of the axis [vel. units]. The value is updated only if <i>stAxis.i_iActualVelocity_vel</i> has been provided, and scaled by <i>i_rFactorVel</i> .
<i>q_rActualTorque</i>	Real	0.0	Actual torque of the axis [Nm]. The value is updated only if <i>stAxis.i_iActualTorque</i> has been provided, and it is scaled by the rated torque of the motor (read out during init).
<i>q_stTouchProbeCh1Status</i>	st_TouchProbeChannelStatus		Touch probe channel 1 data. See <a href="#">st_TouchProbeChannelStatus</a>
<i>q_stTouchProbeCh2Status</i>	st_TouchProbeChannelStatus		Touch probe channel 2 data. See <a href="#">st_TouchProbeChannelStatus</a>
<i>q_xObjAccessLock</i>	Bool	false	TRUE: external access to Object Access granted
<i>q_xObjAccessError</i>	Bool	false	TRUE: error occur during the read or write operation on Object Access
<i>q_siAxisStatus</i>	SInt	0	AxisMgr status, see <a href="#">AxisMgr Status</a>
<i>q_sAxisErrorMsg</i>	String	"	Description of the error
<i>q_sAxisWarningMsg</i>	String	"	Description of the warnings

InOut	Data Type	Default	Description
<i>stAxis</i>	st_Axis		Axis I/O parameters
<i>stObjAccess</i>	st_ObjAccess		Data for Object Access
<i>stObjAccessData</i>	st_ObjAccessData		Data to read or write on Object Access
<i>xWriteObjAccess</i>	Bool	false	TRUE: execute a write on Object Access. Once the request is complete, <i>xWriteObjAccess</i> fall to FALSE
<i>xReadObjAccess</i>	Bool	false	TRUE: execute a read on Object Access. Once the request is complete, <i>xReadObjAccess</i> fall to FALSE



## 5.2.1 State transitions

Axis Manager passes through the states due to a trigger that comes from the user program either via a Boolean input (*i\_xPower*, *i\_xReset*, ...) or via control command *i\_usiCmd* and the *i\_xLaunchCmd*. Moreover, there are states (e.g.: *ErrorStop*) which are reached from an external condition (like an inverter fault), and states that are reached due to the successful completion of another (e.g. axis goes automatically in *Standstill* after a *Position* done).

Triggers that comes from the interface are usually evaluated on the edge (rising or falling or both) of the input, and not on the level.

### 5.2.1.1 i\_xAbort

The abort function is triggered by setting the *i\_xAbort* bit to TRUE, providing immediate system-wide termination control.

All functions currently in progress are immediately aborted, causing the Axis Manager state to transition to ABORTED state without any graceful completion of ongoing operations.

While *i\_xAbort* remains active, all function activation is disabled with no new operations able to be initiated, creating a complete system lockout until the abort condition is cleared.

#### NOTE

Quick Stop Functionality remains accessible during abort conditions, allowing controlled motor deceleration if required.



#### WARNING

The restoration from an abort condition is entirely the user's responsibility and cannot be performed automatically by the system, which includes clearing the *i\_xAbort*.

Since the motor maintained its operational state during the abort, the user must evaluate the current system condition, verify motor position and status, and determine appropriate next steps before reactivating any motion functions. The system will not automatically return to its previous operational mode or complete interrupted sequences.

### 5.2.1.2 From *Disabled* to *Standstill*

AxisMgr start in Disabled state and remains in this state until a rising edge of *i\_xPower* arise.

Setting *i\_xPower* input to TRUE, axis reach *Standstill* from *Disabled*, and it goes back to *Disabled* when the input is reset. If the axis becomes *Disabled* due to other reasons (e.g. at the end of a QuickStop) and *i\_xPower* is still TRUE, then the axis does not get *Standstill* automatically, but can be reached by means of a new rising edge (see [Figure 1: Axis Manager State Diagram](#) for more details).

*q\_xPowered* indicates the actual status of the power stage of the drive, it could take a while to get power-on/off since the *i\_xPower* input was set/reset.

When the axis is powered, if a rising edge on *i\_xQuickStop* is detected, the emergency ramp of the axis will be immediately performed, and the axis state becomes *QuickStop*. The value set in *i\_rQuickStopDeceleration* is relevant, if it remains 0, the value stored in the drive will take effect. When *i\_xQuickStop* falls to FALSE, the axis will be Disabled and *Disabled* state will occur.



### WARNING

#### Unexpected behavior of QuickStop

- ➔  $i\_xQuickStop$  input reset the 3<sup>rd</sup> bit of the drive's remote control word. Behavior of the drive, when a Quick Stop has been requested, may be selectable via *obj. 0x605A - Quick Stop Option Code*. Value of  $i\_rQuickStopDeceleration$  is only relevant when this object is set to 2.
- ➔ At the end of a Quick Stop procedure, the drive may keep the motor powered and standstill until *obj. 0x253B - Holding Time* has been expired (refer to the drive documentation)
- ➔ Deceleration used when in EmergencyStop (i.e.:  $i\_xQuickStop$  is triggered or ErrorStop state occurred) depends on the actual Mode of Operation:
  - Mode Of Operation  $\neq$  2: pos. units/s<sup>2</sup> (*obj. 0x6085* is used)
  - AxisMgr "Light", Mode Of Operation = 2: vel. units/s (*obj. 0x604A* is used)
- ➔ If  $i\_rQuickStopDeceleration$  is set to 0, the value stored in the drive will take effect.

To restore the error state, a rising edge to  $i\_xReset$  is necessary. After the reset the axis switches to *Disabled* state.

### NOTE

- ➔ Resetting an error that is still active has no effect. The axis remains in the *ErrorStop* state.
- ➔ If an hardware error was detected, the reset procedure may take some time to acknowledge the drive fault. If the drive fault is non-acknowledgeable, the reset procedure will probably end into an error.
- ➔ If the error is caused by an user or FB error, then the restoration, thanks to  $i\_xReset$ , is immediate.
- ➔  $i\_xReset$  is always evaluated on the rising edge

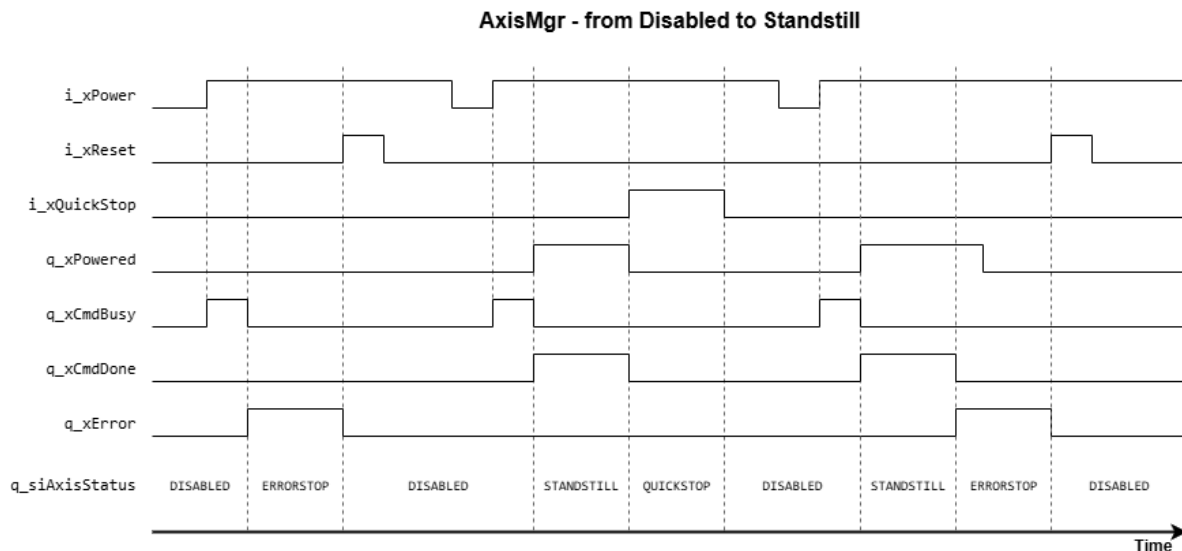
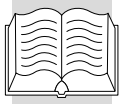


Figure 3: State transitions from Disabled to Standstill



## Powering off the drive

When *i\_xPower* transitions to false, the system behavior is determined by the *i\_usiShutdownMode* parameter, which supports three distinct shutdown sequences:

- ➔ **SHUTDOWN\_DISABLE\_VOLTAGE (0):** The Control Word is immediately set to 0, providing an instant shutdown with all control bits cleared simultaneously
- ➔ **SHUTDOWN\_DISABLE\_OPERATION (1):** This mode follows a sequential shutdown process:
  - The Enable Operation bit (bit 4) in the Control Word is cleared
  - The Switch On bit (bit 1) in the Control Word is cleared after the Operation Enabled bit (bit 3) in the Status Word confirms deactivation
  - Once the Switched On bit (bit 2) in the Status Word is cleared, the remaining control bits, Quick Stop bit (bit 3) and Enable Voltage bit (bit 2), are deactivated
- ➔ **SHUTDOWN\_SHUTDOWN (2):** This mode initiates a controlled shutdown by:
  - Simultaneously clearing both the Enable Operation bit (bit 4) and Switch On bit (bit 1) in the Control Word
  - Monitoring the Status Word until the Switched On bit (bit 2) is cleared
  - Finally clearing the Quick Stop bit (bit 3) and Enable Voltage bit (bit 2) to complete the shutdown sequence

Each shutdown mode provides different levels of control over the power-down sequence, allowing users to select the appropriate method based on their application requirements and safety considerations (see [Figure 4: Shutdown Modes](#)).

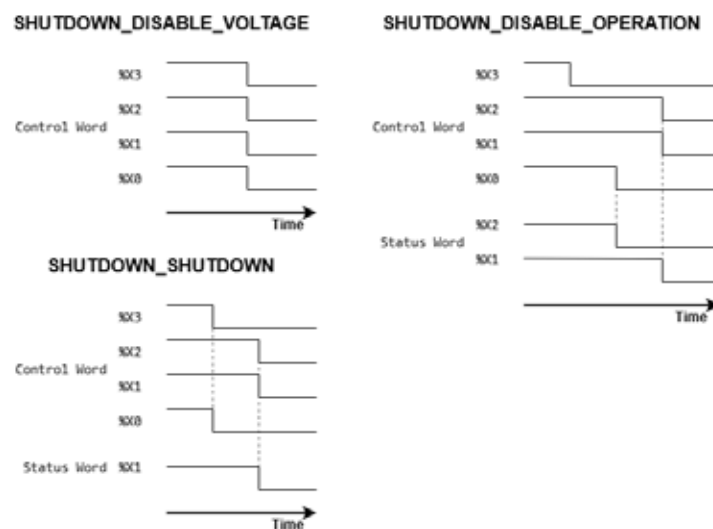


Figure 4: Shutdown Modes



### 5.2.1.3 Launching Commands

For trigger Motion Control Functions  $i\_usiCmd$  and  $i\_xLaunchCmd$  are used. Functions are associated to a unique command number and can be triggered only in certain states (see [Figure 1: Axis Manager State Diagram](#)) otherwise an error occur.

As soon as  $i\_xLaunchCmd$  is triggered (see [Figure 5: Launching Commands](#)), the corresponding function start its execution. If none of the functions matches the command number defined in  $i\_usiCmd$  nothing happen.

$q\_xCmdDone$  and  $q\_xCmdBusy$  signal the actual state of the function that has been launched. In case of a function change (e.g. from *Velocity* to *Position*), the old function is aborted and the new function in started, and  $q\_xCmdBusy$  still remains active.

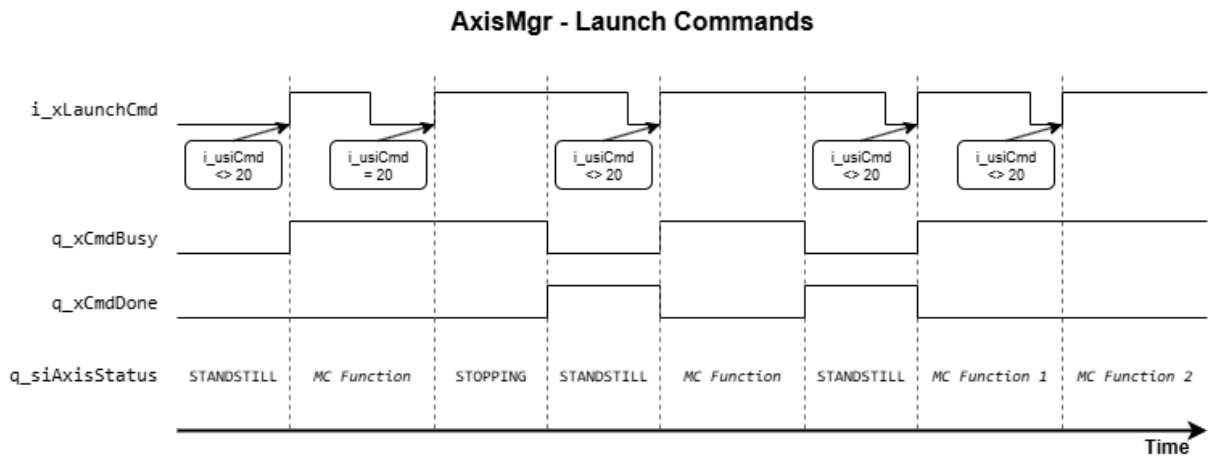
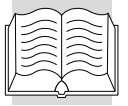


Figure 5: Launching Commands



#### 5.2.1.4 *ErrorStop*: possible causes, *q\_xFault* and *i\_xReset*

The *ErrorStop* state can be reached due to the following reasons:

- **Hardware error:** a drive error has been detected. AxisMgr will immediately report a fault (i.e.: *q\_xFault* set to *TRUE*)
- **User error:** the user program has generated an error (e.g.: invalid input data, invalid command state...).
- **FB error:** the functions itself has encoured an issue (e.g.: timeouts, invalid conditions...)
- **Communication error:** communication between the controller and the physical drive has occurred in an issue (parameter access timeout, unable to writing/reading parameters...).

In order to restore the error state, a rising edge given to *i\_xReset* is necessary. If *q\_xPowered* is set, after the reset, the axis switches to *Standstill* state, otherwise it switches to *Disabled* state.

##### NOTE

- The behavior of the axis in the case of an *ErrorStop* condition is the same as triggering *i\_xQuickStop* (see [5.2.1.2](#)).
- If an hardware error was detected, the reset procedure may take some time to acknowledge the drive fault. If the drive fault is non-acknowledgeable, the reset procedure will probably end into an error.
- If the error is caused by an user or FB error, then the restoration, thanks to *i\_xReset*, is immediate.
- *i\_xReset* is always evaluated on the rising edge.



## 5.2.2 Fieldbus management

AxisMgr is designed to manage PROFINet® and PROFibus® fieldbuses. Communication with the drive is established thanks to **Cyclic data**.

- **Cyclic data:** those information are sent/received to/from the drive in a cyclic manner, and updated at each bus cycle task. Cyclic data represent an integral part of the input/output interface of the FB and should be linked to the I/O mapping of the device (a set of support FBs is implemented in order to help the user in the mapping between the FB and the I/O. See [Support FBs](#)).



### **WARNING**

#### **CYCLIC DATA**

- Some functions requires specific cyclic data to be exchanged with the drive. If those information cannot be transferred/received (e.g.: they has not been mapped), the functions may not work as expected.
- Cyclic data can be eventually assigned to the FB in/out interface by means of assignations in the user program. The user program should not perform any other operations (scaling, casting) that can alterate the datas to/from the physical device.





### 5.2.2.1 PROFInet®

This outlines the configuration rules and considerations for setting up communication between a PLC and AxiaVert inverter drives using PROFInet® communication.

Device overview				
	...	Module	Rack	Slot
		▼ Axia-Vert-Single-DAP	0	0
		▶ Interface PN-IO	0	0 X1
		▼ AxiaVert Axis_1	0	1
		Axia Telegram 120 Object Access	0	1 1
		Axia Telegram 102 IRT Point-to-Point	0	1 2
		Axia Telegram 111 Free Usage, Axis 1 OR Axis ...	0	1 3
			0	1 4
			0	1 5
			0	1 6

Figure 6: Example of PROFInet® map

Restrictions for user-defined configuration settings:

- Maximum **5 submodules** can be selected in total
- Maximum **24 bytes (12 words)** of data transfer capacity
- **Axis-specific telegrams** (001, 101, 102, 104): Can only be used **once per motor axis**
- **Axis-independent telegram** (110, 111, 112, 113, 114, 115): Can only be used **once per entire drive device** and only for axis 1 or axis 2 (where applicable)
- **Object Access:**
  - Has special restriction - only allowed in communication slot 1
  - is **mandatory**
- **Control Word** (*obj. 0x6040*), **Mode of Operation** (*obj. 0x6060*), **Status Word** (*obj. 0x6041*), **Mode of Operation Display** (*obj. 0x6061*) **must** be mapped in at least one telegram

It is only necessary to link the cyclic data relevant for the functions that the AxisMgr is supposed to use. AxisMgr will NOT report any error if any mandatory link is missing.

it is possible that informations linked to the telegram in the PLC need a corresponding mapping made on the inverter side manually (i.e.: via Axia Manager), see the [Velocity mapping example](#).

#### NOTE

- Using function-specific telegrams (i.e.: telegrams 001, 101, 102 and 104), the user shall not make any mapping on the inverter side; the information are already managed as they are presented in the telegram used.



## Velocity mapping example

This provides a practical example of configuring an Axia drive for **Velocity Mode** operation (Mode of Operation = 2) using multiple telegrams. Telegram 102 handles the control interface while Telegram 110 handles the velocity-specific data:

- ➔ Telegram 102: core control/status communication
  - Control Word
  - Mode of Operation
  - Status Word
  - Mode of Operation Display
- ➔ Telegram 110: velocity data
  - v/target velocity
  - v/velocity actual value

In the Device overview of the AxiaVert drive add following telegrams ([Figure 7](#)):

Telegram	I address	Q address
Telegram 120	0 .. 7	0 .. 7
Telegram 102	8 .. 18	8 .. 18
Telegram 110	19 .. 22	19 .. 22

Device overview						
	Module	Rack	Slot	I address	Q address	Type
	▼ Axia-Vert-Single-DAP_1	0	0			AxiaVert Single Axi...
	▶ Interface PN-IO	0	0 X1			Axia-Vert-Single-DAP
	▼ AxiaVert Axis_1	0	1			AxiaVert Axis
	Axia Telegram 120 Object ...	0	1 1	0...7	0...7	Axia Telegram 120 ...
	Axia Telegram 102 IRT Point...	0	1 2	8...18	8...18	Axia Telegram 102 ...
	Axia Telegram 110 Free Us...	0	1 3	19...22	19...22	Axia Telegram 110 ...
		0	1 4			
		0	1 5			
		0	1 6			

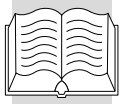
Figure 7: PROFInet® Velocity mode mapping example – Drive

In AxiaManager set the following drive parameters ([Figure 8](#)):

Object	SubIdx	Name	Value
0x2200	1..4	Control Mode	0x03: State Machine
0x25E8		VL Target Velocity Source	0x0391618: PD1 OUT Word 0 Int 16
0x3915	9	PD1 IN Word 0	0x0604400: Actual Velocity 6044:00

Object	SubIdx	Name	Unit	Type	Value
0x2200	1	Control Mode		enum	0x00000003: State Machine
0x2200	2	Control Mode		enum	0x00000003: State Machine
0x2200	3	Control Mode		enum	0x00000003: State Machine
0x2200	4	Control Mode		enum	0x00000003: State Machine
0x25E8		VL Target Velocity Source		enum	0x00391618: PD1 OUT Word 0 Int16
0x3915	9	PD1 IN Word 0		enum	0x00604400: Actual Velocity 6044:00

Figure 8: PROFInet® Velocity mode mapping example - AxiaManager configuration



Define the following PLC tags (Figure 9):

Name	Data Type	Address	Description
<i>q_wCtrlWord</i>	Word	%QW8	Control Word
<i>i_wStatusWord</i>	Word	%IW8	Status Word
<i>q_siModeOfOp</i>	SInt	%QB18	Mode of Operation
<i>i_siModeOfOpDsp</i>	SInt	%IB18	Mode of Operation Display
<i>q_iTargetVelocity</i>	Int	%QW19	v/ Target Velocity
<i>i_iActualVelocity</i>	Int	%IW19	v/ Actual Velocity
<i>i_wCmd</i>	Byte	%IB0	Input values of Telegram 120
<i>i_iSubIdx</i>	Byte	%IB1	
<i>i_uiIndex</i>	UInt	%IW2	
<i>i_wDataB0</i>	Byte	%IB4	
<i>i_wDataB1</i>	Byte	%IB5	
<i>i_wDataB2</i>	Byte	%IB6	
<i>i_wDataB3</i>	Byte	%IB7	
<i>q_wCmd</i>	Byte	%QB0	Output values for Telegram 120
<i>q_iSubIdx</i>	Byte	%QB1	
<i>q_uiIndex</i>	UInt	%QW2	
<i>q_wDataB0</i>	Byte	%QB4	
<i>q_wDataB1</i>	Byte	%QB5	
<i>q_wDataB2</i>	Byte	%QB6	
<i>q_wDataB3</i>	Byte	%QB7	





















	Name	Data type	Address
1	 <i>q_wCtrlWord</i>	Word	%QW8
2	 <i>i_wStatusWord</i>	Word	%IW8
3	 <i>q_siModeOfOp</i>	SInt	%QB18
4	 <i>i_siModeOfOpDsp</i>	SInt	%IB18
5	 <i>i_wCmd</i>	Byte	%IB0
6	 <i>i_iSubIdx</i>	Byte	%IB1
7	 <i>i_uiIndex</i>	UInt	%IW2
8	 <i>i_wDataB0</i>	Byte	%IB4
9	 <i>i_wDataB1</i>	Byte	%IB5
10	 <i>i_wDataB2</i>	Byte	%IB6
11	 <i>i_wDataB3</i>	Byte	%IB7
12	 <i>q_wCmd</i>	Byte	%QB0
13	 <i>q_iSubIdx</i>	Byte	%QB1
14	 <i>q_uiIndex</i>	UInt	%QW2
15	 <i>q_wDataB0</i>	Byte	%QB4
16	 <i>q_wDataB1</i>	Byte	%QB5
17	 <i>q_wDataB2</i>	Byte	%QB6
18	 <i>q_wDataB3</i>	Byte	%QB7
19	 <i>q_iTargetVelocity</i>	Int	%QW19
20	 <i>i_iActualVelocity</i>	Int	%IW19

Figure 9: PROFINET® Velocity mode mapping example -Tag definition



In a DB define the following variables (Figure 10):

Name	Data Type
<i>axis</i>	st_Axis
<i>objAccess</i>	st_ObjAccess

	Name	Data type
1	Static	
2	axis	"st_Axis"
3	objAccess	"st_ObjAccess"

Figure 10: PROFInet® Velocity mode mapping example - DB definition

In an OB call the FB\_PROFInetMap (Figure 11)

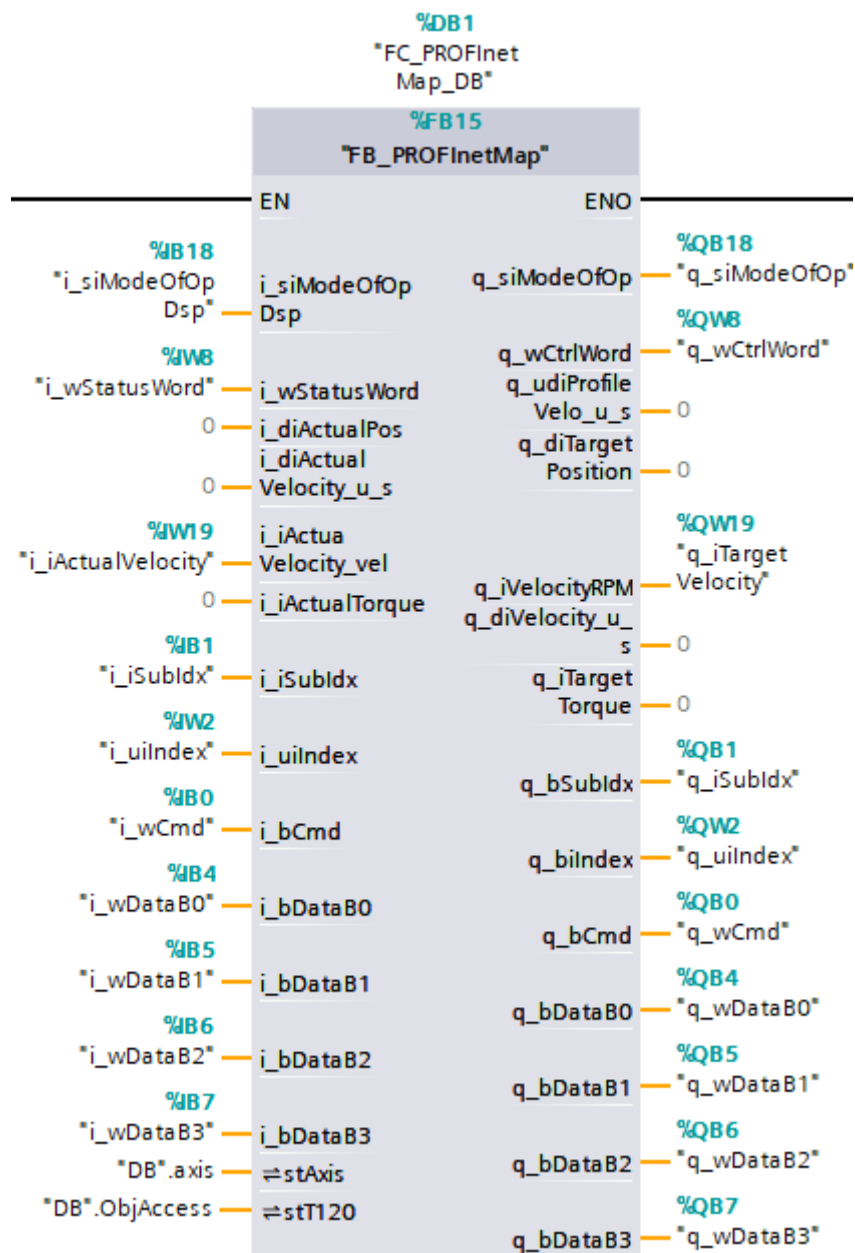
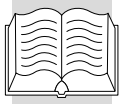


Figure 11: Velocity mode mapping example - PROFInet® map



### 5.2.2.2 PROFibus®

This outlines the configuration rules and considerations for setting up communication between a PLC and AxiaVert inverter drives using PROFibus® communication.


Device overview				
	...	Module	Rack	Slot
		Slave_1	0	0
		Object Access_1	0	1
		Process Data_1	0	2
		Process Data_2	0	3
			0	4
			0	5
			0	6

Figure 12: Example of PROFibus® map

Restrictions for user-defined configuration settings:

- Maximum **6 submodules** can be selected in total
- Maximum **24 bytes (12 words)** of data transfer capacity
- **Object Access:**
  - Has special restriction - only allowed in communication slot 1
  - is **mandatory**
- **Control Word** (*obj. 0x6040*), **Mode of Operation** (*obj. 0x6060*), **Status Word** (*obj. 0x6041*), **Mode of Operation Display** (*obj. 0x6061*) **must** be mapped in at least one telegram
- **Process Data 1** (first plugged process data module) has fixed assignment. It consists of two In/Out objects:
  - OUT: Control Word in word 0 (*obj. 0x6040*)
  - OUT: Frequency set point N2 in word 1 (*obj. 0x21F1/1*)
  - IN: Status Word in word 0 (*obj. 6041*)
  - IN: Actual frequency N2 (*obj. 21F1/3*)

It is only necessary to link the cyclic data relevant for the functions that the AxisMgr is supposed to use. AxisMgr will NOT report any error if any mandatory link is missing.

it is possible that informations linked to the telegram in the PLC need a corresponding mapping made on the inverter side manually (i.e.: via Axia Manager), see the [Velocity mapping example](#).



## Velocity mapping example

This provides a practical example of configuring an Axia drive for **Velocity Mode** operation (Mode of Operation = 2) using both Object Access and Process Data. The first mapped process data handles the control interface, while the second mapped Process Data handles the velocity-specific data:

→ Process Data\_1: core control/status communication

- Control Word
- Status Word

→ Process Data\_2: velocity data

- v/target velocity
- Mode of Operation
- v/velocity actual value
- Mode of Operation Display

In the Device overview of the AxiaVert drive add following telegrams ([Figure 13](#)):

Telegram	I address	Q address
Object Access_1	50 .. 57	50 .. 57
Process Data_1	60 .. 63	60 .. 63
Process Data_2	64 .. 67	64 .. 67

Device overview							
	...	Module	Rack	Slot	I address	Q address	Type
		Slave_1	0	0			AxiaVert Single Axis
		Object Access_1	0	1	50...57	50...57	Object Access
		Process Data_1	0	2	60...63	60...63	Process Data
		Process Data_2	0	3	64...67	64...67	Process Data
			0	4			
			0	5			
			0	6			

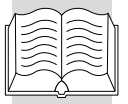
Figure 13: PROFibus® Velocity mode mapping example – Drive

In AxiaManager set the following drive parameters ([Figure 14](#)):

Object	SubIdx	Name	Value
0x2200	1..4	Control Mode	0x03: State Machine
0x25E1		Mode of Operation Source	0x039160A: PD1 OUT Byte 2 Int 8
0x25E8		VL Target Velocity Source	0x0391618: PD1 OUT Word 0 Int 16
0x3915	3	PD1 IN Byte 2	0x0606100: Mode of Operat. Display 6061:00
0x3915	9	PD1 IN Word 0	0x0004400: Actual Velocity 6044:00

Object	SubIdx	Name	Unit	Type	Value
0x2200	1	Control Mode		enum	0x00000003: State Machine
0x2200	2	Control Mode		enum	0x00000003: State Machine
0x2200	3	Control Mode		enum	0x00000003: State Machine
0x2200	4	Control Mode		enum	0x00000003: State Machine
0x25E1		Mode of Operation Source		enum	0x0039160A: PD1 OUT Byte 2 Int8
0x25E8		VL Target Velocity Source		enum	0x00391618: PD1 OUT Word 0 Int16
0x3915	3	PD1 IN Byte 2		enum	0x00606100: Mode Of Operat. Display 6061:00
0x3915	9	PD1 IN Word 0		enum	0x00604400: Actual Velocity 6044:00

Figure 14: PROFibus® Velocity mode mapping example - AxiaManager configuration



Define the following PLC tags (Figure 15):

Name	Data Type	Address	Description
<i>q_wCtrlWordPBUS</i>	Word	%QW60	Control Word
<i>i_wStatusWordPBUS</i>	Word	%IW60	Status Word
<i>q_siModeOfOpPBUS</i>	SInt	%QB66	Mode of Operation
<i>i_siModeOfOpPBUS</i>	SInt	%IB66	Mode of Operation Display
<i>q_iTargetVelocityPBUS</i>	Int	%QW64	v/ Target Velocity
<i>i_iActualVelocityPBUS</i>	Int	%IW64	v/ Actual Velocity
<i>i_wCmdPBUS</i>	Byte	%IB50	Input values of Object Access Channel
<i>i_iSubIdxPBUS</i>	Byte	%IB51	
<i>i_uiIndexPBUS</i>	UInt	%IW52	
<i>i_wDataB0PBUS</i>	Byte	%IB54	
<i>i_wDataB1PBUS</i>	Byte	%IB55	
<i>i_wDataB2PBUS</i>	Byte	%IB56	
<i>i_wDataB3PBUS</i>	Byte	%IB57	
<i>q_wCmdPBUS</i>	Byte	%QB50	Output values for Object Access Channel
<i>q_iSubIdxPBUS</i>	Byte	%QB51	
<i>q_uiIndexPBUS</i>	UInt	%QW52	
<i>q_wDataB0PBUS</i>	Byte	%QB54	
<i>q_wDataB1PBUS</i>	Byte	%QB55	
<i>q_wDataB2PBUS</i>	Byte	%QB56	
<i>q_wDataB3PBUS</i>	Byte	%QB57	





















	Name	Data type	Address
1	 <i>q_wCtrlWordPBUS</i>	Word	%QW60
2	 <i>i_wStatusWordPBUS</i>	Word	%IW60
3	 <i>q_siModeOfOpPBUS</i>	SInt	%QB66
4	 <i>i_siModeOfOpPBUS</i>	SInt	%IB66
5	 <i>q_iTargetVelocityPBUS</i>	Int	%QW64
6	 <i>i_iActualVelocityPBUS</i>	Int	%IW64
7	 <i>i_wCmdPBUS</i>	Byte	%IB50
8	 <i>i_iSubIdxPBUS</i>	Byte	%IB51
9	 <i>i_uiIndexPBUS</i>	UInt	%IW52
10	 <i>i_wDataB0PBUS</i>	Byte	%IB54
11	 <i>i_wDataB1PBUS</i>	Byte	%IB55
12	 <i>i_wDataB2PBUS</i>	Byte	%IB56
13	 <i>i_wDataB3PBUS</i>	Byte	%IB57
14	 <i>q_wCmdPBUS</i>	Byte	%QB50
15	 <i>q_iSubIdxPBUS</i>	Byte	%QB51
16	 <i>q_uiIndexPBUS</i>	UInt	%QW52
17	 <i>q_wDataB0PBUS</i>	Byte	%QB54
18	 <i>q_wDataB1PBUS</i>	Byte	%QB55
19	 <i>q_wDataB2PBUS</i>	Byte	%QB56
20	 <i>q_wDataB3PBUS</i>	Byte	%QB57

Figure 15: PROFibus® Velocity mode mapping example -Tag definition



In a DB define the following variables ([Figure 16](#)):

Name	Data Type
<i>axis</i>	st_Axis
<i>objAccess</i>	st_ObjAccess

	Name	Data type
1	Static	
2	axis	"st_Axis"
3	objAccess	"st_ObjAccess"

Figure 16: PROFibus® Velocity mode mapping example - DB definition

In an OB call the FB\_PROFibusMap ([Figure 17](#))

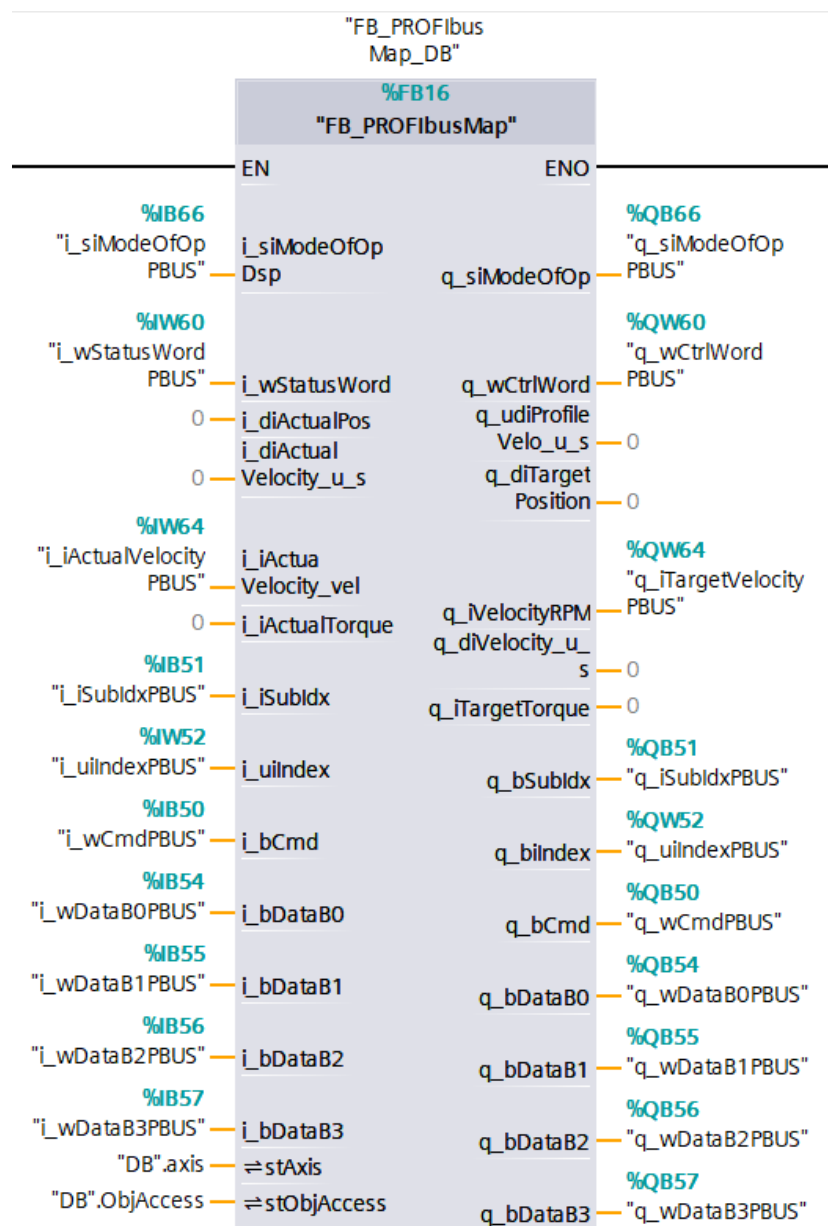


Figure 17: Velocity mode mapping example - PROFibus® map





## 5.3 Overview of AxisMgr Functions

AxisMgr manages functions responsible of axis control, movement and diagnostic.

Functions can be triggered via the assigned command selections (i.e.: *i\_usiCmd*) and *i\_xLaunchCmd* or via a specific Boolean input. The control sequence and the global result is established by the AxisMgr.

Normally functions need valid input data that can be usually found in the *i\_stXXX* structured input. Refer to the specific structure chapter for further information.

Supported functions are:

Function	<i>i_iCmd</i> / Bool Input	Description
Power ( <a href="#">5.3.5</a> )	<i>i_xPower</i>	It power on and off the drive. If this bit is set to TRUE, the axis is put in <i>Standstill</i> state, otherwise the axis is put in <i>Disabled</i> state. During the poweroff phase, <i>i_usiShutdownMode</i> input data is relevant
Hardware Error ( <a href="#">5.3.3</a> )	N/A	Always active function. It's task is to monitor drive errors.
Reset ( <a href="#">5.3.9</a> )	<i>i_xReset</i>	It reset the error state, either user or FB error or hardware error. The reset function is activated on the rising edge of <i>i_xReset</i> .
Object Access management ( <a href="#">5.3.12</a> and <a href="#">5.3.13</a> )	N/A	Support utilities designed to manage and coordinate access requests to the Object Access. These functions are called by other functions, or by external access
QuickStop ( <a href="#">5.2.1.2</a> )	<i>i_xQuickStop</i>	It performs immediately the emergency ramp of the axis. <i>i_rQuickStopDeceleration</i> input data is relevant.
Stop ( <a href="#">5.3.14</a> )	20	It stops any ongoing movement putting the axis in <i>Stopping</i> state. <i>i_rStopDeceleration</i> input data is relevant. End state depends on the actual Mode Of Operation. It isn't implemented in a specific FB, the stop procedure is implemented in <i>cleaning</i> phase of each motion control function
Homing ( <a href="#">5.3.4</a> )	40	It performs the homing procedure provided by the drive (CiA402 Mode Of Operation 6) putting the axis in <i>Homing</i> state. <i>i_stHoming</i> input data is relevant. A successful Homing ends in <i>Standstill</i> state.
Profile Position ( <a href="#">5.3.6</a> )	60	The axis moves towards the specified target position (either absolute or relative), with the specified dynamic settings. It puts the axis in <i>Position</i> state. <i>i_stProfilePos</i> input data is relevant. Mode Of Operation 1 is used
Profile Torque ( <a href="#">5.3.7</a> )	110	The axis speeds up until it reaches the specified target torque, with the specified torque slope. It puts the axis in <i>Torque</i> state. <i>i_stTorque</i> input data is relevant. Mode Of Operation 4 is used



Profile Velocity ( <a href="#">5.3.8</a> )	50	The axis moves endless with the specified dynamic settings. It puts the axis in <i>Velocity</i> state. <i>i_stProfileVelocity</i> input data is relevant. Mode Of Operation 3 is used.
Velocity Mode ( <a href="#">5.3.11</a> )	51	The axis moves endless with the specified dynamic settings. It puts the axis in <i>Velocity</i> state. <i>i_stVelocity</i> input data is relevant. Mode Of Operation 2 is used.
Touch Probe ( <a href="#">0</a> )	See description	As long as it is enabled (i.e.: <i>i_stTouchProbe.stTPx.xTouchProbeEnable</i> = <i>TRUE</i> ), AxisMgr reads out the related TouchProbe channel information.

#### NOTE

Before changing the Mode of Operation, the system writes all relevant starting parameters (e.g. acceleration, deceleration, target velocity). Only after all parameters are properly configured the system change the Mode of Operation value

Most functions within the system are built upon the FB\_ETrigA (see [FB\\_ETrigA](#)) state machine framework, providing standardized execution control and consistent operational behavior. Selected functions operate independently of the FB\_ETrigA architecture:

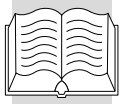
- Touch probe data reading operations
- Object Access access request management
- Certain real-time monitoring functions

These exceptions typically involve continuous operations or specialized communication protocols that require different execution patterns than the standard state machine provides.

#### NOTE

In addition to the flow chart, functions can implement a “background” routine that is always running, regardless the actual status of the function. Examples:

- TouchProbe function does only work in background (when enabled), it is not affected by state changes and/or external aborts.
- HardwareError function is always active



### 5.3.1 FB\_ETrigA

Motion Control functions are based on FB\_ETrigA state machine.

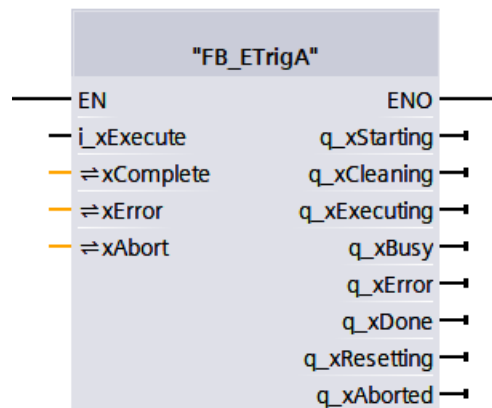


Figure 18: FB\_ETrigA

FB\_ETrigA is a 8-state state machine-based function block designed for TIA Portal that implements a standardized automation sequence with error handling and abort capabilities. The function block follows a typical industrial automation pattern with multiple operational states and comprehensive status reporting.

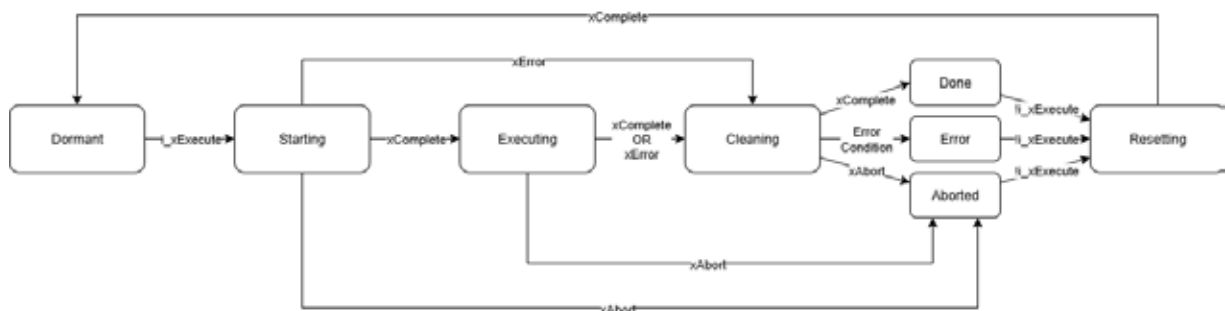


Figure 19: FB\_ETrigA State Machine

Input	Data Type	Default	Description
<i>i_xExecute</i>	Bool	false	Execute command to start the sequence

Output	Data Type	Default	Description
<i>q_xStarting</i>	Bool	false	Indicates the function block is in STARTING state
<i>q_xCleaning</i>	Bool	false	Indicates the function block is in CLEANING state
<i>q_xExecuting</i>	Bool	false	Indicates the function block is in EXECUTING state
<i>q_xBusy</i>	Bool	false	Indicates the function block is busy (active operation)
<i>q_xError</i>	Bool	false	Indicates an error condition has occurred
<i>q_xDone</i>	Bool	false	Indicates successful completion of the sequence
<i>q_xResetting</i>	Bool	false	Indicates the function block is in RESETTING state
<i>q_xAborted</i>	Bool	false	Indicates the sequence was aborted



InOut	Data Type	Default	Description
<i>xComplete</i>	Bool	false	Completion signal from external process
<i>xError</i>	Bool	false	Error signal from external process
<i>xAbort</i>	Bool	false	Abort signal from external process

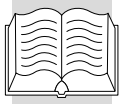
### 5.3.1.1 State Machine Description

The function block implements an 8-state state machine with the following states:

State	Value	Description
<i>DORMANT</i>	0	Idle state, ready to accept new execute command
<i>STARTING</i>	1	Initialization/startup phase
<i>EXECUTING</i>	2	Main execution phase
<i>CLEANING</i>	3	Cleanup/finalization phase
<i>ERROR</i>	4	Error state
<i>ABORTED</i>	5	Aborted state
<i>DONE</i>	6	Successfully completed state
<i>RESETTING</i>	7	Reset phase, returning to DORMANT

### 5.3.1.2 State transition

- DORMANT → STARTING: rising edge of *i\_xExecute*
- STARTING → EXECUTING: transition to EXECUTING state when *xComplete* = TRUE
- STARTING → CLEANING: transition to CLEANING state when *xError* = TRUE
- STARTING → ABORTED: transition to ABORTED state when *xAbsort* = TRUE
- EXECUTING → CLEANING: transition to CLEANING state when *xComplete* = TRUE or *xError* = TRUE
- EXECUTING → ABORTED: transition to ABORTED state when *xAbsort* = TRUE
- CLEANING → ERROR: transition to ERROR state then when *xError* = TRUE
- CLEANING → ABORTED: transition to ABORTED state when *xAbsort* = TRUE
- CLEANING → DONE: transition to DONE state when *xComplete* = TRUE
- ERROR/ABORTED/DONE → RESETTING: transition to RESETTING state when *i\_xExecute* = FALSE
- RESETTING → DORMANT: transition to DORMANT state when *xComplete* = TRUE



### 5.3.1.3 Operational Behaviour

#### Normal Operation Sequence

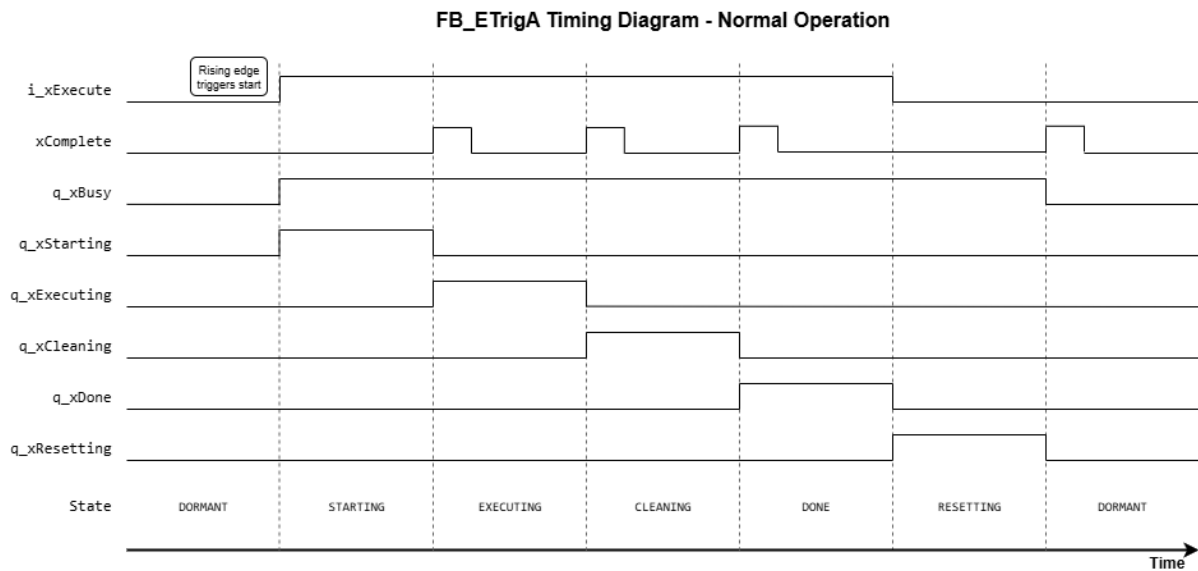
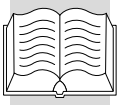


Figure 20: FB\_ETrigA Normal Operation Sequence

1. **DORMANT:** Function block waits for a trigger on *i\_xExecute* to be set to move to STARTING
2. **STARTING:** Initialization phase, sets *q\_xStarting* and *q\_xBusy*. Waits for *xComplete* to move to EXECUTING
3. **EXECUTING:** Main operation phase, sets *q\_xExecuting*. Waits for *xComplete* to move to CLEANING
4. **CLEANING:** Cleanup phase, sets *q\_xCleaning*. Waits for *xComplete* to move to DONE
5. **DONE:** Completion state, sets *q\_xDone*. Waits for the falling of *i\_xExecute* to move to RESETTING, at the end of the phase *q\_xBusy* is resetted
6. **RESETTING:** Reset phase when *i\_xExecute* goes FALSE. Waits for *xComplete* to move to DORMANT
7. **DORMANT:** Returns to idle state



## Error Handling

If  $xError$  is set during operation, the function block transitions to CLEANING and then to ERROR state.

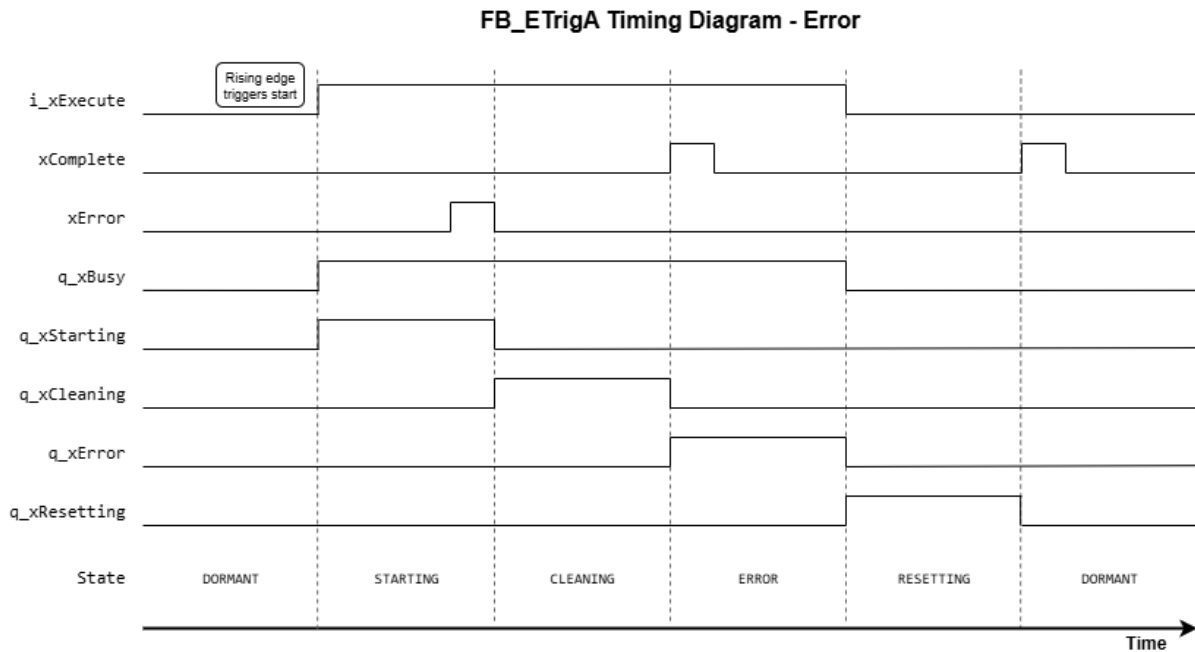


Figure 21: FB\_ETrigA Error Handling

While in ERROR state, sets  $q_xError$  and waits for the falling of  $i_xExecute$  to move to RESETTING.

## Abort Handling

If  $xAbort$  is set during operation, the function block transitions to ABORTED state.

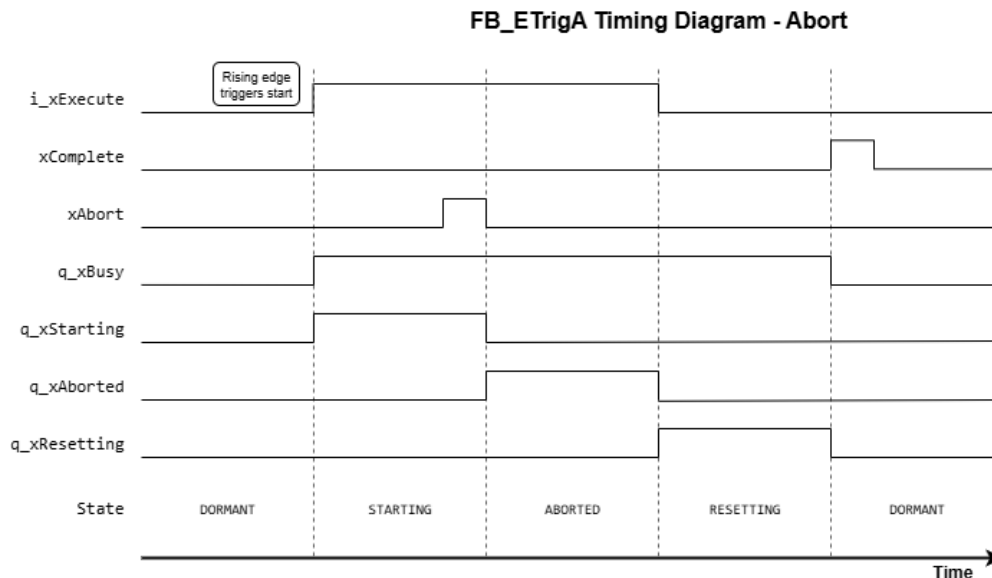
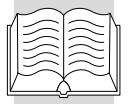


Figure 22: FB\_ETrigA Abort Handling

While in ABORTED state, sets  $q_xAborted$  and waits for the falling of  $i_xExecute$  to move to RESETTING.



## 5.3.2 Input/Output data

### 5.3.2.1 *i\_stXXX* structure

AxisMgr provide the *i\_stXXX* data structure, which comprehends all the relevant datas for function to be executed.

AxisMgr validates the data values before to execute the function, otherwise an error is reported. Moreover, some input data are constantly evaluated and, depending on the function, changes may have immediately effect.

The *i\_stXXX* structure, as well as FB's output data, contains information (in addition to all the others) that are related to position or speed. Those figures are expressed in **units (u, u/s, u/s<sup>2</sup>)** and are probably affected by conversion factors before to be sent/received by the drive.

### 5.3.2.2 Values conversion factor, user units definition

Information that are sent/receive to/from the drive are probably converted. Moreover, the drive itself features its internal conversion factor, that is applied for the final calculation of the motor turns. The combination of both factors (controller side and drive side) results in the final definition of the **user unit (u)**.

Informations to/from the real drive are scaled according to what provided to *i\_rFactorPos* and *i\_rFactorVel*.

- *i\_rFactorPos*: it is expressed in *increments/user unit* and defines the scaling factor applied for the **pos. units** informations to/from the drive. It is used for those functions that works in combination with the position controlled mode of operation (e.g.: Profile Position, Profile Velocity, Homing, ecc...). The final calculation of the motor turns is also affected by the drive's *obj. 0x6092 Feed Constant* and *obj. 0x6091 Gear ratio* that are applied afterwards the drive increments.  
*i\_rFactorPos* defines how may increments are to be considered for 1 u displacement. The presence of a gear box should be also considered in order to establish the value to set in *i\_rFactorPos*. It is always considered as ABSOLUTE.

The following relations result:

- Starting from motor turns, a possible calculation of Position in **u** is as follows:

$$\left( \text{Act. encoder turns} \times \frac{\frac{\text{Act. drive increments (e.g.: stAxis.i_diActualPos) [incr]}{\text{obj. 0x6091.2}}}{\text{obj. 0x6091.1}} \times \frac{\text{(Feed)}}{\frac{\text{obj. 0x6092.1}}{\text{obj. 0x6092.2}}} \right) - \text{Home Offset [incr]}$$

$$\text{Act. Position (e.g.: q_rActualPosition_u) [u]} = \frac{\text{Act. drive increments [incr]}}{i_rFactorPos [incr/u]}$$

- Which also results:

$$\text{Target drive increments (e.g.: stAxis.q_diTargetPosition) [incr]} = \text{Target value (e.g.: i_stProfilePosition.rTargetPos) [u]} \times i_rFactor\_pos [incr/u]$$



$$\begin{aligned}
 & \text{Target motor turns} = \frac{\text{obj.0x6092.2}}{\text{obj.0x6092.1}} \times \frac{\text{obj.0x6091.1}}{\text{obj.0x6091.2}} \\
 \text{Target drive increments [incr]} & \times \frac{((\text{Driving}) \text{ Shaft Revolutions})}{(\text{Feed})} \times \frac{(\text{Motor Shaft Revolutions})}{(\text{Driving Shaft Revolutions})}
 \end{aligned}$$

- *i\_rFactorVel*: it is expressed in *VL Units/user unit* and defines the scaling factor applied for the **vel. units** informations to/from the drive. It is used for those functions that works specifically with the speed controller (e.g.: Velocity Mode). The final calculation of the motor RPM can also be affected by the drive's *obj. 0x604C VL Dimension* that is applied afterwards the drive's VL Units.  
*i\_rFactorVel* defines how may *VL Units* are to be considered for 1 u speed. It is always considered as ABSOLUTE.



- ➔ The user can eventually decide to leave *obj. 0x6091* and *obj. 0x6092* to their default values, managing the *i\_rFactorPos* only. Instead it is possible to set *i\_rFactorPos* to 1, which means that the only active frame of reference is on the drive side.
- ➔ The user should consider that the information to/from the drive (in *increments*), are integers value. Setting of *obj. 0x6092* to a low value results in big loss of information when converted in a floating number. Depending on the application and the axis travel range, *obj. 0x6092* should be set as big as possible.
- ➔ *i\_rFactorPos* can be changed during the execution of the AxisMgr. Depending on the running function, the target vaues are immediately recalculated and updated the information to/from the drive. This can be useful when the linear travel ratio of the application changes during the movement (e.g.: dynamic diameter recalculation for winding applications).
- ➔ *i\_rFactorVel* can be changed during the execution of the AxisMgr. Depending on the running function, the target vaues are immediately recalculated and updated the information to/from the drive.





### 5.3.3 Hardware error (FB\_MCHardwareError)

This function block is designed to monitor AxiaVert drive errors.

This function is always active, and it's executed in background.

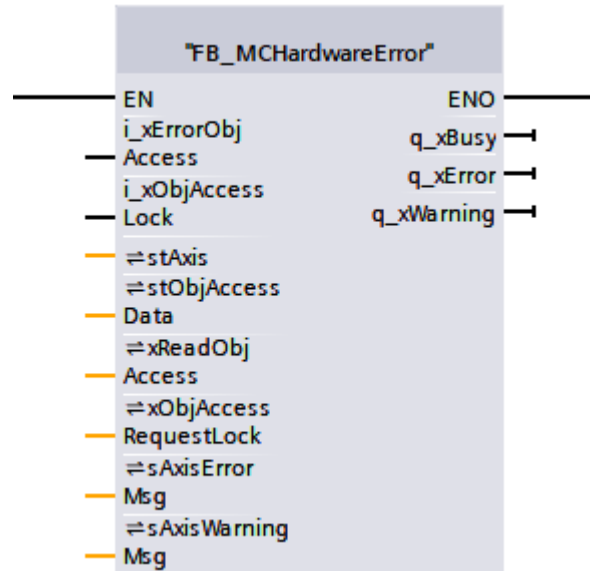


Figure 23: FB\_MCHardwareError

Input	Data Type	Default	Description
<i>i_xErrorObjAccess</i>	Bool	false	Indicate an error on Object Access
<i>i_xObjAccessLock</i>	Bool	false	Usage of Object Access is authorized

Output	Data Type	Default	Description
<i>q_xBusy</i>	Bool	false	Indicates the function block is busy (active operation)
<i>q_xError</i>	Bool	false	Indicate a drive error (4 <sup>th</sup> bit of the Status Word set)
<i>q_xWarning</i>	Bool	false	Indicate a drive warning (8 <sup>th</sup> bit of the Status Word set)

InOut	Data Type	Default	Description
<i>stAxis</i>	st_Axis		Axis I/O parameters
<i>stObjAccessData</i>	st_ObjAccessData		Data for Object Access
<i>xReadObjAccess</i>	Bool	false	Read request for Object Access. It is cleared when the read is completed, or an error occurs ( <i>i_xErrorObjAccess</i> is set)
<i>xObjAccessRequestLock</i>	Bool	false	Request the authorization to use Object Access for read
<i>sAxisErrorMsg</i>	String	''	Axis error description
<i>sAxisWarningMsg</i>	String	''	Axis warnings description



---

### Starting

N/A

### Execution

Continuously monitoring of the Fault bit (4<sup>th</sup> bit) and Warning bit (8<sup>th</sup> bit) of the Status Word.

In case of a fault *q\_xError* is set, object *0x4010/2 – Fault Code* is read and translated into a human readable string in *sAxisErrorMsg*.

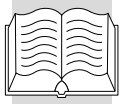
In case of a warning *q\_xWarning* is set, objects *0x4015/1..5* are read and translated into a human readable string in *sAxisWarningMsg*.

### Clearing

N/A

### Error Conditions

N/A



### 5.3.4 Homing function (FB\_MCHoming)

Homing function triggers the drive's homing procedure (CiA DS402 Mode Of Operation 6) and puts the AxisMgr in *Homing* state. *i\_stHoming* input data is relevant. A successful Homing ends in *Standstill* state.

As soon as the functions has been triggered by means of *i\_usiCmd* = 40 and a rising edge given to *i\_xLaunchCmd*, AxisMgr requests the Homing mode of operation.

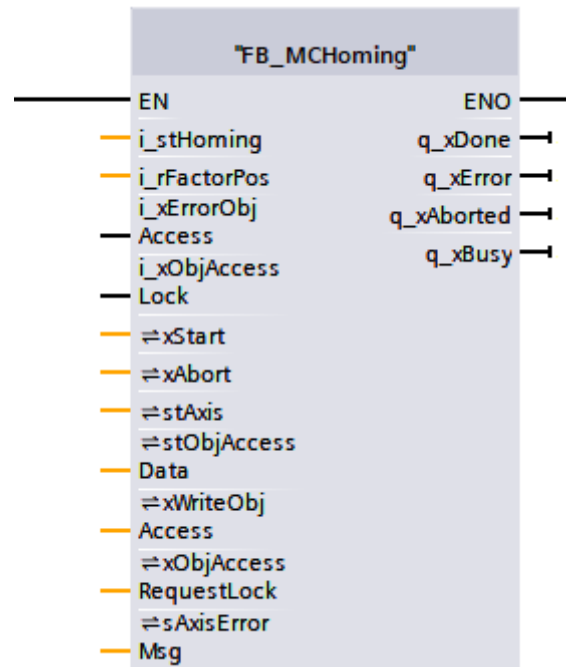


Figure 24: FB\_MCHoming

Input	Data Type	Default	Description
<i>i_stHoming</i>	st_Homing		Homing parameters. See <a href="#">st_Homing</a>
<i>i_rFactorPos</i>	Real	0.0	Scale factor
<i>i_xErrorObjAccess</i>	Bool	false	Indicate an error on Object Access
<i>i_xObjAccessLock</i>	Bool	false	Usage of Object Access is authorized

Output	Data Type	Default	Description
<i>q_xDone</i>	Bool	false	Homing completed or, in case of a stop command ( <i>i_usiCmd</i> = 20) indicates that the stopping procedure is completed
<i>q_xError</i>	Bool	false	Error during Homing procedure
<i>q_xAborted</i>	Bool	false	Homing procedure aborted
<i>q_xBusy</i>	Bool	false	Indicates the function block is busy



InOut	Data Type	Default	Description
<i>xStart</i>	Bool	False	Rising edge: start procedure. It is cleared at the end of the Homing procedure or, in case is cleared externally, it stops immediately the Homing procedure
<i>xAbort</i>	Bool	false	TRUE: abort Homing procedure
<i>stAxis</i>	st_Axis		Axis I/O parameters
<i>stObjAccessData</i>	st_ObjAccessData		Data for Object Access
<i>xWriteObjAccess</i>	Bool	false	Write request for Object Access. It is cleared when the write is completed, or an error occurs ( <i>i_xErrorObjAccess</i> is set)
<i>xObjAccessRequestLock</i>	Bool	false	Request the authorization to use Object Access
<i>sAxisErrorMsg</i>	String	''	Axis error description

### Starting

The Modes of Operation is set to 6 (Homing Mode) and the following homing parameters are written to their respective addresses:

Address	Parameter	Source
0x607C/00	Home Offset Position	$i\_stHoming.rFinalPosition * i\_rFactorPos$
0x6098/00	Homing Method	$i\_stHoming.usiMode$
0x6099/01	Search Cam Velocity	$i\_stHoming.rSearchCamVelocity * i\_rFactorPos$
0x6099/02	Leave Cam Velocity	$i\_stHoming.rLeaveCamVelocity * i\_rFactorPos$
0x609A/00	Homing Acceleration	$i\_stHoming.rAcceleration * i\_rFactorPos$

Once the parameter configuration is complete and the drive confirms Mode of Operation 6 is active, the homing sequence initiates by setting the Homing Operation Start bit (bit 5) and clearing the Halt bit (bit 9) in the Control Word.

The homing starting operation concludes when the Homing Attained bit (bit 13) in the Status Word becomes active.

### Execution

The homing procedure is considered successfully completed when homing Attained bit (bit 13) in the Status Word is set and Homing Error bit (bit 14) in the Status Word not. Upon meeting these conditions, *q\_xDone* output is activated.

The homing procedure will also terminate if a stop command is issued (i.e. by setting  $i\_usiCmd = 20$ ). In this case *q\_xDone* remains inactive since the homing was not completed successfully.

### Clearing

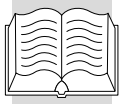
When homing completes successfully (*q\_xDone* = TRUE) Halt bit (bit 9) in the Control Word is set and no additional operations are required.

When homing terminates without success (error condition or manual stop) the stop ramp is activated. Parameter 0x6084/00 (Profile deceleration) is set to  $i\_stHoming.rAcceleration * i\_rFactorPos$  (limited between 1 and 2147483647) and Halt bit (bit 9) in the Control Word is set to initiate the stop ramp.

The deceleration phase concludes when either of the following Status Word conditions occurs:

- ➔ Target Reached bit (bit 11) is set, indicating controlled stop completion
- ➔ Operation Enabled bit (bit 3) becomes cleared, indicating drive has exited operational state

At the end of the deceleration phase, if no errors occur, *q\_xDone* is set to TRUE.



## Error Conditions

During the starting phase, errors are triggered under the following conditions:

- ➔ Object Access Error: when *i\_xErrorObjAccess* is active
- ➔ When the homing mode (*i\_stHoming.usiMode*) is not set to HOMING\_NOHOMING (0) and HOMING\_CURRENTPOSITION (35) the following parameter ranges are out of range:
  - *i\_stHoming.rAcceleration* \* *i\_rFactorPos* < 1.0 or > 2147483647.0
  - *i\_stHoming.rFinalPosition* \* *i\_rFactorPos* < -2147483648.0 or > 2147483647.0
  - *i\_stHoming.rSearchCamVelocity* \* *i\_rFactorPos* < 1.0 or > 2147483647.0
  - *i\_stHoming.rLeaveCamVelocity* \* *i\_rFactorPos* < 1.0 or > 2147483647.0
- ➔ Parameters are not successfully written within 2 seconds

During the execution phase, the Homing Error bit (bit 14) of the Status Word is continuously monitored and, when set, an error condition is generated.

### NOTE

- ➔ Unlike other functions, *i\_stHoming* is only evaluated at the beginning of the function. A value change, while the function is already running, won't have any effect.



### 5.3.5 Power (FB\_MCPower)

This function block is designed to power on and off AxiaVert drive.

This function is activated by rising and falling edges of *i\_xPower*.

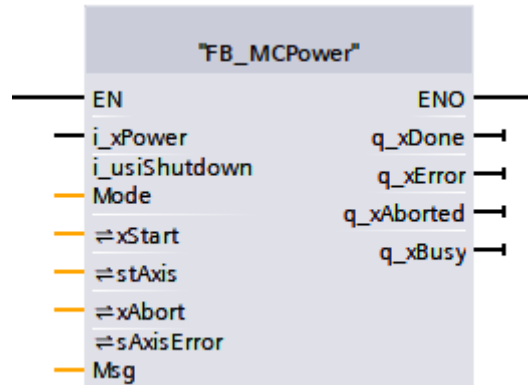
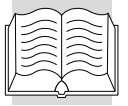


Figure 25: FB\_MCPower

Input	Data Type	Default	Description
<i>i_xPower</i>	Bool	false	TRUE: power on the drive, FALSE: power off the drive
<i>i_usiShutdownMode</i>	USint	0	Powering off mode

Output	Data Type	Default	Description
<i>q_xDone</i>	Bool	false	Power function completed successfully
<i>q_xError</i>	Bool	false	Error during Power procedure
<i>q_xAborted</i>	Bool	false	Power procedure aborted
<i>q_xBusy</i>	Bool	false	Indicates the function block is busy

InOut	Data Type	Default	Description
<i>xStart</i>	Bool	false	Rising edge: start procedure. It is cleared at the end of the powering procedure
<i>xAbort</i>	Bool	false	TRUE: abort Power procedure
<i>stAxis</i>	st_Axis		Axis I/O parameters
<i>sAxisErrorMsg</i>	String	''	Axis error description



## Starting

### Power On

Set Mode Of Operation to 2 and vl Target Velocity to 0.

An error occurs if Voltage enabled bit (5<sup>th</sup> bit) or Remote bit (10<sup>th</sup> bit) of the Status Word are false.

### Power Off

Check the status of *i\_usiShutdownMode*, if it is not equal to SHUTDOWN\_DISABLE\_OPERATION, SHUTDOWN\_DISABLE\_VOLTAGE or SHUTDOWN\_SHUTDOWN an error arise.

## Executing

### Power On

It performs the 3 steps sequence (6 → 7 → 15) with the Halt bit set.

- ➔ Step 1: Control Word configured with Enable voltage bit (2<sup>nd</sup> bit), Quick stop bit (3<sup>rd</sup> bit) and Halt bit (9<sup>th</sup> bit)
- ➔ Step 2: Sets the Switch On bit (bit 1) in the Control Word when all of the following Status Word conditions are met: Ready to Switch On bit (bit 1), Voltage Enabled bit (bit 5) and Quick Stop bit (bit 6)
- ➔ Step 3: Sets Enable operation bit (4<sup>nd</sup> bit) in the Control Word when Switched on bit (2<sup>nd</sup> bit) of the Status Word is set.

If the drive doesn't power up within 5 seconds, the function block will generate an error.

### Power Off

It performs the shutdown procedure. See [5.2.1.2 From Disabled to Standstill](#) for more details.

## Clearing

N/A

## Error Conditions

When powering on the drive, an error occurs if Voltage enabled bit (5<sup>th</sup> bit) or Remote bit (10<sup>th</sup> bit) are not set, or if the drive doesn't power up within 5 seconds.

When powering off the drive, an error occurs if *i\_usiShutdownMode* is not set properly.



### 5.3.6 Profile Position (FB\_MCProfilePosition)

Profile Position function triggers the drive's profile position mode (CiA DS402 Mode Of Operation 1) and puts the AxisMgr in *Positioning* state. *i\_stProfilePosition* input data is relevant. A successful profile position ends in *Standstill* state.

As soon as the functions has been triggered by means of *i\_usiCmd* = 60 and a rising edge given to *i\_xLaunchCmd*, AxisMgr requests the Profile Position mode of operation.

Depending on the positioning type (Absolute or Relative) the *i\_stProfilePos.rTargetPosition [u]* defines the absolute quota to reach (referred to the last home position) or the distance to cover.

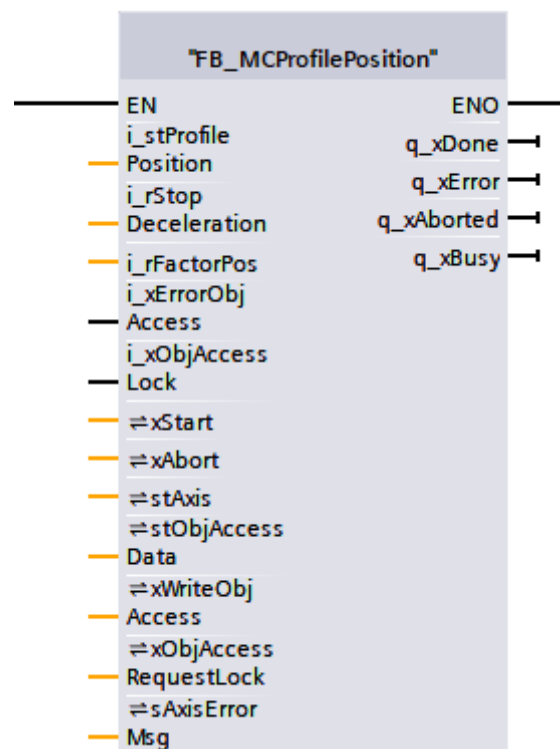
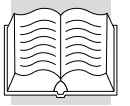


Figure 26: FB\_MCProfilePosition

Input	Data Type	Default	Description
<i>i_stProfilePosition</i>	st_ProfilePosition		Profile Position parameters. See <a href="#">st_ProfilePosition</a> <b>Errore. L'origine riferimento non è stata trovata.</b>
<i>i_rStopDeceleration</i>	Real	0.0	Deceleration ramp in case of a stop command ( <i>i_usiCmd</i> = 20) of the function. It is corrected with <i>i_rFactorPos</i>
<i>i_rFactorPos</i>	Real	0.0	Scaling factor
<i>i_xErrorObjAccess</i>	Bool	false	Indicate an error on Object Access
<i>i_xObjAccessLock</i>	Bool	false	Usage of Object Access is authorized

Output	Data Type	Default	Description
<i>q_xDone</i>	Bool	false	Profile Position function completed successfully
<i>q_xError</i>	Bool	false	Error during Profile Position procedure
<i>q_xAborted</i>	Bool	false	Profile Position procedure aborted
<i>q_xBusy</i>	Bool	false	Indicates the function block is busy







InOut	Data Type	Default	Description
<i>xStart</i>	Bool	False	Rising edge: start procedure. It is cleared at the end of the powering procedure
<i>xAbort</i>	Bool	false	TRUE: abort Power procedure
<i>stAxis</i>	st_Axis		Axis I/O parameters
<i>stObjAccessData</i>	st_ObjAccessData		Data for Object Access
<i>xWriteObjAccess</i>	Bool	false	Write request for Object Access. It is cleared when the write is completed, or an error occurs ( <i>i_xErrorObjAccess</i> is set)
<i>xObjAccessRequestLock</i>	Bool	false	Request the authorization to use Object Access
<i>sAxisErrorMsg</i>	String	''	Axis error description

### Starting

The Mode of Operation is set to 1 (Profile Position Mode), target position is set to *i\_stProfilePosition.rTargetPos*, target velocity is set to *i\_stProfilePosition.rTargetVel* and the following profile position parameters are written to their respective addresses:

Address	Parameter	Source
0x6067/00	Position window	<i>i_stProfilePosition.rPositionWIndow * i_rFactor_pos</i>
0x6068/00	Position window time	<i>i_stProfilePosition.uiPositionWIndowTime_ms</i>
0x6083/00	Profile acceleration	<i>i_stProfilePosition.rAcceleration * i_rFactor_pos</i>
0x6084/00	Profile deceleration	<i>i_stProfilePosition.rDeceleration * i_rFactor_pos</i>
0x6067/00	Position window	<i>i_stProfilePosition.rPositionWIndow * i_rFactor_pos</i>

Once the parameter configuration is complete and the drive confirms Mode of Operation 1 the starting phase terminate.

### Executing

This phase continuously updates Target Position and Target Velocity, and the following Control Word bits are mapped to their respective input parameters:

Bit	Description	Source
6	Change Set Immediately	<i>i_stProfilePosition.xChangeSetImmediately</i>
7	Absolute/Relative Mode	<i>i_stProfilePosition.xRelative</i>
9	Change on Set-Point	<i>i_stProfilePosition.xChangeOnSetPoint</i>

New Set-Point bit (bit 5) of the Control Word is activated until acknowledgment (Set-Point Acknowledge bit, bit 13, in the Status Word is set).

During execution, acceleration and deceleration parameters, *i\_stProfilePosition.rAcceleration* and *i\_stProfilePosition.rDeceleration*, can be modified. When these parameters change, a new trigger of bit 5 occurs to update the drive with the modified values.

The profile position procedure is considered successfully completed when Target reached bit (bit 11) in the Status Word is set and Set-point acknowledge bit (bit 13) in the Status Word not. Upon meeting these conditions, *q\_xDone* output is activated.

The profile position procedure will also terminate if a stop command is issued (i.e. by setting *i\_usiCmd* = 20). In this case *q\_xDone* remains inactive since the procedure was not completed successfully.



## Clearing

When profile position completes successfully ( $q\_xDone = \text{TRUE}$ ) Halt bit (bit 9) in the Control Word is set and no additional operations are required.

When profile position terminates without success (error condition or manual stop) the stop ramp is activated. Parameter 0x6084/00 (Profile deceleration) is set to  $i\_rStopDeceleration * i\_rFactorPos$  (limited between 1 and 2147483647) and Halt bit (bit 9) in the Control Word is set to initiate the stop ramp.

The deceleration phase concludes when either of the following Status Word conditions occurs:

- ➔ Target Reached bit (bit 11) is set, indicating controlled stop completion
- ➔ Operation Enabled bit (bit 3) becomes clear, indicating drive has exited operational state

At the end of the deceleration phase, if no errors occur,  $q\_xDone$  is set to TRUE.

## Error Conditions

During the starting phase and the execution phase, errors are triggered under the following conditions:

- ➔ Object Access Error: when  $i\_xErrorObjAccess$  is active
- ➔  $i\_stProfilePosition.xChangeSetImmediately$  and  $i\_stProfilePosition.xChangeOnSetPoint$  are set to TRUE
- ➔ The following parameter ranges are out of range:
  - $i\_stProfilePosition.rTargetPos * i\_rFactor\_pos < -2147483648.0$  or  $> 2147483647.0$
  - $i\_stProfilePosition.rTargetVel * i\_rFactor\_pos < 1.0$  or  $> 2147483647.0$
  - $i\_stProfilePosition.rAcceleration * i\_rFactor\_pos < 1.0$  or  $> 2147483647.0$
  - $i\_stProfilePosition.rFinalPosition * i\_rFactor\_pos < 1.0$  or  $> 2147483647.0$
- ➔ Parameters are not successfully written within 2 seconds (only for the starting phase)

### NOTE

- ➔ When using relative movement, change of target values won't have any effect. A new rising edge of  $i\_xLaunchCmd$  should be used to trigger a new positioning during the ongoing one.
- ➔ Updating value during an ongoing movement it is only possible once at each position reached. If more than one values should be applied at the next positioning, they must be updated together before the execution of AxisMgr.



### 5.3.7 Profile Torque (FB\_MCProfileTorque)

Profile Position function triggers the drive's profile torque mode (CiA DS402 Mode Of Operation 4) and puts the AxisMgr in *Torque* state. *i\_stProfileTorque* input data is relevant. The axis moves endlessly with the specified dynamic settings.

As soon as the functions has been triggered by means of *i\_usiCmd* = 100 and a rising edge given to *i\_xLaunchCmd*, AxisMgr requests the Profile Torque mode of operation.

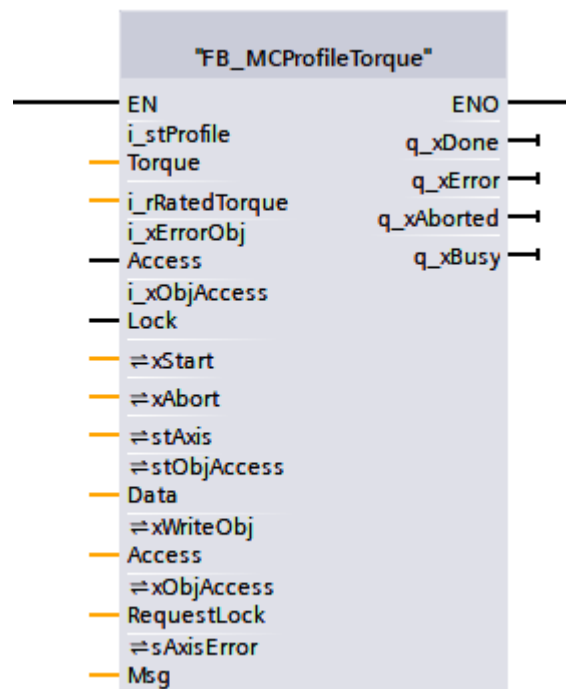
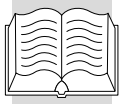


Figure 27: FB\_MCProfileTorque

Input	Data Type	Default	Description
<i>i_stProfileTorque</i>	st_ProfileTorque		Profile Torque parameters. See <a href="#">st_ProfileTorque</a> <b>Errore. L'origine riferimento non è stata trovata.</b>
<i>i_rRatedTorque</i>	Real	0.0	Motor rated torque. Used as scaling factor for <i>i_stProfileTorque.rTargetTorque</i> and <i>i_stProfileTorque.udiTorqueSlope</i>
<i>i_xErrorObjAccess</i>	Bool	false	Indicate an error on Object Access
<i>i_xObjAccessLock</i>	Bool	false	Usage of Object Access is authorized

Output	Data Type	Default	Description
<i>q_xDone</i>	Bool	False	This is an endless function. In case of a stop command ( <i>i_usiCmd</i> = 20) indicates that the stopping procedure is completed
<i>q_xError</i>	Bool	false	Error during Profile Torque procedure
<i>q_xAborted</i>	Bool	false	Profile Torque procedure aborted
<i>q_xBusy</i>	Bool	false	Indicates the function block is busy



InOut	Data Type	Default	Description
<i>xStart</i>	Bool	False	Rising edge: start procedure. It is cleared at the end of the powering procedure
<i>xAbort</i>	Bool	false	TRUE: abort Power procedure
<i>stAxis</i>	st_Axis		Axis I/O parameters
<i>stObjAccessData</i>	st_ObjAccessData		Data for Object Access
<i>xWriteObjAccess</i>	Bool	false	Write request for Object Access. It is cleared when the write is completed, or an error occurs ( <i>i_xErrorObjAccess</i> is set)
<i>xObjAccessRequestLock</i>	Bool	false	Request the authorization to use Object Access
<i>sAxisErrorMsg</i>	String	''	Axis error description

### Starting

The Mode of Operation is set to 4 (Profile Torque Mode), target torque is set to *i\_stProfileTorque.iTargetTorque*, and the following torque parameter is written to its respective address:

Address	Parameter	Source
0x6087/00	Torque slope	<i>i_stProfileTorque.udiTorqueSlope</i>

Once the parameter configuration is complete and the drive confirms Mode of Operation 4 the starting phase terminate.

### Executing

This phase continuously updates Target Torque and, during execution, torque slope parameter (*i\_stProfileTorque.udiTorqueSlope*), can be modified.

The profile torque procedure doesn't have a competition condition. The only way to terminate it is to issue a stop command (i.e. by setting *i\_usiCmd* = 20). *q\_xDone* remains inactive.

### Clearing

When profile torque receives a stop command Control Word is set to 0.

At the end of the phase, if no errors occur, *q\_xDone* is set to TRUE.

### Error Conditions

During the starting phase and the execution phase, errors are triggered under the following conditions:

- ➔ Object Access Error: when *i\_xErrorObjAccess* is active
- ➔ The following parameter ranges are out of range:
  - *i\_stProfileTorque.iTargetTorque* | *i\_rRatedTorque* \* 1000.0 < -32767 or > 32767
  - *i\_stProfileTorque.udiTorqueSlope* | *i\_rRatedTorque* \* 1000.0 < 0 or > 600000
- ➔ Parameters are not successfully written within 2 seconds (only for the starting phase)

#### NOTE

Rated Torque (parameter 0x2009.1) is read at startup by the *FB\_AxisMgr* and stored locally in the FB.



### 5.3.8 Profile Velocity (FB\_MCProfileVelocity)

Profile Velocity function triggers the drive's profile position mode (CiA DS402 Mode Of Operation 3) and puts the AxisMgr in *Velocity* state. *i\_stProfileVelocity* input data is relevant. The axis moves endlessly with the specified dynamic settings.

As soon as the functions has been triggered by means of *i\_usiCmd* = 50 and a rising edge given to *i\_xLaunchCmd*, AxisMgr requests the Profile Velocity mode of operation.

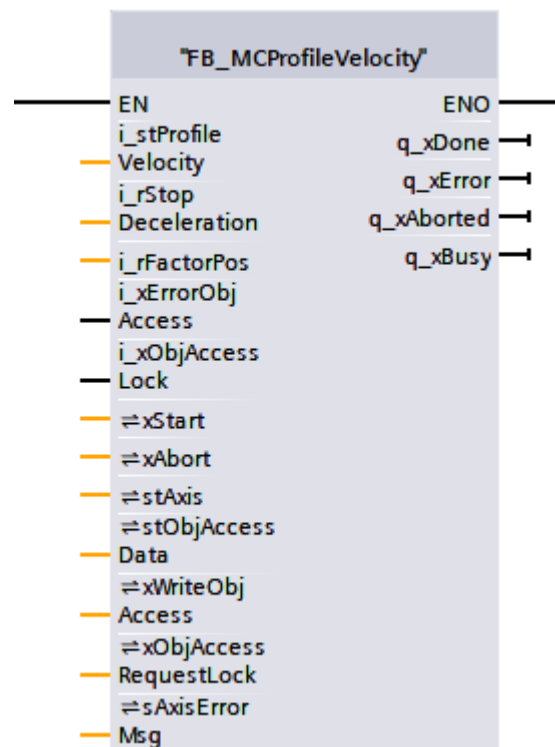
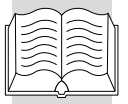


Figure 28: FB\_MCProfileVelocity

Input	Data Type	Default	Description
<i>i_stProfileVelocity</i>	st_ProfileVelocity		Profile Velocity parameters. See <a href="#">st_ProfileVelocity</a> <b>Errore. L'origine riferimento non è stata trovata.</b>
<i>i_rStopDeceleration</i>	Real	0.0	Deceleration ramp in case of a stop command ( <i>i_usiCmd</i> = 20) of the function. It is corrected with <i>i_rFactorPos</i>
<i>i_rFactorPos</i>	Real	0.0	Scaling factor
<i>i_xErrorObjAccess</i>	Bool	false	Indicate an error on Object Access
<i>i_xObjAccessLock</i>	Bool	false	Usage of Object Access is authorized

Output	Data Type	Default	Description
<i>q_xDone</i>	Bool	false	This is an endless function. In case of a stop command ( <i>i_usiCmd</i> = 20) indicates that the stopping procedure is completed
<i>q_xError</i>	Bool	false	Error during Profile Velocity procedure
<i>q_xAborted</i>	Bool	false	Profile Velocity procedure aborted
<i>q_xBusy</i>	Bool	false	Indicates the function block is busy



InOut	Data Type	Default	Description
<i>xStart</i>	Bool	False	Rising edge: start procedure.
<i>xAbort</i>	Bool	false	TRUE: abort Power procedure
<i>stAxis</i>	st_Axis		Axis I/O parameters
<i>stObjAccessData</i>	st_ObjAccessData		Data for Object Access
<i>xWriteObjAccess</i>	Bool	false	Write request for Object Access. It is cleared when the write is completed, or an error occurs ( <i>i_xErrorObjAccess</i> is set)
<i>xObjAccessRequestLock</i>	Bool	false	Request the authorization to use Object Access
<i>sAxisErrorMsg</i>	String	''	Axis error description

### Starting

The Mode of Operation is set to 3 (Profile Velocity Mode), target velocity is set to *i\_stProfileVelocity.rTargetVelocity* and the following profile velocity parameters are written to their respective addresses:

Address	Parameter	Source
0x6083/00	Profile acceleration	<i>i_stProfileVelocity.rAcceleration</i> * <i>i_rFactorPos</i>
0x6084/00	Profile deceleration	<i>i_stProfileVelocity.rDeceleration</i> * <i>i_rFactorPos</i>

Once the parameter configuration is complete and the drive confirms Mode of Operation 3 the starting phase terminate.

### Executing

This phase continuously updates Target Velocity and, during execution, acceleration (*i\_stProfileVelocity.rAcceleration*) and deceleration (*i\_stProfileVelocity.rDeceleration*) parameters can be modified.

The profile velocity procedure doesn't have a competition condition. The only way to terminate it is to issue a stop command (i.e. by setting *i\_usiCmd* = 20). *q\_xDone* remains inactive.

### Clearing

When profile velocity terminates (error condition or manual stop) the stop ramp is activated. Parameter 0x6084/00 (Profile deceleration) is set to *i\_rStopDeceleration* \* *i\_rFactorPos* (limited between 1 and 2147483647) and Halt bit (bit 9) in the Control Word is set to initiate the stop ramp.

The deceleration phase concludes when either of the following Status Word conditions occurs:

- ➔ Target Reached bit (bit 11) is set, indicating controlled stop completion
- ➔ Operation Enabled bit (bit 3) becomes clear, indicating drive has exited operational state

At the end of the deceleration phase, if no errors occur, *q\_xDone* is set to TRUE.

### Error Conditions

During the starting phase and the execution phase, errors are triggered under the following conditions:

- ➔ Object Access Error: when *i\_xErrorObjAccess* is active
- ➔ The following parameter ranges are out of range:
  - *i\_stProfileVelocity.rTargetVelocity* \* *i\_rFactorPos* < -2147483648.0 or > 2147483647.0
  - *i\_stProfileVelocity.rAcceleration* \* *i\_rFactorPos* < 1.0 or > 2147483647.0
  - *i\_stProfileVelocity.rDeceleration* \* *i\_rFactorPos* < 1.0 or > 2147483647.0
- ➔ Parameters are not successfully written within 2 seconds (only for the starting phase)

#### NOTE

A negative value set in *i\_stProfileVelocity.rTargetVelocity* (expressed in *pos. units/s*) results in a negative direction.



### 5.3.9 Reset (FB\_MCRreset)

This function block is designed to clear AxiaVert drive errors by cycling the Fault Reset bit (bit 8) in the drive's Control Word.

As soon as the functions has been triggered by a rising edge given to *i\_xReset*. This function run in background, and doesn't change the status of the AxisMgr (except for the error condition).

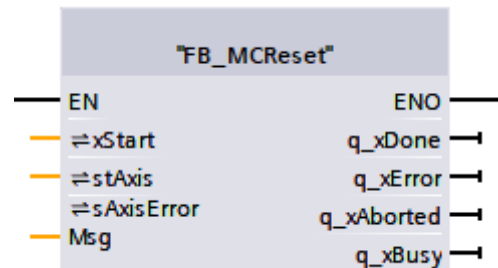


Figure 29: FB\_MCRreset

Output	Data Type	Default	Description
<i>q_xDone</i>	Bool	false	Reset completed
<i>q_xError</i>	Bool	false	Error during Reset procedure
<i>q_xAborted</i>	Bool	false	Reset procedure aborted
<i>q_xBusy</i>	Bool	false	Indicates the function block is busy

InOut	Data Type	Default	Description
<i>xStart</i>	Bool	False	Rising edge: start procedure. It is cleared at the end of the reset procedure
<i>stAxis</i>	st_Axis		Axis I/O parameters
<i>sAxisErrorMsg</i>	String	"	Axis error description

#### Starting

Performs a continuous toggle operation on the Fault Reset bit (bit 8) of the Control Word until Fault bit (4<sup>th</sup> bit) of the Status Word is cleared.

#### Executing

N/A

#### Clearing

N/A

#### Error Conditions

If the drive errors are not successfully cleared within the 2-second timeout period, the function block will generate an error condition to indicate that the fault reset operation has failed.

#### NOTE

- ➔ This function operates autonomously once initiated and cannot be interrupted (i.e. resetting *i\_xReset* input has no effect on the running function, as of setting *i\_usiCmd* = 20)
- ➔ The function will terminate under only two circumstances: Successful Completion (when the drive fault is successfully cleared) or Error Condition (when the 2-seconds timeout expires without fault clearance)





### 5.3.10 Touch Probe (FB\_MCTouchProbe)

TouchProbe function sets the special touch probe channels provided by the drive and gets the information from them. When enabled, the drive's TP channel detects the edges (both positive and negative) of the related physical digital input and stores the actual axis position in the lowest time, resulting in the most precise position capture. This is mostly needed when PLC cycle time is not fast enough to establish the exact axis position at a precise event.

Refer to the used drive documentation for further information about the touch probe function.

To activate this function *i\_stTouchProbe.stTP1.xTouchProbeEnable* or *i\_stTouchProbe.stTP2.xTouchProbeEnable* have to be set.



Figure 30: FB\_MCTouchProbe

Input	Data Type	Default	Description
<i>i_stTouchProbe</i>	st_TouchProbe		Touchprobe parameters. See <a href="#">st_TouchProbe</a>
<i>i_rFactorPos</i>	Real	0.0	Scale factor applied to <i>rPosEdgePositionValue</i> and <i>rNegEdgePositionValue</i>
<i>i_xErrorObjAccess</i>	Bool	false	Indicate an error on Object Access
<i>i_xObjAccessLockRead</i>	Bool	false	Usage of Object Access for reading is authorized
<i>i_xObjAccessLockWrite</i>	Bool	False	Usage of Object Access for writing is authorized

Output	Data Type	Default	Description
<i>q_stTouchProbeCh1Status</i>	st_TouchProbeChannelStatus		Touch probe channel 1 data. See <a href="#">st_TouchProbeChannelStatus</a>
<i>q_stTouchProbeCh2Status</i>	st_TouchProbeChannelStatus		Touch probe channel 2 data. See <a href="#">st_TouchProbeChannelStatus</a>



InOut	Data Type	Default	Description
<i>stObjAccessData</i>	st_ObjAccessData		Data for Object Access
<i>xReadObjAccess</i>	Bool	false	Read request for Object Access. It is cleared when the read is completed, or an error occurs ( <i>i_xErrorObjAccess</i> is set)
<i>xWriteObjAccess</i>	Bool	false	Write request for Object Access. It is cleared when the write is completed, or an error occurs ( <i>i_xErrorObjAccess</i> is set)
<i>xObjAccessRequestLockRead</i>	Bool	false	Request the authorization to use Object Access for read
<i>xObjAccessRequestLockWrite</i>	Bool	false	Request the authorization to use Object Access for write

This function operates independently of the *FB\_ETrigA* state machine architecture, providing direct touch probe data handling.

Touch probe data is read at regular intervals defined by *i\_stTouchProbe.tRefreshTouchProbeStatusTimer* and acquired touch probe information is stored in dedicated output structures *q\_stTouchProbeCh1Status* and *q\_stTouchProbeCh2Status*.

When fresh touch probe data becomes available for reading the corresponding ValueValid (*xNegEdgeValueValid* or *xPosEdgeValueValid*) bit is reset. After the data reading procedure successfully completes the ValueValid bit is set again to true.

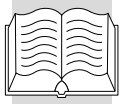
#### NOTE

##### TouchProbe function limitations, Real axis

- ➔ The function block performs continuous cyclic reading of touch probe data utilizing Object Access communication protocol.
- ➔ The reading phase itself introduces a delay in the overall cycle timing. The total cycle period equals: Timer Interval + Reading Phase Delay
- ➔ Read operation is delayed until Object Access becomes available
- ➔ Enabling of TP channel, as well as mode (either continuous or single shot), is carried out via Object Access as soon as *i\_stTouchProbe.stTPx.xTouchProbeEnable* is set to TRUE. If one (or more) TP event occurs during the TP enabling, it may not be detected. Be sure to enable the function in advance.
- ➔ Since TP channel inputs of the drive are usually not affected by any filter time, electromechanical sensor should not be used for this purpose. Additional signal edges (due to bouncing effects) may be detected by the drive, which overwrites Pos and Neg edge positions.
- ➔ TouchProbe function can only work with Position control configurations of the drive.



- ➔ Position informations are expressed in user-units. It means that the drive's increment conversion is already carried out by the function itself, as well as modulo calculation (for modulo axis).
- ➔ *q\_stTouchProbeStatus.stTPx.uiPosEdgeCounter* and *q\_stTouchProbeStatus.stTPx.uiNegEdgeCounter* are stored in the non-volatile memory of the drive. A reset/reboot of the PLC won't restore those values. Instead, a drive reset may be necessary.



### 5.3.11 Velocity mode (FB\_MCVelocity)

Velocity function triggers the drive's profile position mode (CiA DS402 Mode Of Operation 2) and puts the AxisMgr in *Velocity* state. *i\_stVelocity* input data is relevant. The axis moves endless with the specified dynamic settings.

As soon as the functions has been triggered by means of *i\_usiCmd* = 51 and a rising edge given to *i\_xLaunchCmd*, AxisMgr requests the Velocity mode of operation.

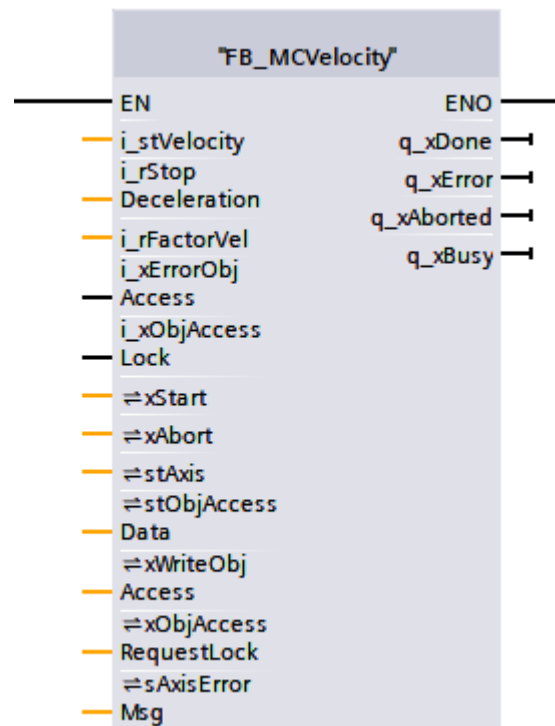


Figure 31: FB\_MCVelocity

Input	Data Type	Default	Description
<i>i_stVelocity</i>	st_Velocity		Velocity parameters. See <a href="#">st_Velocity</a> . <b>L'origine riferimento non è stata trovata.</b>
<i>i_rStopDeceleration</i>	Real	0.0	Deceleration ramp in case of a stop command ( <i>i_usiCmd</i> = 20) of the function.
<i>i_rFactorVel</i>	Real	0.0	Scaling factor
<i>i_xErrorObjAccess</i>	Bool	false	Indicate an error on Object Access
<i>i_xObjAccessLock</i>	Bool	false	Usage of Object Access is authorized

Output	Data Type	Default	Description
<i>q_xDone</i>	Bool	false	This is an endless function. In case of a stop command ( <i>i_usiCmd</i> = 20) indicates that the stopping procedure is completed
<i>q_xError</i>	Bool	false	Error during Profile Velocity procedure
<i>q_xAborted</i>	Bool	false	Profile Velocity procedure aborted
<i>q_xBusy</i>	Bool	false	Indicates the function block is busy



InOut	Data Type	Default	Description
<i>xStart</i>	Bool	False	Rising edge: start procedure.
<i>xAbort</i>	Bool	false	TRUE: abort Velocity procedure
<i>stAxis</i>	st_Axis		Axis I/O parameters
<i>stObjAccessData</i>	st_ObjAccessData		Data for Object Access
<i>xWriteObjAccess</i>	Bool	false	Write request for Object Access. It is cleared when the write is completed, or an error occurs ( <i>i_xErrorObjAccess</i> is set)
<i>xObjAccessRequestLock</i>	Bool	false	Request the authorization to use Object Access
<i>sAxisErrorMsg</i>	String	''	Axis error description

### Starting

The Mode of Operation is set to 2 (Velocity Mode), target velocity is set to 0 and the following profile velocity parameters are written to their respective addresses:

Address	Parameter	Source
0x6048/01	Acceleration delta speed	Calculated starting from $i\_stVelocity.rAcceleration * i\_rFactorVel$
0x6048/02	Acceleration delta time	Calculated starting from $i\_stVelocity.rAcceleration * i\_rFactorVel$
0x6049/01	Deceleration delta speed	Calculated starting from $i\_stVelocity.rDeceleration * i\_rFactorVel$
0x6049/02	Deceleration delta time	Calculated starting from $i\_stVelocity.rDeceleration * i\_rFactorVel$

Once the parameter configuration is complete and the drive confirms Mode of Operation 2 the starting phase terminate.

### Executing

This phase continuously updates Target Velocity and, during execution, acceleration (*i\_stVelocity.rAcceleration*) and deceleration (*i\_stVelocity.rDeceleration*) parameters can be modified. The velocity procedure doesn't have a competition condition. The only way to terminate it is to issue a stop command (i.e. by setting *i\_usiCmd* = 20). *q\_xDone* remains inactive.

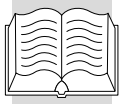
### Clearing

When profile velocity terminates (error condition or manual stop) the stop ramp is activated. Parameters 0x6049/01 (Deceleration delta speed) and 0x6049/02 (Deceleration delta time) are set according to *i\_rStopDeceleration* and Halt bit (bit 9) in the Control Word is set to initiate the stop ramp.

The deceleration phase concludes when either of the following Status Word conditions occurs:

- ➔ Target Reached bit (bit 11) is set, indicating controlled stop completion
- ➔ Operation Enabled bit (bit 3) becomes clear, indicating drive has exited operational state

At the end of the deceleration phase, if no errors occur, *q\_xDone* is set to TRUE.



## Error Conditions

During the starting phase and the execution phase, errors are triggered under the following conditions:

- ➔ Object Access Error: when *i\_xErrorObjAccess* is active
- ➔ The following parameter ranges are out of range:
  - *i\_stVelocity.rTargetVelocity* \* *i\_rFactorVel* < -32768 or > 32767
  - *i\_stVelocity.rAcceleration* \* *i\_rFactorVel* <= 0 or > 4294967295.0
  - *i\_stVelocity.rDeceleration* \* *i\_rFactorVel* <= 0 or > 4294967295.0
- ➔ Parameters are not successfully written within 2 seconds (only for the starting phase)

### NOTE

A negative value set in *i\_stVelocity.rTargetVelocity* (expressed in *vel. units*) results in a negative direction.



### 5.3.12 Object Access – Access Request (FB\_Mutex\_ObjAccess)

This function block serves as a support utility designed to manage and coordinate access requests to the Object Access communication protocol.

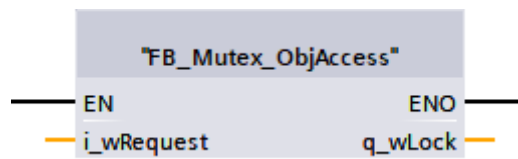


Figure 32: FB\_Mutex\_ObjAccess

Input	Data Type	Default	Description
<i>i_wRequest</i>	Word	16#0	Access request to Object Access. Each function has a specific request bit associated with it

Output	Data Type	Default	Description
<i>q_wLock</i>	Word	16#0	Lock status. If the corresponding bit of <i>i_wRequest</i> is set, function has access to the Object Access

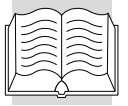
The *i\_wRequest* input word contains individual access request bits for different system functions:

Bit	Function Block	Purpose
1	<i>FB_MCVelocity</i>	Velocity control operations
2	<i>FB_MCProfileVelocity</i>	Profile velocity operations
3	<i>FB_MCProfilePosition</i>	Profile position operations
4	<i>FB_MCHoming</i>	Homing sequence operations
5	<i>FB_MCProfileTorque</i>	Profile torque operations
6	<i>Quick Stop Function</i>	Deceleration ramp operations
7	<i>FB_MCTouchProbe</i>	Touch probe data reading
8	<i>FB_MCTouchProbe</i>	Touch probe data writing
9	<i>FB_MCHardwareError</i>	Read fault code
10	<i>External Access</i>	User program access to Object Access

Bit 10 is for external Object Access access (i.e. custom applications) and is activated by setting *i\_xObjAccessRequest* bit. See [External Access to Object Access](#) for more details.

#### NOTE

The access request system implements a priority hierarchy. Highest priority: Internal function blocks (bits 1-9), lowest priority: External access (bit 10). When multiple requests occur simultaneously (e.g., *FB\_MCHoming* + external access), the external request is delayed, external requests wait until higher-priority operations complete.



### 5.3.13 Object Access (FB\_ObjAccess)

This function block is designed to perform reading and writing operations on Object Access.

This function is activated by setting *xRead* or *xWrite*.

#### NOTE

Before starting the request, an authorization access is necessary

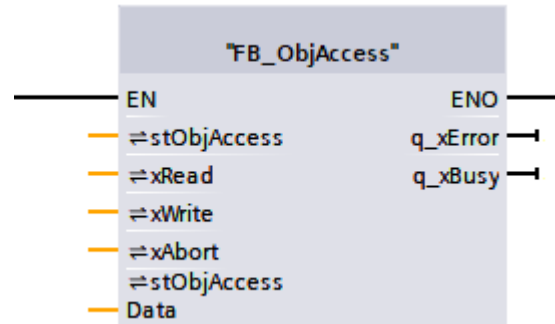


Figure 33: FB\_ObjAccess

Input	Data Type	Default	Description
<i>i_wObjectAccessLock</i>	Word	16#0	Object Access lock status

Output	Data Type	Default	Description
<i>q_xError</i>	Bool	false	Error during Profile Velocity procedure
<i>q_xBusy</i>	Bool	false	Indicates the function block is busy

InOut	Data Type	Default	Description
<i>stObjAccess</i>	st_ObjAccess		Data for Object Access. See <a href="#">st_ObjectAccess</a>
<i>xRead</i>	Word	16#0	Read request
<i>xWrite</i>	Word	16#0	Write request
<i>xAbort</i>	Bool	false	Abort function
<i>stObjAccessData</i>	st_ObjAccessData		Data to read or write on Object Access. See <a href="#">st_ObjectAccessData</a>

#### Starting

The starting phase set data in *stObjAccess* parameter, starting from *stObjAccessData*.

#### Executing

Perform the request (read or write) and read the result.

#### Clearing

Reset the request.

#### Error Conditions

In starting phase an error is triggered when both read and write operations are requested (*xRead* and *xWrite* set to true) simultaneously, or if *siDataType* is not configured with a valid value.

During execution, the system monitors the drive's response for error conditions (6<sup>th</sup> bit in *i\_bcmd*)



## External Access to Object Access

External Object Access access follows [Figure 34](#) flow.

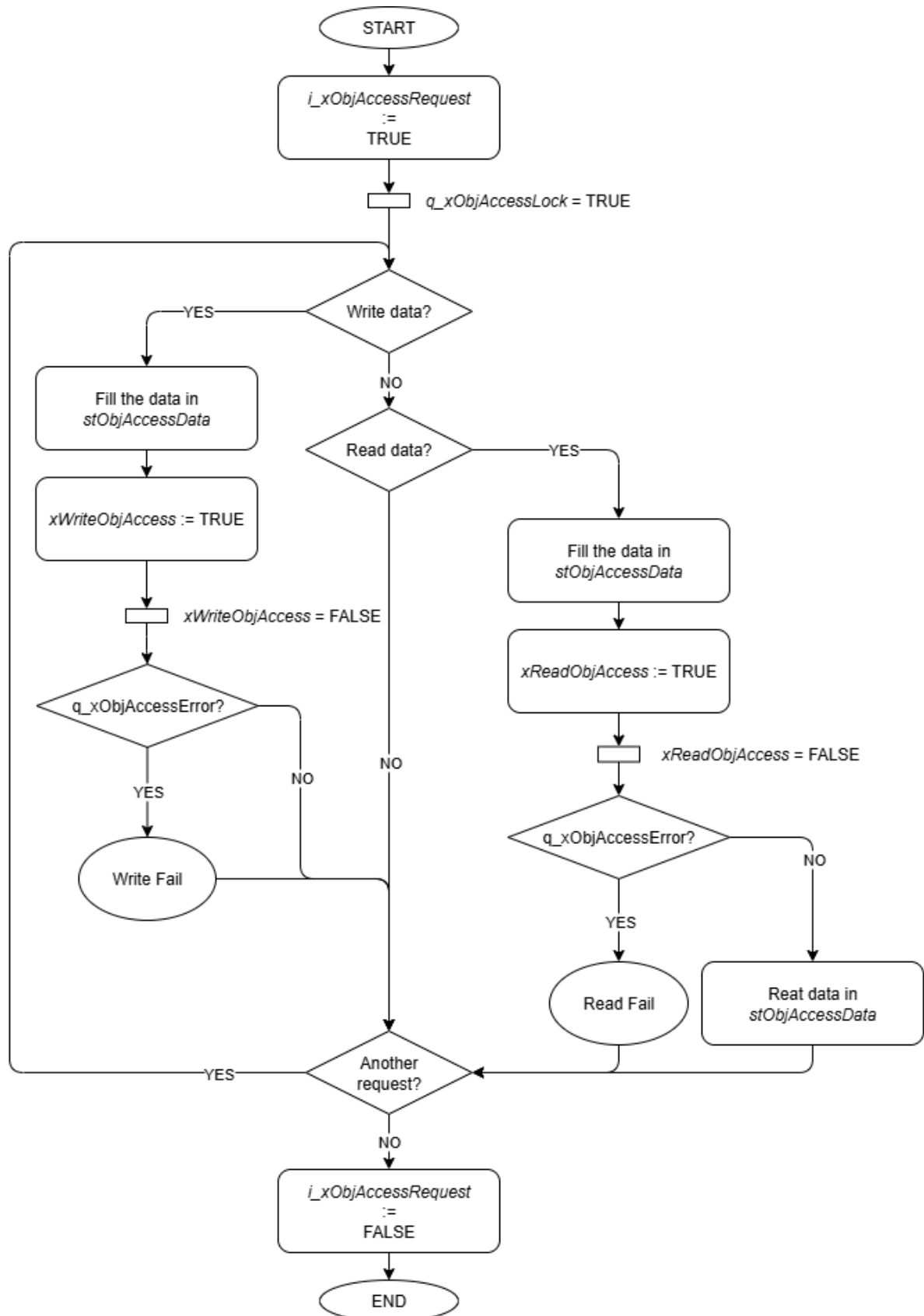


Figure 34: External Access to Object Access





### 5.3.14 Stop

As mentioned before, there isn't an FB that implement a stop function. This functionality is implemented in the clearing phase of each motion control function.

The Stop procedure stops any ongoing movement putting the axis in *Stopping* state. *i\_rStopDeceleration* input data is relevant.

In the case of invalid input data (e.g.: *i\_rStopDeceleration*  $\leq 0$ ) the function will be triggered anyway and the last deceleration value stored in the drive will be used. The deceleration value specified in the actual running function will be probably used.

#### NOTE

- Deceleration value are written into the drive via Object Access. In this case, the start of the execution of Stop function is delayed by the time taken for the data to be written. This must be considered in time-critical situation; a Quick Stop shall be used in this case.

Deceleration used to stop the axis. Depending on the AxisMgr used and to the actual Mode of Operation, the dimension of this value and the behavior will change:

- Mode Of Operation = 1: pos. units/s<sup>2</sup> (*obj. 0x6084* is used). If it is set to 0, the value stored in the drive will take effect. Only relevant if *obj. 0x605D* = 1.
- Mode Of Operation = 2: vel. units/s (*obj. 0x6049* is used). If it is set to 0, the value stored in the drive will take effect. Only relevant if *obj. 0x605D* = 1.
- Mode Of Operation = 3: pos. units/s<sup>2</sup> (*obj. 0x6084* is used). If it is set to 0, the value stored in the drive will take effect.
- Mode Of Operation = 4: not used, *obj 0x2502 – JOG Deceleration* is used
- Mode Of Operation = 6: not used. *i\_stHoming.lAcceleration* will be used instead.

Depending on the actual Mode of Operation, Stop function behaves accordingly:

- **1 - Profile Position** mode: as soon as Stop function was triggered by means of *i\_usiCmd* = 20 and a rising edge given to *i\_xLaunchCmd*, the function begins its execution and AxisMgr reaches *Stopping* state. AxisMgr halts the drive and the axis decelerates with the specified *i\_rStopDeceleration* (which is expressed in pos. units/s<sup>2</sup>) to 0 speed. Stop function ends its execution once the drive reaches 0 speed, so Axis Mgr reaches *Standstill* state.

#### NOTE

- Behavior of the drive when Stop function has been triggered during Profile Position Mode function may depend on the value set in *obj. 0x605D Halt Option Code*. Value of *i\_rStopDeceleration* is only relevant when this object is set to 1.
- **2 - Velocity** mode: as soon as Stop function was triggered by means of *i\_usiCmd* = 20 and a rising edge given to *i\_xLaunchCmd*, the function writes the target speed to 0, then AxisMgr reaches *Stopping* state. The axis decelerates with the specified *i\_rStopDeceleration* [vel. units/s] to 0 speed. Stop function ends its execution once the drive reaches 0 speed, so Axis Mgr reaches *Standstill* state.
- **3 - Profile Velocity** mode: as soon as Stop function was triggered by means of *i\_usiCmd* = 20 and a rising edge given to *i\_xLaunchCmd*, the function writes the target speed to 0, then AxisMgr reaches *Stopping* state. The axis decelerates with the specified *i\_rStopDeceleration* (which is expressed in pos. units/s<sup>2</sup>) to 0 speed. Stop function ends its execution once the drive reaches 0 speed, so Axis Mgr reaches *Standstill* state.
- **4 - Profile Torque** mode: as soon as Stop function was triggered by means of *i\_usiCmd* = 20 and a rising edge given to *i\_xLaunchCmd*, the function writes Control Word to 0, then Axis Mgr reaches *Stopping* state. The axis decelerates with the specified in *object 0x2502 – JOG Deceleration* [rpm/s]. Stop function ends its execution once the drive disables the motor power, so Axis Mgr reaches



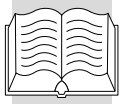
*Disabled state.*

#### NOTE

- At the end of the Stop function (after a running Velocity Mode), the drive may keep the motor powered and standstill until the *Holding time* has been expired (drive's internal parameter, refer to the drive documentation).
- **6 - Homing mode:** as soon as Stop function was triggered by means of  $i\_usiCmd = 20$  and a rising edge given to  $i\_xLaunchCmd$ , the function begins its execution and AxisMgr reaches *Stopping* state. AxisMgr halts the drive and the axis decelerates to 0 speed. In this case  $i\_rStopDeceleration$  is not relevant, instead the axis will decelerate with the value defined in  $i\_stHoming.rAcceleration$ . Stop function ends its execution once the drive reaches 0 speed, so AxisMgr reaches *Standstill* state.

#### NOTE

- The deceleration ramp used by the Stop function, when during an Homing procedure, is the value of  $i\_stHoming.rAcceleration$  when Homing began its execution.
- Behavior of the drive when Stop function has been triggered during Profile Position Mode function may depend on the value set in *obj. 0x605D Halt Option Code*. Value of  $i\_stHoming.rAcceleration$  is only relevant when this object is set to 1.



## 5.4 Support FBs and FCs

A collection of Function Blocks is designed to simplify the integration between drive hardware and software applications.

These FBs acts as mapping utilities between drive I/O and software structures. The need for manual, low-level programming is eliminated by providing:

- Automated mapping: automatically handles the correspondence between hardware signals and software variables
- Simplified integration: reduces development time and potential mapping errors

The user simply configures these support FBs to automatically handle all the mapping, allowing them to focus on the application logic rather than low-level drive communication details.

### 5.4.1 FB\_ObjAccessMap

Support utility for automatic mapping between *stObjAccess* (See [st ObjectAccess](#)) and *Object Access* I/Os.

The FB eliminates the need for manual data transfer by automatically handling the correspondence between the internal software object (*stObjAccess*) and the physical I/O data from the communication telegram.

The *FB\_ObjAccessMap* ensures that data from this telegram automatically populates the corresponding fields in internal *stObjAccess* data structure, and vice versa.

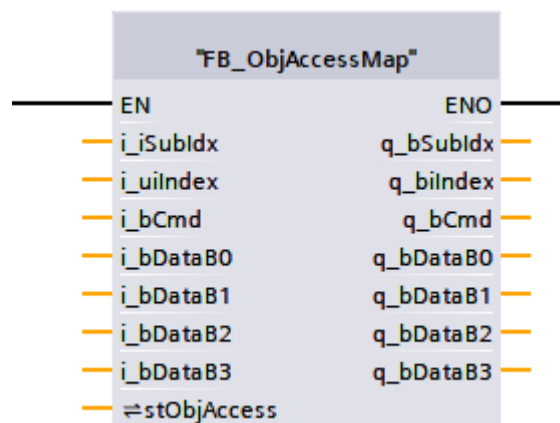


Figure 35: *FB\_ObjAccessMap*

Input	Data Type	Default	Description
<i>i_SubIdx</i>	Byte	16#0	Response (inverter → PLC) Byte 1 of Object Access
<i>i_uiIndex</i>	UInt	0	Response (inverter → PLC) Word 1 (Bytes 2 and 3) of Object Access
<i>i_bCmd</i>	Byte	16#0	Response (inverter → PLC) Byte 0 of Object Access
<i>i_bDataB0</i>	Byte	16#0	Response (inverter → PLC) Byte 4 of Object Access
<i>i_bDataB1</i>	Byte	16#0	Response (inverter → PLC) Byte 5 of Object Access
<i>i_bDataB2</i>	Byte	16#0	Response (inverter → PLC) Byte 6 of Object Access
<i>i_bDataB3</i>	Byte	16#0	Response (inverter → PLC) Byte 7 of Object Access



Output	Data Type	Default	Description
<i>q_bSubIdx</i>	Byte	16#0	Request (PLC → inverter) Byte 1 of Object Access
<i>q_biIndex</i>	UInt	0	Request (PLC → inverter) Word 1 (Bytes 2 and 3) of Object Access
<i>q_bCmd</i>	Byte	16#0	Request (PLC → inverter) Byte 0 of Object Access
<i>q_bDataB0</i>	Byte	16#0	Request (PLC → inverter) Byte 4 of Object Access
<i>q_bDataB1</i>	Byte	16#0	Request (PLC → inverter) Byte 5 of Object Access
<i>q_bDataB2</i>	Byte	16#0	Request (PLC → inverter) Byte 6 of Object Access
<i>q_bDataB3</i>	Byte	16#0	Request (PLC → inverter) Byte 7 of Object Access

InOut	Data Type	Default	Description
<i>stObjAccess</i>	st_ObjAccess		Data for Object Access. See <a href="#">st_ObjAccess</a>

#### NOTE

Object Access input/output parameters are mandatory

### 5.4.2 FB\_PROFINetMap

Support utility for automatic mapping between PLC software and AxiaVert I/O Telegrams.

The FB eliminates the need for manual data transfer by automatically handling the correspondence between the internal software objects (*stObjAccess* and *stAxis*) and the physical I/O data from the communication telegram.

The *FB\_PROFINetMap* ensures that data from this telegram automatically populates the corresponding fields in internal *stObjAccess* (see [st\\_ObjAccess](#)) and *stAxis* (see [st\\_Axis](#)) data structures, and vice versa.

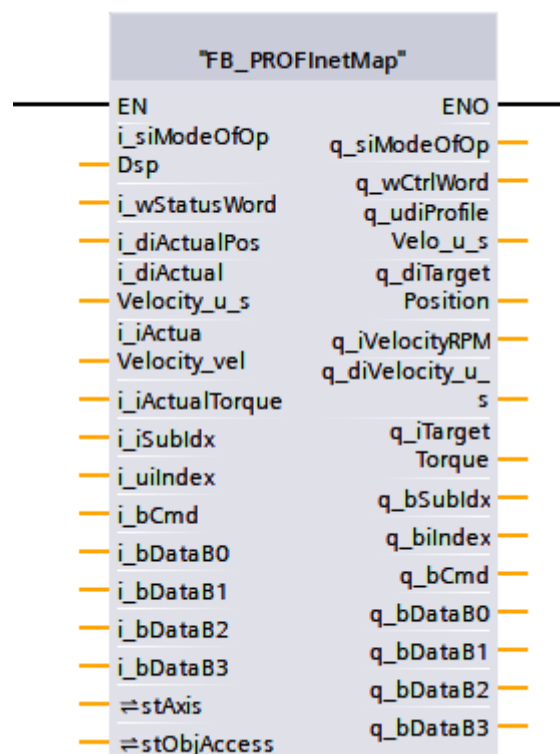
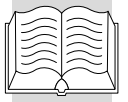


Figure 36: *FB\_PROFINetMap*



Input	Data Type	Default	Description
<i>i_siModeOfOpDsp</i>	SInt	0	Inverter → PLC Mode of Operation display
<i>i_wStatusWord</i>	Word	16#0	Inverter → PLC Status Word
<i>i_diActualPos</i>	DInt	0	Inverter → PLC Actual Position
<i>i_diActualVelocity_u_s</i>	DInt	0	Inverter → PLC Actual Velocity [u/s]
<i>i_iActuaVelocity_vel</i>	Int	0	Inverter → PLC Actual Velocity [rpm]
<i>i_iActualTorque</i>	Int	0	Inverter → PLC Actual Torque
<i>i_iSubIdx</i>	Byte	16#0	Response (inverter → PLC) Byte 1 of Object Access
<i>i_uiIndex</i>	UInt	0	Response (inverter → PLC) Word 1 (Bytes 2 and 3) of Object Access
<i>i_bCmd</i>	Byte	16#0	Response (inverter → PLC) Byte 0 of Object Access
<i>i_bDataB0</i>	Byte	16#0	Response (inverter → PLC) Byte 4 of Object Access
<i>i_bDataB1</i>	Byte	16#0	Response (inverter → PLC) Byte 5 of Object Access
<i>i_bDataB2</i>	Byte	16#0	Response (inverter → PLC) Byte 6 of Object Access
<i>i_bDataB3</i>	Byte	16#0	Response (inverter → PLC) Byte 7 of Object Access

Output	Data Type	Default	Description
<i>q_siModeOfOp</i>	SInt	0	PLC → inverter Mode of Operation
<i>q_wCtrlWord</i>	Word	16#0	PLC → inverter Control Word
<i>q_udiProfileVelo_u_s</i>	UDInt	0	PLC → inverter Profile Velocity [u/s]
<i>q_diTargetPosition</i>	DInt	0	PLC → inverter Target Position
<i>q_iVelocityRPM</i>	Int	0	PLC → inverter Velocity [rpm]
<i>q_diVelocity_u_s</i>	DInt	0	PLC → inverter Velocity [u/s]
<i>q_iTargetTorque</i>	Int	0	PLC → inverter Target Torque
<i>q_bSubIdx</i>	Byte	16#0	Request (PLC → inverter) Byte 1 of Object Access
<i>q_biIndex</i>	UInt	0	Request (PLC → inverter) Word 1 (Bytes 2 and 3) of Object Access
<i>q_bCmd</i>	Byte	16#0	Request (PLC → inverter) Byte 0 of Object Access
<i>q_bDataB0</i>	Byte	16#0	Request (PLC → inverter) Byte 4 of Object Access
<i>q_bDataB1</i>	Byte	16#0	Request (PLC → inverter) Byte 5 of Object Access
<i>q_bDataB2</i>	Byte	16#0	Request (PLC → inverter) Byte 6 of Object Access
<i>q_bDataB3</i>	Byte	16#0	Request (PLC → inverter) Byte 7 of Object Access

InOut	Data Type	Default	Description
<i>stAxis</i>	st_Axis		Axis I/O parameters. See <a href="#">st_Axis</a>
<i>stObjAccess</i>	st_ObjAccess		Data for Object Access. See <a href="#">st_ObjAccess</a>

#### NOTE

- ➔ Not all the input and output parameters are mandatory. Mandatory parameters are:
  - Object Access parameters
  - *i\_siModeOfOpDsp* and *q\_siModeOfOp*
  - *i\_wStatusWord* and *q\_wCtrlWord*
- ➔ The mapping of the other parameters are software dependent



### 5.4.3 FB\_PROFIBusMap

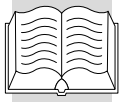
Support utility for automatic mapping between PLC software and AxiaVert POFIbus® channels. The FB eliminates the need for manual data transfer by automatically handling the correspondence between the internal software objects (*stObjAccess* and *stAxis*) and the physical I/O data from the communication telegram.

The *FB\_PROFIBusMap* ensures that data from this channels automatically populates the corresponding fields in internal *stObjectAccess* (see [st\\_ObjectAccess](#)) and *stAxis* (see [st\\_Axis](#)) data structures, and vice versa.



Figure 37: *FB\_PROFIBusMap*

Input	Data Type	Default	Description
<i>i_siModeOfOpDsp</i>	SInt	0	Inverter → PLC Mode of Operation display
<i>i_wStatusWord</i>	Word	16#0	Inverter → PLC Status Word
<i>i_diActualPos</i>	DInt	0	Inverter → PLC Actual Position
<i>i_diActualVelocity_u_s</i>	DInt	0	Inverter → PLC Actual Velocity [u/s]
<i>i_iActuaVelocity_vel</i>	Int	0	Inverter → PLC Actual Velocity [rpm]
<i>i_iActualTorque</i>	Int	0	Inverter → PLC Actual Torque
<i>i_iSubIdx</i>	Byte	16#0	Response (inverter → PLC) Byte 1 of Object Access
<i>i_uiIndex</i>	UInt	0	Response (inverter → PLC) Word 1 (Bytes 2 and 3) of Object Access
<i>i_bCmd</i>	Byte	16#0	Response (inverter → PLC) Byte 0 of Object Access
<i>i_bDataB0</i>	Byte	16#0	Response (inverter → PLC) Byte 4 of Object Access
<i>i_bDataB1</i>	Byte	16#0	Response (inverter → PLC) Byte 5 of Object Access
<i>i_bDataB2</i>	Byte	16#0	Response (inverter → PLC) Byte 6 of Object Access
<i>i_bDataB3</i>	Byte	16#0	Response (inverter → PLC) Byte 7 of Object Access



Output	Data Type	Default	Description
<i>q_siModeOfOp</i>	SInt	0	PLC → inverter Mode of Operation
<i>q_wCtrlWord</i>	Word	16#0	PLC → inverter Control Word
<i>q_udiProfileVelo_u_s</i>	UDInt	0	PLC → inverter Profile Velocity [u/s]
<i>q_diTargetPosition</i>	DInt	0	PLC → inverter Target Position
<i>q_iVelocityRPM</i>	Int	0	PLC → inverter Velocity [rpm]
<i>q_diVelocity_u_s</i>	DInt	0	PLC → inverter Velocity [u/s]
<i>q_iTargetTorque</i>	Int	0	PLC → inverter Target Torque
<i>q_bSubIdx</i>	Byte	16#0	Request (PLC → inverter) Byte 1 of Object Access
<i>q_biIndex</i>	UInt	0	Request (PLC → inverter) Word 1 (Bytes 2 and 3) of Object Access
<i>q_bCmd</i>	Byte	16#0	Request (PLC → inverter) Byte 0 of Object Access
<i>q_bDataB0</i>	Byte	16#0	Request (PLC → inverter) Byte 4 of Object Access
<i>q_bDataB1</i>	Byte	16#0	Request (PLC → inverter) Byte 5 of Object Access
<i>q_bDataB2</i>	Byte	16#0	Request (PLC → inverter) Byte 6 of Object Access
<i>q_bDataB3</i>	Byte	16#0	Request (PLC → inverter) Byte 7 of Object Access

InOut	Data Type	Default	Description
<i>stAxis</i>	st_Axis		Axis I/O parameters. See <a href="#">st_Axis</a>
<i>stObjAccess</i>	st_ObjAccess		Data for Object Access. See <a href="#">st_ObjAccess</a>

#### NOTE

- ➔ Not all the input and output parameters are mandatory. Mandatory parameters are:
  - Object Access parameters
  - *i\_siModeOfOpDsp* and *q\_siModeOfOp*
  - *i\_wStatusWord* and *q\_wCtrlWord*
- ➔ The mapping of the other parameters are software dependent



#### 5.4.4 FC\_GetFaultMessage

This FC serves as a fault code translator for drive systems. Translates numeric drive fault codes into human-readable error descriptions.

Input	Data Type	Default	Description
<i>i_iFaultCode</i>	Int		Drive fault code ( <i>obj. 0x4010/2</i> )

Output	Data Type	Default	Description
<i>q_sFaultMessage</i>	String		Description of <i>i_iFaultCode</i>

#### 5.4.5 FC\_GetWarningMessage

This FC serves as a warning codes translator for drive systems. Translates numeric drive warning codes into human-readable descriptions.

Input	Data Type	Default	Description
<i>i_usiWarningGroup</i>	USInt		Warning group (Power, Communication, Application, Encoder, <i>obj. 0x4015/1</i> )
<i>i_uiWarningCode</i>	UInt		Warning fault code ( <i>objs. 0x4015/2..5</i> )

Output	Data Type	Default	Description
<i>q_sWarningMessage</i>	String		Description of <i>i_uiWarningCode</i> and <i>i_usiWarningGroup</i>

#### 5.4.6 FB\_B\_AxisMgr\_Constants

This FB defines Axis Manager constants. See [Constants](#) for more details.





## 6 Data Unit Types

### 6.1 Constants

Constants are defined in the *FB\_B\_AxisMgr\_Constants* as output variables.

#### 6.1.1 AxisMgr Status

The enumeration contains the status that the axis can reach according to the AxisManager state diagram (see chapter [5.1](#)).

##### Constant Elements

Name	Data Type	Value	Description
<i>AXISMGR_DISABLED</i>	SInt	0	Axis not powered
<i>AXISMGR_STANDSTILL</i>	SInt	1	Axis powered and standstill
<i>AXISMGR_VELOCITY</i>	SInt	2	The axis is moving endless with the given target data
<i>AXISMGR_POSITIONING</i>	SInt	3	The axis is performing the positioning with the given target data.
<i>AXISMGR_TORQUE</i>	SInt	4	The axis is moving with the given torque data
<i>AXISMGR_HOMING</i>	SInt	5	Homing procedure is being carried out
<i>AXISMGR_ERRORSTOP</i>	SInt	6	The axis is in error state
<i>AXISMGR_QUICKSTOP</i>	SInt	7	The axis is performing the emergency stop triggered by the <i>i_xQuickStop</i> input
<i>AXISMGR_STOPPING</i>	SInt	8	The axis is stopping with the given target data
<i>AXISMGR_ABORTED</i>	SInt	9	AxisMgr aborted



## 6.1.2 Object Access

These constants can be used to set *siDataType* in *st\_ObjAccessData* variables.

### Constant Elements

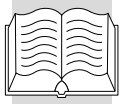
Name	Data Type	Value	Description
<i>OBJACCESS_INT8</i>	SInt	0	Data type to read/write is a 8 bits integer
<i>OBJACCESS_UINT8</i>	SInt	1	Data type to read/write is a 8 bits unsigned integer
<i>OBJACCESS_INT16</i>	SInt	2	Data type to read/write is a 16 bits integer
<i>OBJACCESS_UINT16</i>	SInt	3	Data type to read/write is a 16 bits unsigned integer
<i>OBJACCESS_INT32</i>	SInt	4	Data type to read/write is a 32 bits integer
<i>OBJACCESS_UINT32</i>	SInt	5	Data type to read/write is a 32 bits unsigned integer
<i>OBJACCESS_REAL</i>	SInt	6	Data type to read/write is a 32 bits floating point

## 6.1.3 Shutdown Procedure

These constants can be used to set *i\_usiShutdownMode* parameter.

### Constant Elements

Name	Data Type	Value	Description
<i>SHUTDOWN_DISABLE_VOLTAGE</i>	SInt	0	Set Control Word to 0
<i>SHUTDOWN_DISABLE_OPERATION</i>	SInt	1	Perform the shutdown procedure 7 → 6 → 0
<i>SHUTDOWN_SHUTDOWN</i>	SInt	2	Perform the shutdown procedure 6 → 0



## 6.1.4 Homing Method

These constants group defines all the possible choices for the Homing procedure that are supported by Bonfiglioli's drive (refer to the Application manual Positioning of the used drive for further information). The homing types are based on the CANOpen specification DSP402.

These constants can be used to set *usiMode* in *st\_Homing* variables.

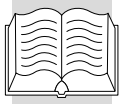
No.	Main destination		Fine destination (Ref. signal)	Limit Switch ?	
1	Left	limit switch	Ref. signal right	Left limit switch	
2	Right		Ref. signal left	Right limit switch	
3	Negative	home switch	Ref. signal left	Without limit switch	
4			Ref. signal right		
5	Positive		Ref. signal right		
6			Ref. signal left		
7	Left edge	home switch	Ref. signal left	Right limit switch	
8			Ref. signal right		
9	Right edge		Ref. signal left		
10			Ref. signal right		
11	Right edge		Ref. signal right	Left limit switch	
12			Ref. signal left		
13	Left edge		Ref. signal right		
14			Ref. signal left		
15	Reserved				
16	Reserved				
17	Left	limit switch	Falling edge	Left limit switch	
18	Right		Falling edge	Right limit switch	
19	Negative	home switch	Falling edge	Without limit switch	
20			Rising edge		
21	Positive		Falling edge		
22			Rising edge		
23	Left edge	home switch	Falling edge	Right limit switch	
24			Rising edge		
25	Right edge		Rising edge		
26			Falling edge		
27	Right edge		Falling edge	Left limit switch	
28			Rising edge		
29	Left edge		Rising edge		
30			Falling edge		
31	Reserved				
32	Reserved				
33	Left	ref. signal			
34	Right				
35	Current	position			

Figure 38: Homing Modes



## Constant Elements

Name	Data Type	Value	Description
<i>HOMING_NOHOMING</i>	SInt	0	No homig; the current position value is not changed. The current position value is the value saved upon last disconnection of power supply.
<i>HOMING_NEG_LIMITSWITCH_AND_REFSIGNAL</i>	SInt	1	Homing to negative HW limit switch with detection of encoder ref. signal.
<i>HOMING_POS_LIMITSWITCH_AND_REFSIGNAL</i>	SInt	2	Homing to positive HW limit switch with detection of encoder ref. signal.
<i>HOMING_POS_HOMESWITCH_REFSIGNAL_LEFTOFEDGE</i>	SInt	3	Homing to positive home switch with detection of encoder ref. signal. Home position is the first encoder ref. signal to the left of the edge of the home switch signal.
<i>HOMING_POS_HOMESWITCH_REFSIGNAL_RIGHTOFEDGE</i>	SInt	4	Homing to positive home switch with detection of encoder ref. signal. Home position is the first encoder ref. signal to the right of the edge of the home switch signal.
<i>HOMING_NEG_HOMESWITCH_REFSIGNAL_RIGHTOFEDGE</i>	SInt	5	Homing to negative home switch with detection of encoder ref. signal. Home position is the first encoder ref. signal to the left of the edge of the home switch signal.
<i>HOMING_NEG_HOMESWITCH_REFSIGNAL_LEFTOFEDGE</i>	SInt	6	Homing to negative home switch with detection of encoder ref. signal. Home position is the first encoder ref. signal to the left of the edge of the home switch signal.
<i>HOMING_POS_LIMITSWITCH_REFSIGNAL_LEFTOFFLEFTEDGEOFHOMESWITCH</i>	SInt	7	Homing to home switch with detection of encoder ref. signal. Homing direction positive (clockwise). Reversal of direction of rotation when positive HW limit switch is reached. Home position is the first encoder ref. signal to the left or right of the left or right edge of the home signal switch.
<i>HOMING_POS_LIMITSWITCH_REFSIGNAL_RIGHTOFFLEFTEDGEOFHOMESWITCH</i>	SInt	8	
<i>HOMING_POS_LIMITSWITCH_REFSIGNAL_LEFTOFFRIGHTEDGEOFHOMESWITCH</i>	SInt	9	
<i>HOMING_POS_LIMITSWITCH_REFSIGNAL_RIGHTOFFRIGHTEDGEOFHOMESWITCH</i>	SInt	10	
<i>HOMING_NEG_LIMITSWITCH_REFSIGNAL_LEFTOFFLEFTEDGEOFHOMESWITCH</i>	SInt	11	Homing to home switch with detection of encoder ref. signal. Homing direction negative (anticlockwise). Reversal of direction of rotation when negative HW limit switch is reached. Home position is the first encoder ref. signal to the left or right of the left or right edge of the home signal switch.
<i>HOMING_NEG_LIMITSWITCH_REFSIGNAL_RIGHTOFFLEFTEDGEOFHOMESWITCH</i>	SInt	12	
<i>HOMING_NEG_LIMITSWITCH_REFSIGNAL_LEFTOFFRIGHTEDGEOFHOMESWITCH</i>	SInt	13	
<i>HOMING_NEG_LIMITSWITCH_REFSIGNAL_RIGHTOFFRIGHTEDGEOFHOMESWITCH</i>	SInt	14	
17..30: like 1..14 but without encoder ref. signal			
<i>HOMING_NEG_LIMITSWITCH</i>	SInt	17	Homing to negative HW limit switch
<i>HOMING_POS_LIMITSWITCH</i>	SInt	18	Homing to positive HW limit switch
<i>HOMING_POS_HOMESWITCH_LEFTOFEDGE</i>	SInt	19	Homing to positive home switch. Home position is at the left of the edge of the home switch signal.
<i>HOMING_POS_HOMESWITCH_RIGHTOFEDGE</i>	SInt	20	Homing to positive home switch. Home position is at the right of the edge of the home switch signal.
<i>HOMING_NEG_HOMESWITCH_RIGHTOFEDGE</i>	SInt	21	Homing to negative home switch. Home position is at the right of the edge of the home switch signal.



<i>HOMING_NEG_HOMESWITCH_LEFTOFEDGE</i>	SInt	22	Homing to negative home switch. Home position is at the left of the edge of the home switch signal.
<i>HOMING_POS_LIMITSWITCH_LEFTOFFLEFTEDGEOFHOMESWITCH</i>	SInt	23	Homing to home switch. Homing direction positive (clockwise). Reversal of direction of rotation when positive HW limit switch is reached. Home position is at the left or right of the left or right edge of the home switch signal.
<i>HOMING_POS_LIMITSWITCH_RIGHTOFFLEFTEDGEOFHOMESWITCH</i>	SInt	24	
<i>HOMING_POS_LIMITSWITCH_LEFTOFFRIGHTEDGEOFHOMESWITCH</i>	SInt	25	
<i>HOMING_POS_LIMITSWITCH_RIGHTOFFRIGHTEDGEOFHOMESWITCH</i>	SInt	26	
<i>HOMING_NEG_LIMITSWITCH_LEFTOFFLEFTEDGEOFHOMESWITCH</i>	SInt	27	Homing to home switch. Homing direction negative (anticlockwise). Reversal of direction of rotation when negative HW limit switch is reached. Home position is at the left or right of the left or right edge of the home switch signal.
<i>HOMING_NEG_LIMITSWITCH_RIGHTOFFLEFTEDGEOFHOMESWITCH</i>	SInt	28	
<i>HOMING_NEG_LIMITSWITCH_LEFTOFFRIGHTEDGEOFHOMESWITCH</i>	SInt	29	
<i>HOMING_NEG_LIMITSWITCH_RIGHTOFFRIGHTEDGEOFHOMESWITCH</i>	SInt	30	
<i>HOMING_REFSIGNAL_LEFTOFACTPOSITION</i>	SInt	33	Home position is the first encoder ref. signal in negative (operation mode 33) or positive (operation mode 34) direction.
<i>HOMING_REFSIGNAL_RIGHTOFACTPOSITION</i>	SInt	34	
<i>HOMING_CURRENTPOSITION</i>	SInt	35	Current position is home position. Final position is taken over as actual position value.

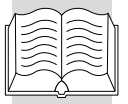


### 6.1.5 Error strings

These constants are used to set *q\_sAxisErrorMsg*.

#### Constant Elements

Name	Data Type	Value
ERROR_POWER_5_SECONDS	String	'Axis not powered within 5 seconds'
ERROR_POWER_STATUS_WORD	String	'Check status word. Bit 4 or 9 missing'
ERROR_POWER_SHUTDOWN_MODE	String	'Shutdown mode not supported'
ERROR_OBJACCESS_WRITE	String	'Error during writing Object Access'
ERROR_ACCELERATION_OOR	String	'Acceleration out of range'
ERROR_DECELERATION_OOR	String	'Deceleration out of range'
ERROR_TARGET_VELO_OOR	String	'Target velocity out of range'
ERROR_TIMEOUT	String	'Timer expired'
ERROR_TRIGGER_CMD_NOT_POWERED	String	'Triggered command while axis is not powered'
ERROR_TRIGGER_CMD_IN_ERROR	String	'Triggered command while axis in error'
ERROR_COMMUNICATION_READ	String	'Error reading axis module status'
ERROR_COMMUNICATION_ERROR	String	'Communication error with axis'
ERROR_TARGET_TORQUE_OOR	String	'Target torque out of range'
ERROR_TORQUE_SLOPE_OOR	String	'Torque slope out of range'
ERROR_TARGET_POS_OOR	String	'Target position out of range'
ERROR_XCHANGEIMMEDIATELY_AND_XCHANGESETPOINT	String	'xChangeOnSetPoint and xChangeSetImmediately are both TRUE'
ERROR_HOMING_METHOD_NOT_SUPPORTED	String	'Homing method not supported'
ERROR_POWEROFF_NOT_PERMITTED	String	'Power off not permitted'
ERROR_COMMAND_NOT_AVAILABLE	String	'Command cannot be triggered'
ERROR_HOMING_SEARCH_OOR	String	'Search velocity out of range'
ERROR_HOMING_LEAVE_OOR	String	'Leave velocity out of range'
ERROR_HOMING_ERROR	String	'Homing error'



## 6.2 Structures

Structures are defined as PLC Data Types.

### 6.2.1 st\_Axis

This structure contains parameters that must be connected to AxiaVert Telegrams to control it.

Parameter	Data Type	Default	Description
<i>i_wStatusWord</i>	Word	16#0	Input: status word
<i>i_siModeOfOpDsp</i>	SInt	0	Input: mode of operation display
<i>i_diActualPos</i>	DInt	0	Input: position actual value
<i>i_diActualVelocity_u_s</i>	DInt	0	Input: velocity actual value
<i>i_iActuaVelocity_vel</i>	Int	0	Input: vl velocity actual value
<i>i_iActualTorque</i>	Int	0	Input: actual torque
<i>q_wCtrlWord</i>	Word	16#0	Output: control word
<i>q_siModeOfOp</i>	SInt	0	Output: mode of operation
<i>q_iVelocityRPM</i>	Int	0	Output: vl target velocity (Modes of Operation = 2)
<i>q_diVelocity_u_s</i>	DInt	0	Output: velocity (in u/s) for Profile Velocity Mode (Modes of Operation = 3)
<i>q_diTargetPosition</i>	DInt	0	Output: target position for Profile Position (Modes of Operation = 1)
<i>q_udiProfileVelo_u_s</i>	UDInt	0	Output: target velocity (in u/s) for Profile Position (Modes of Operation = 1)
<i>q_iTargetTorque</i>	Int	0	Output: target torque for Profile Torque Mode (Modes of Operation = 4)



### 6.2.2 st\_Homing

This structure contains the relevant parameter for the Homing function.

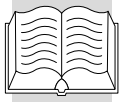
Parameter	Data Type	Default	Description
<i>usiMode</i>	USInt	0	HOME mode desired. See <a href="#">6.1.4</a>
<i>rSearchCamVelocity</i>	Real	0.0	Search speed [units/sec] of the reference cam. Not relevant for virtual axis/simulation mode
<i>rLeaveCamVelocity</i>	Real	0.0	Zero cam [units/sec] abandonment speed. Not relevant for virtual axis/simulation mode
<i>rAcceleration</i>	Real	0.0	Acceleration [units/sec <sup>2</sup> ] used during movement in <i>Homing</i> . Not relevant for virtual axis/simulation mode
<i>rFinalPosition</i>	Real	0.0	Position assigned to the axis at the end of the <i>Homing</i> command [units]

### 6.2.3 st\_ProfilePosition

This structure contains the relevant parameter for the Profile Position function.

Parameter	Data Type	Default	Description
<i>rTargetPos</i>	Real	0.0	Target position [units] that should be reached (for absolute positioning) or Distance from the actual position (for relative positioning). Changing of this value during an ongoing positioning will trigger a new positioning command with different target (not in the case of relative positioning).
<i>rTargetVel</i>	Real	0.0	Target velocity [units/s] used during positioning (not necessarily reached). Changing of this value during an ongoing positioning will trigger a new positioning command with different target (not in the case of relative positioning).
<i>rAcceleration</i>	Real	0.0	Acceleration [units/s <sup>2</sup> ] used during positioning. Changing of this value during an ongoing positioning will trigger a new positioning command with different value (not in the case of relative positioning).
<i>rDeceleration</i>	Real	0.0	Deceleration [units/s <sup>2</sup> ] used during positioning. Changing of this value during an ongoing positioning will trigger a new positioning command with different value (not in the case of for relative positionin).





<i>rPositionWindow</i>	Real	0.0	It defines the symmetrical window [units] around the target position where the axis should stay in order to terminate the function. If it is greater than 0, AxisMgr will copy the value into <i>obj. 0x6067 Position window</i> and used by the drive to establish the whether the axis is in the window. A value of 0 won't be copied into the drive, the value stored in <i>obj. 0x6067</i> will be used.
<i>uiPositionWindowTime_ms</i>	UInt	0	It defines the time delay [ms] within which the axis should remain in <i>rPositionWindow</i> before to raise <i>q_xInPos</i> . If it is greater than 0, AxisMgr will copy the value into <i>obj. 0x6068 Position window time</i> and used by the drive to establish whether the axis is in the window for the time set. A value of 0 won't be copied into the drive, the value stored in <i>obj. 0x6068</i> will be used.
<i>xChangeSetImmediately</i>	Bool	false	TRUE: Immediately. As soon as target position, velocity, acceleration and/or deceleration are updated during an ongoing positioning, the function writes the information into the drive's object. When all the information are written successfully, a new positioning command is triggered and the new target values will be applied.
<i>xChangeOnSetPoint</i>	Bool	false	TRUE: Immediate On Set Point. As soon as target position, velocity, acceleration and/or deceleration are updated during an ongoing positioning, the function writes the information into the drive's object. The new target values will be applied only once the previous movement reaches the target. The axis won't stop on the target but it will maintain the same operating speed, then the new profile position will be applied immediately "on-the-fly"
<i>xRelative</i>	Bool	false	FALSE: target position relates to the fixed reference position which is determined by a homing operation. An absolute distance is covered, referred to the reference position TRUE: A relative positioning operation relates always to the current position of the axis. New target position = current position + relative distance

#### NOTE

if both *xChangeSetImmediately* and *xChangeOnSetPoint* are set to FALSE, as soon as target position, velocity, acceleration and/or deceleration are updated during an ongoing positioning, the function writes the information into the drive's object. The new target values will be applied only once the previous movement reaches the target. The axis will stop on the target, then the new profile position will immediately start from 0 speed



## 6.2.4 st\_ProfileTorque

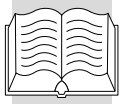
This structure contains the relevant parameter for the Profile Torque function.

Parameter	Data Type	Default	Description
<i>iTargetTorque</i>	Real	0.0	Target torque [Nm] used during torque control. Changes of this value during an ongoing torque control immediately affect the axis.
<i>udiTorqueSlope</i>	Real	0.0	Torque Slope [Nm/s] used during torque control. Changes of this value during an ongoing torque control immediately affect the axis.

## 6.2.5 st\_ProfileVelocity

This structure contains the relevant parameter for the Profile Velocity function.

Parameter	Data Type	Default	Description
<i>rTargetVelocity</i>	Real	0.0	Target velocity [units/s] used during velocity. Changing of this value during an ongoing positioning will take effect.
<i>rAcceleration</i>	Real	0.0	Acceleration [units/s <sup>2</sup> ] used during velocity. Changing of this value during an ongoing positioning will take effect.
<i>rDeceleration</i>	Real	0.0	Deceleration [units/s <sup>2</sup> ] used during velocity. Changing of this value during an ongoing positioning will take effect.
<i>rVelocityWindow</i>	Real	1.0	During cleaning phase (stop phase) it defines the symmetrical velocity windows [units/s] around the 0 velocity where the axis should operate in order to consider it stopped.
<i>tVelocityWindowTime</i>	Time	T#100ms	Time delay in which the axis should operate in <i>rVelocityWindow</i> before consider it stopped



## 6.2.6 st\_ObjectAccess

Parameters that must be connected to Telegram 120 (PROFINet®) or to the Object Access Channel (PROFibus®).

### OUT/Request (PLC → inverter)

### IN/Response (inverter → PLC)

Byte	0	1	2	3	4	5	6	7	
			MSB	LSB					
	cmd	SubIdx	Index		data				
	ATTENTION: Data is located in the „data“ part in Big-Endian format!  see here →				uint8/int8				
					MSB		LSB		
					uint16/int16				
					MSB				LSB
					uint32/int32/float				

Figure 39: Object Access structure

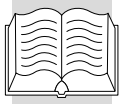
Parameter	Data Type	Default	Description
<i>i_bCmd</i>	Byte	16#0	Input: command byte (byte 0)
<i>i_bSubIdx</i>	Byte	16#0	Input: subindex byte (byte 1)
<i>i_uiIndex</i>	UInt	0	Input: index word (bytes 2 and 3)
<i>i_bDataB0</i>	Byte	16#0	Input: data byte 0 (byte 4)
<i>i_bDataB1</i>	Byte	16#0	Input: data byte 1 (byte 5)
<i>i_bDataB2</i>	Byte	16#0	Input: data byte 2 (byte 6)
<i>i_bDataB3</i>	Byte	16#0	Input: data byte 3 (byte 7)
<i>q_bCmd</i>	Byte	16#0	Output: command byte (byte 0)
<i>q_bSubIdx</i>	Byte	16#0	Output: subindex byte (byte 1)
<i>q_uiIndex</i>	UInt	0	Output: index word (bytes 2 and 3)
<i>q_bDataB0</i>	Byte	16#0	Output: data byte 0 (byte 4)
<i>q_bDataB1</i>	Byte	16#0	Output: data byte 1 (byte 5)
<i>q_bDataB2</i>	Byte	16#0	Output: data byte 2 (byte 6)
<i>q_bDataB3</i>	Byte	16#0	Output: data byte 3 (byte 7)



## 6.2.7 st\_ObjectAccessData

This structure contains the relevant parameter for the data to read or write via Object Access.

Parameter	Data Type	Default	Description
<i>diData</i>	DInt	0	Data in DINT format. In case of a read request, it contains the read value. In case of a write request, it contains the DINT value that has to be written ( <i>siDataType</i> = "OBJACCESS_INT8", "OBJACCESS_INT16" or "OBJACCESS_INT16")
<i>udiData</i>	UDInt	0	Data in UDINT format. In case of a read request, it contains the read value. In case of a write request, it contains the UDINT value that has to be written ( <i>siDataType</i> = "OBJACCESS_UINT8", "OBJACCESS_UINT16" or "OBJACCESS_UINT16")
<i>rData</i>	Real	0.0	Data in REAL format. In case of a read request, it contains the read value. In case of a write request, it contains the REAL value that has to be written ( <i>siDataType</i> = "OBJACCESS_REAL")
<i>bSubIdx</i>	Byte	16#0	Subindex to read or write
<i>uiIndex</i>	UInt	0	Index to read or write
<i>siDataType</i>	SInt	0	Data type to read or write: → 0: OBJACCESS_INT8 → 1: OBJACCESS_UINT8 → 2: OBJACCESS_INT16 → 3: OBJACCESS_UINT16 → 4: OBJACCESS_INT32 → 5: OBJACCESS_UINT32 → 6: OBJACCESS_REAL See <a href="#">Object Access</a> for more details



## 6.2.8 st\_TouchProbe

This structure contains the relevant parameter for the Touch Probe function.

Parameter	Data Type	Default	Description
<i>stTP1</i>	st_TouchProbeChannel		Touch Probe data for channel 1
<i>stTP2</i>	st_TouchProbeChannel		Touch Probe data for channel 2
<i>tRefreshTouchProbeStatusTimer</i>	Time	T#100ms	Refresh update timer. AxisMgr refreshes the Touch probe status information every <i>tRefreshTouchProbeStatusTimer</i> time.

## 6.2.9 st\_TouchProbeChannel

This structure contains the actual TP channel specific configuration data.

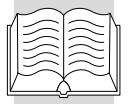
Parameter	Data Type	Default	Description
<i>xTouchProbeEnable</i>	Bool	false	TRUE: Enables the touch probe channel. Touch probe status for this channel is updated.
<i>xTouchProbeModeContinuous</i>	Bool	false	FALSE: Since TP channel was enabled, Touch Probe function detects the next TP event only (both negative and positive edges). Further TP input events won't be evaluated. The user should disable TP channel and re-enable again to start another evaluation. TRUE: As long as TP channel was enabled, Touch Probe continuously evaluates the TP channel inputs. Information are updated at each TP event detected.



## 6.2.10 st\_TouchProbeChannelStatus

This structure contains the actual TP channel specific configuration data.

Parameter	Data Type	Default	Description
<i>xPosEdgeValueValid</i>	Bool	false	<p><i>xPosEdgeValueValid</i> should be evaluated on the edges:</p> <ul style="list-style-type: none"><li>➔ Negative edge: a positive edge on the TP channel has been detected. From this moment, AxisMgr will read out the Pos edge position and the Pos edge counter registers stored in the drive. These data are to be considered invalid (not up to date) and should not be evaluated yet.</li><li>➔ Positive edge: the register has been read succesfully, <i>rPosEdgePositionValue</i> and <i>uiPosEdgeCounter</i> can be now considered as valid (up to date, related to the last Pos edge of the TP channel).</li></ul> <p>The time delay between the physical TP channel event and the negative edge of <i>xPosEdgeValueValid</i> is affected at maximum by <i>tRefreshTouchProbeStatusTimer + time for the Object Access response + task cycle</i></p>
<i>xNegEdgeValueValid</i>	Bool	false	<p><i>xNegEdgeValueValid</i> should be evaluated on the edges:</p> <ul style="list-style-type: none"><li>➔ Negative edge: a negative edge on the TP channel has been detected. From this moment, AxisMgr will read out the Neg edge position and the Neg edge counter registers stored in the drive. These data are to be considered invalid (not up to date) and should not be evaluated yet.</li><li>➔ Positive edge: the register has been read succesfully, <i>rNegEdgePositionValue</i> and <i>uiNegEdgeCounter</i> can be now considered as valid (up to date, related to the last Neg edge of the TP channel).</li></ul> <p>The time delay between the physical TP channel event and the indication of <i>xNegEdgeValueValid</i> is affected at maximum by <i>tRefreshTouchProbeStatusTimer + time for the Object Access response + task cycle</i></p>



<i>rPosEdgePositionValue</i>	Real	0.0	It returns the position value [units] stored in the drive when the positive edge of the TP channel was detected by the drive. It should be evaluated at the rising edge of <i>xPosEdgeValueValid</i> .
<i>rNegEdgePositionValue</i>	Real	0.0	It returns the position value [units] stored in the drive when the negative edge of the TP channel was detected by the drive. It should be evaluated at the rising edge of <i>xNegEdgeValueValid</i> .
<i>uiPosEdgeCounter</i>	UInt	0	It returns the number of the positive edges of the TP channel detected by the inverter (since the last power-on). It should be evaluated at the rising edge of <i>xPosEdgeValueValid</i> .
<i>uiNegEdgeCounter</i>	UInt	0	It returns the number of the positive edges of the TP channel detected by the inverter (since the last power-on). It should be evaluated at the rising edge of <i>xNegEdgeValueValid</i> .
<i>udiPosEdgeTimeStamp</i>	UDInt	0	It returns the timestamp stored in the drive when the positive edge of the TP channel was detected by the drive. It should be evaluated at the rising edge of <i>xPosEdgeValueValid</i> .
<i>udiNegEdgeTimeStamp</i>	UDInt	0	It returns the timestamp stored in the drive when the negative edge of the TP channel was detected by the drive. It should be evaluated at the rising edge of <i>xPosEdgeValueValid</i> .

### 6.2.11 st\_Velocity

This structure contains the relevant parameter for the Velocity function.

Parameter	Data Type	Default	Description
<i>rTargetVelocity</i>	Real	0.0	Target velocity [vel. units] used during velocity. Changing of this value during an ongoing positioning will take effect.
<i>rAcceleration</i>	Real	0.0	Acceleration [vel. units/s] used during velocity. Changing of this value during an ongoing positioning will take effect.
<i>rDeceleration</i>	Real	0.0	Deceleration [vel. units/s] used during velocity. Changing of this value during an ongoing positioning will take effect.
<i>rVelocityWindow</i>	Real	1.0	During cleaning phase (stop phase) it defines the symmetrical velocity windows [vel. units] around the 0 velocity where the axis should operate in order to consider it stopped.
<i>tVelocityWindowTime</i>	Time	T#100ms	Time delay in which the axis should operate in <i>rVelocityWindow</i> before consider it stopped



## 7 Example

The following example, that is available in the library, is the continuation of what was explained in [Fieldbus management](#), and is fieldbus independent.

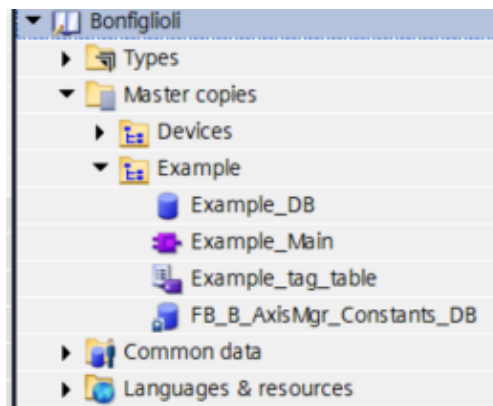


Figure 40: Library - Example

To control the motor in velocity mode (*obj. 0x6060 = 2*) these steps must be followed:

- ➔ Declare new variables
- ➔ Create *FB\_B\_AxisMgr\_Constants\_DB* instance of *FB\_B\_AxisMgr\_Constants* (available in the library)
- ➔ Create an instance of *FB\_AxisMgr*
- ➔ Create SCL program

### 7.1 Variable definition

Variables that have been declared are listed in the following table.

Name	Data Type	Description
<i>axis</i>	st_Axis	Axis I/O
<i>objAccess</i>	st_ObjAccess	Object Access Channel I/O
<i>objAccessData</i>	st_ObjAccessData	Object Access Channel data
<i>xPower</i>	Bool	Power on/off the axis
<i>usiCmd</i>	USInt	Command to launch
<i>xLaunchCmd</i>	Bool	Launch the command
<i>xQuickStop</i>	Bool	Quick stop command for the axis
<i>xReset</i>	Bool	Error reset
<i>xAbort</i>	Bool	Abort AxisMgr
<i>stVelocity</i>	st_Velocity	Velocity mode parameters
<i>l_xStart</i>	Bool	Used by the SCL program
<i>i</i>	USint	State machine index, used by SCL program
<i>xT120LockRequest</i>	Bool	Object Access channel access request
<i>xT120Locked</i>	Bool	Object Access channel locked
<i>xT120Error</i>	Bool	Object Access channel in error
<i>xT120Read</i>	Bool	Read via Object Access Channel
<i>xT120Write</i>	Bool	Write via Object Access Channel
<i>l_xReadObj</i>	Bool	Used by the SCL program
<i>l_xWriteObj</i>	Bool	Used by the SCL program





<i>j</i>	USInt	State machine index for read via Object Access channel
<i>k</i>	USInt	State machine index for write via Object Access channel
<i>l_uiAccTime</i>	UInt	Used by the SCL program
<i>xError</i>	Bool	Used by the SCL program

DB			
	Name	Data type	Start value
1	Static		
2	axis	"st_Axis"	
3	objAccess	"st_ObjAccess"	
4	objAccessData	"st_ObjAccessData"	
5	xPower	Bool	false
6	usiCmd	USInt	0
7	xLaunchCmd	Bool	false
8	xQuickStop	Bool	false
9	xReset	Bool	false
10	xAbort	Bool	false
11	stVelocity	"st_Velocity"	
12	l_xStart	Bool	false
13	i	USInt	0
14	xT120LockRequest	Bool	false
15	xT120Locked	Bool	false
16	xT120Error	Bool	false
17	xT120Read	Bool	false
18	xT120Write	Bool	false
19	l_xReadObj	Bool	false
20	l_xWriteObj	Bool	false
21	j	USInt	0
22	k	USInt	0
23	l_uiAccTime	UInt	0
24	xError	Bool	false

Figure 41: Velocity mode example - variables definition



## 7.2 FB\_AxisMgr instance

Instantiate *FB\_AxisMgr* and set the following parameters

Parameter	Variable
<i>i_xPower</i>	"DB".xPower
<i>i_usiCmd</i>	"DB".usiCmd
<i>i_xLaunchCmd</i>	"DB".xLaunchCmd
<i>i_xQuickStop</i>	"DB".xQuickStop
<i>i_xReset</i>	"DB".xReset
<i>i_xAbort</i>	"DB".xAbort
<i>i_stVelocity</i>	"DB".stVelocity
<i>I_xObjectAccessRequestLock</i>	"DB".xT120LockRequest
<i>i_stAxiaVertLADDR</i>	<i>AxiaVert LADDR HW_DEVICE address</i>
<i>stAxis</i>	"DB".stAxis
<i>stObjAccess</i>	"DB".objAccess
<i>stObjAccessData</i>	"DB".objAccessData
<i>xWriteObjAccess</i>	"DB".xT120Write
<i>xReaadObjAccess</i>	"DB".xT120Read
<i>q_xObjectAccessLock</i>	"DB".xT120Locked
<i>q_xObjAccessError</i>	"DB".xT120Error

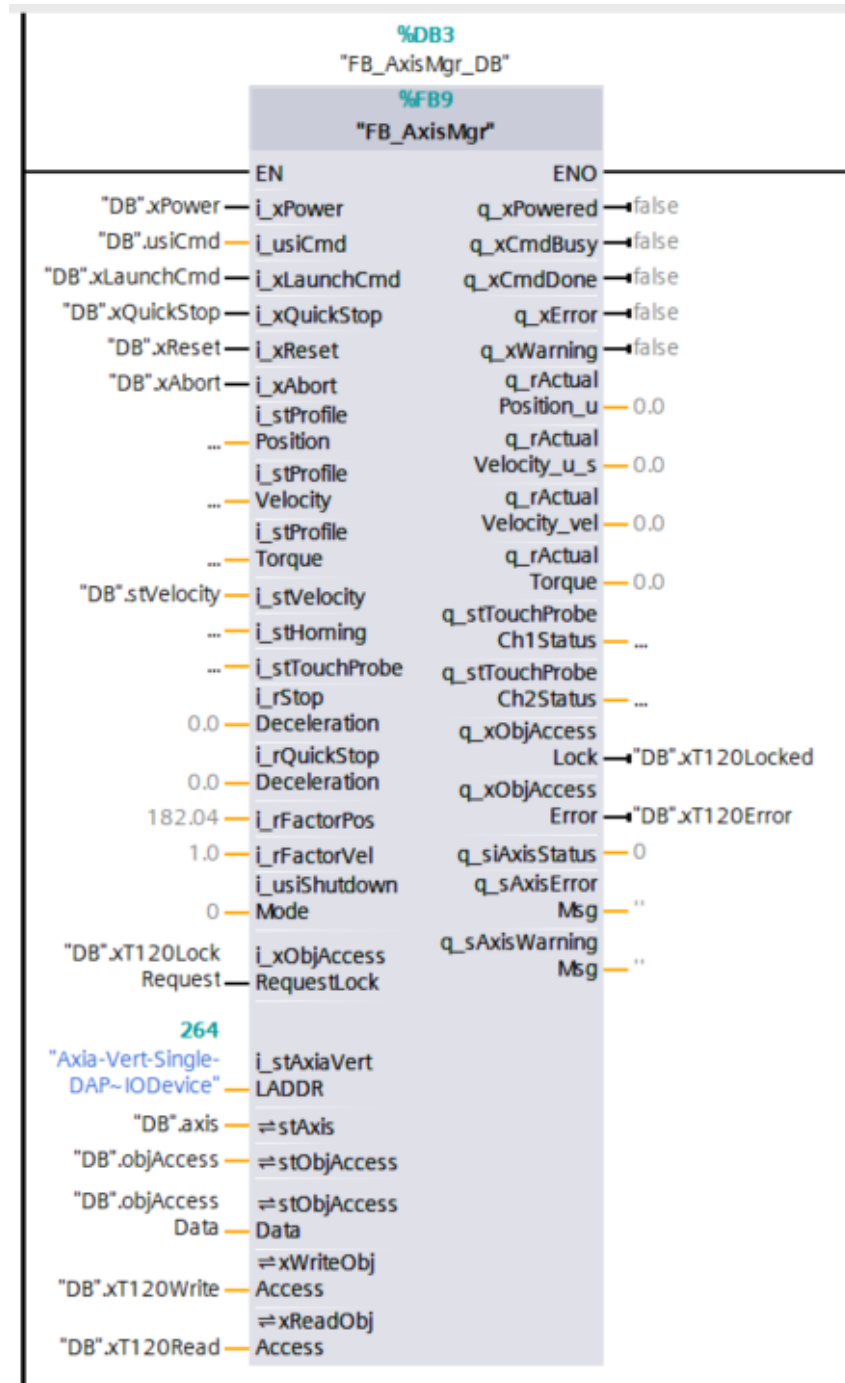
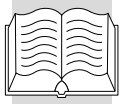


Figure 42: Velocity mode example - FB\_AxisMgr



## 7.3 SCL Program – Velocity Mode

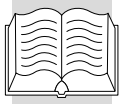
Create the following SCL Program to control the axis in Velocity Mode:

```
//In case of a fault reset the state machine
//"DB".xReset must be set to 1 to reset the fault
IF "FB_AxisMgr_DB_1".q_xError THEN
    "DB".i := 0;
    "DB".l_xStart := FALSE;
    "DB".xLaunchCmd := FALSE;
    RETURN;
END_IF;

//Reset "DB".xReset
IF NOT "FB_AxisMgr_DB_1".q_xError AND "DB".xReset THEN
    "DB".xReset := FALSE;
END_IF;

CASE "DB".i OF
    0:
        "DB".xPower := FALSE;
        IF "DB".l_xStart THEN
            //Power on the axis
            "DB".xPower := TRUE;
            IF "FB_AxisMgr_DB_1".q_xPowered THEN
                "DB".i := 5;
            END_IF;
        END_IF;
    5:
        //Set velocity parameters
        "DB".stVelocity.rAcceleration := 1000.0;
        "DB".stVelocity.rDeceleration := 1000.0;
        "DB".stVelocity.rVelocityWindow := 30;
        "DB".stVelocity.tVelocityWindowTime := T#500ms;
        "DB".stVelocity.rTargetVelocity := 180.0;

        //Launch the velocity command
        "DB".usiCmd := 51;
        "DB".xLaunchCmd := TRUE;
        IF "FB_AxisMgr_DB_1".q_xCmdBusy THEN
            "DB".i := 10;
        END_IF;
    10:
        //Axis in velocity mode. You can change velocity parameters
        "DB".xLaunchCmd := FALSE;
        IF NOT "DB".l_xStart THEN
            "DB".usiCmd := 20;
            "DB".xLaunchCmd := TRUE;
            "DB".i := 15;
        END_IF;
    15:
        IF "FB_AxisMgr_DB_1".q_xCmdDone THEN
            "DB".xLaunchCmd := FALSE;
            "DB".xPower := FALSE;
            "DB".i := 20;
        END_IF;
    20:
        IF "FB_AxisMgr_DB_1".q_xCmdDone THEN
            "DB".i := 0;
        END_IF;
END_CASE;
```



```
1 //In case of a fault reset the state machine
2 //"DB".xReset must be set to 1 to reset the fault
3 IF "FB_AxisMgr_DB_1".q_xError THEN
4     "DB".i := 0;
5     "DB".l_xStart := FALSE;
6     "DB".xLaunchCmd := FALSE;
7     RETURN;
8 END_IF;
9
10 //Reset "DB".xReset
11 IF NOT "FB_AxisMgr_DB_1".q_xError AND "DB".xReset THEN
12     "DB".xReset := FALSE;
13 END_IF;
14
15 CASE "DB".i OF
16     0:
17         "DB".xPower := FALSE;
18         IF "DB".l_xStart THEN
19             //Power on the axis
20             "DB".xPower := TRUE;
21             IF "FB_AxisMgr_DB_1".q_xPowered THEN
22                 "DB".i := 5;
23             END_IF;
24         END_IF;
25     5:
26         //Set velocity parameters
27         "DB".stVelocity.rAcceleration := 1000.0;
28         "DB".stVelocity.rDeceleration := 1000.0;
29         "DB".stVelocity.rVelocityWindow := 30;
30         "DB".stVelocity.tVelocityWindowTime := T#500ms;
31         "DB".stVelocity.rTargetVelocity := 180.0;
32
33         //Launch the velocity command
34         "DB".usiCmd := 51;
35         "DB".xLaunchCmd := TRUE;
36         IF "FB_AxisMgr_DB_1".q_xCmdBusy THEN
37             "DB".i := 10;
38         END_IF;
39     10:
40         //Axis in velocity mode. You can change velocity parameters
41         "DB".xLaunchCmd := FALSE;
42         IF NOT "DB".l_xStart THEN
43             "DB".usiCmd := 20;
44             "DB".xLaunchCmd := TRUE;
45             "DB".i := 15;
46         END_IF;
47     15:
48         IF "FB_AxisMgr_DB_1".q_xCmdDone THEN
49             "DB".xLaunchCmd := FALSE;
50             "DB".xPower := FALSE;
51             "DB".i := 20;
52         END_IF;
53     20:
54         IF "FB_AxisMgr_DB_1".q_xCmdDone THEN
55             "DB".i := 0;
56         END_IF;
57 END_CASE;
```

Figure 43: Velocity mode example - SCL Program – Velocity Mode



- ➔ Set *DB.l\_xStart* to TRUE to start the motor.
- ➔ Set *DB.l\_xStart* to FALSE to stop the motor.



## 7.1 SCL Program – Object Access

Create the following SCL Program to read and write objects with Object Access channel (e.g. via Telegram 120).

```
//Read
IF "DB".l_xReadObj OR ("DB".j > 0 AND "DB".j < 15) THEN
  CASE "DB".j OF
    0:
      //Request access
      "DB".xTl20LockRequest := TRUE;
      //Access granted
      IF "DB".xTl20Locked THEN
        "DB".j := 5;
      END_IF;
    5:
      //Read object 0x6048/02 (Acceleration Delta Time)
      "DB".objAccessData.uiIndex := 16#6048;
      "DB".objAccessData.bSubIdx := 16#02;
      "DB".objAccessData.siDataType := "FB_B_AxisMgr_Constants_DB".OBJACCESS_UINT16;
      //Read command
      "DB".xTl20Read := TRUE;

      "DB".j := 10;
    10:
      //Object read
      IF NOT "DB".xTl20Read AND NOT "DB".xTl20Error THEN
        //Read data
        "DB".l_uiAccTime := UDINT_TO_UINT("DB".objAccessData.udiData);
        //Release
        "DB".xTl20LockRequest := FALSE;
        "DB".l_xReadObj := FALSE;

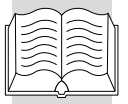
        "DB".j := 15;
      //Read error
      ELSIF NOT "DB".xTl20Read AND "DB".xTl20Error THEN
        //Release
        "DB".xTl20LockRequest := FALSE;
        "DB".l_xReadObj := FALSE;

        "DB".j := 15;
      END_IF;
    END_CASE;
  ELSIF "DB".l_xWriteObj OR ("DB".k > 0 AND "DB".k < 15) THEN
    CASE "DB".k OF
      0:
        //Request access
        "DB".xTl20LockRequest := TRUE;
        //Access granted
        IF "DB".xTl20Locked THEN
          "DB".k := 5;
        END_IF;
      5:
        //Write 10000 into object 0x6048/01 (Acceleration Delta Speed)
        "DB".objAccessData.uiIndex := 16#6048;
        "DB".objAccessData.bSubIdx := 16#01;
        "DB".objAccessData.udiData := 10000;
        "DB".objAccessData.siDataType := "FB_B_AxisMgr_Constants_DB".OBJACCESS_UINT32;
        //Write command
        "DB".xTl20Write := TRUE;

        "DB".k := 10;
      10:
        //Object written sucesfully
        IF NOT "DB".xTl20Write AND NOT "DB".xTl20Error THEN
          //Release
          "DB".xTl20LockRequest := FALSE;
          "DB".l_xWriteObj := FALSE;

          "DB".k := 15;
        //Write error
        ELSIF NOT "DB".xTl20Write AND "DB".xTl20Error THEN
          //Release
          "DB".xTl20LockRequest := FALSE;
          "DB".l_xWriteObj := FALSE;

          "DB".k := 15;
        END_IF;
      END_CASE;
    END_IF;
  END_IF;
```



```
        END_IF;  
    END_CASE;  
  
ELSE  
    "DB".j := 0;  
    "DB".k := 0;  
    "DB".xT120LockRequest := FALSE;  
END_IF;
```

```
1 //Read  
2 IF "DB".l_xReadObj OR ("DB".j > 0 AND "DB".j < 15) THEN  
3 CASE "DB".j OF  
4  
5     //Request access  
6     "DB".xT120LockRequest := TRUE;  
7     //Access granted  
8     IF "DB".xT120Locked THEN  
9         "DB".j := 5;  
10    END_IF;  
11  
12    5:  
13    //Read object 0x048/02 (Acceleration Delta Time)  
14    "DB".objAccessData.uiIndex := 16#048;  
15    "DB".objAccessData.bSubIdx := 16#02;  
16    "DB".objAccessData.siDataType := "FB_B_AxisMgr_Constants_DB".OBJACCESS_UINT16;  
17    //Read command  
18    "DB".xT120Read := TRUE;  
19  
20    "DB".j := 10;  
21  
22    10:  
23    //Object read  
24    IF NOT "DB".xT120Read AND NOT "DB".xT120Error THEN  
25        //Read data  
26        "DB".l_uiAccTime := UDINT_TO_UINT("DB".objAccessData.usiData);  
27        //Release  
28        "DB".xT120LockRequest := FALSE;  
29        "DB".l_xReadObj := FALSE;  
30  
31        "DB".j := 15;  
32        //Read error  
33        ELIF NOT "DB".xT120Read AND "DB".xT120Error THEN  
34            //Release  
35            "DB".xT120LockRequest := FALSE;  
36            "DB".l_xReadObj := FALSE;  
37  
38            "DB".j := 15;  
39        END_IF;  
40    END_CASE;  
41    ELIF "DB".l_xWriteObj OR ("DB".k > 0 AND "DB".k < 15) THEN  
42    CASE "DB".k OF  
43  
44        //Request access  
45        "DB".xT120LockRequest := TRUE;  
46        //Access granted  
47        IF "DB".xT120Locked THEN  
48            "DB".k := 5;  
49        END_IF;  
50  
51        5:  
52        //Write 10000 into object 0x045/01 (Acceleration Delta Speed)  
53        "DB".objAccessData.uiIndex := 16#045;  
54        "DB".objAccessData.bSubIdx := 16#01;  
55        "DB".objAccessData.siDataType := "FB_B_AxisMgr_Constants_DB".OBJACCESS_UINT32;  
56        //Write command  
57        "DB".xT120Write := TRUE;  
58  
59        "DB".k := 10;  
60  
61        10:  
62        //Object written successfully  
63        IF NOT "DB".xT120Write AND NOT "DB".xT120Error THEN  
64            //Release  
65            "DB".xT120LockRequest := FALSE;  
66            "DB".l_xWriteObj := FALSE;  
67  
68            "DB".j := 15;  
69            //Write error  
70            ELIF NOT "DB".xT120Write AND "DB".xT120Error THEN  
71                //Release  
72                "DB".xT120LockRequest := FALSE;  
73                "DB".l_xWriteObj := FALSE;  
74  
75                "DB".j := 15;  
76            END_IF;  
77        END_CASE;  
78    ELSE  
79        "DB".j := 0;  
80        "DB".xT120LockRequest := FALSE;  
81    END_IF;
```

Figure 44: Velocity mode example - SCL Program – Object access



- ➔ Set `DB.l_xReadObj` to TRUE to read objects.
- ➔ Set `DB.l_xWriteObj` to TRUE to write objects.



## 8 List of Figures

Figure 1: Axis Manager State Diagram .....	14
Figure 2: FB_AxisMgr .....	16
Figure 3: State transitions from Disabled to Standstill .....	20
Figure 4: Shutdown Modes .....	21
Figure 5: Launching Commands .....	22
Figure 6: Example of PROFINet® map .....	25
Figure 7: PROFINet® Velocity mode mapping example – Drive .....	26
Figure 8: PROFINet® Velocity mode mapping example - AxiaManager configuration .....	26
Figure 9: PROFINet® Velocity mode mapping example -Tag definition .....	27
Figure 10: PROFINet® Velocity mode mapping example - DB definition .....	28
Figure 11: Velocity mode mapping example - PROFINet® map .....	28
Figure 12: Example of PROFIBus® map .....	29
Figure 13: PROFIBus® Velocity mode mapping example – Drive .....	30
Figure 14: PROFIBus® Velocity mode mapping example - AxiaManager configuration .....	30
Figure 15: PROFIBus® Velocity mode mapping example -Tag definition .....	31
Figure 16: PROFIBus® Velocity mode mapping example - DB definition .....	32
Figure 17: Velocity mode mapping example - PROFIBus® map .....	32
Figure 18: FB_ETrigA .....	35
Figure 19: FB_ETrigA State Machine .....	35
Figure 20: FB_ETrigA Normal Operation Sequence .....	37
Figure 21: FB_ETrigA Error Handling .....	38
Figure 22: FB_ETrigA Abort Handling .....	38
Figure 23: FB_MCHardwareError .....	41
Figure 24: FB_MCHoming .....	43
Figure 25: FB_MCPower .....	46
Figure 26: FB_MCProfilePosition .....	48
Figure 27: FB_MCProfileTorque .....	52
Figure 28: FB_MCProfileVelocity .....	54
Figure 29: FB_MCReset .....	56
Figure 30: FB_MCTouchProbe .....	57
Figure 31: FB_MCVelocity .....	59
Figure 32: FB_Mutex_ObjAccess .....	62
Figure 33: FB_ObjAccess .....	63
Figure 34: External Access to Object Access .....	64
Figure 35: FB_ObjAccessMap .....	67
Figure 36: FB_PROFINetMap .....	68
Figure 37: FB_PROFIBusMap .....	70
Figure 38: Homing Modes .....	75
Figure 39: Objecy Access structure .....	83
Figure 40: Library - Example .....	88
Figure 41: Velocity mode example - variables definition .....	89
Figure 42: Velocity mode example - FB_AxisMgr .....	91
Figure 43: Velocity mode example - SCL Program – Velocity Mode .....	93
Figure 44: Velocity mode example - SCL Program – Object access .....	95





---

## 9 References

AXIA VERT - Operating Instructions

AXIA VERT - Communication Module Manual CMA-IE-01 for PROFINET®

AXIA VERT - Communication Module Manual CMA-PB-01 for PROFIBUS®



## 10 REVISION INDEX (R)

BR_IOM_AXMG-SIE_STD_ENG_R01_0			
Revision	Change	Date	Author
1.0	First draft	01/08/2025	R. Zandonati

This publication supersedes and replaces any previous edition and revision. We reserve the right to implement modifications without notice.  
This catalogue cannot be reproduced, even partially, without prior consent.

*To the extent permitted by the applicable law, the Seller shall not assume any liability, for any legal reason whatsoever, including, without limitation, liability for damages to the integrated system, business interruption, reduced usability, availability and completeness, loss of profits, loss of stored information or other economic losses arising from the use of the Software, even if the Buyer has been advised of the possibility of such damages.*

*In any case, and always to the extent permitted by law, the Seller's liability to the Buyer shall be limited to an amount corresponding to the price paid for the purchase of the Software, which the parties predetermine as a penalty in accordance with article 1382 of Italian Civil Code.*

*By using the application examples the Buyer acknowledges that the Seller cannot be held liable for any damage beyond the liability provisions described above.*



We have a relentless commitment to excellence, innovation & sustainability. Our team creates, distributes and services world-class power transmission & drive solutions to keep the world in motion.

#### HEADQUARTERS

##### **Bonfiglioli S.p.A**

Via Cav. Clementino Bonfiglioli, 1  
40012 Calderara di Reno - Bologna (Italy)  
Tel. +39 051 6473111

