

ctrlX DRIVE

Technology Function

Copyright

© Bosch Rexroth AG 2024

All rights reserved, also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.

Disclaimer

The data specified above only serve to describe the product. As our products are constantly being further developed, no statements concerning a certain condition or suitability for a certain application can be derived from our information. The information given does not release the user from the obligation of own judgment and verification. It must be remembered that our products are subject to a natural process of wear and aging.

Table of contents

1	About this documentation	6
2	Product presentation	7
2.1	What is the ctrlX DRIVE Technology Function?	7
2.2	Technical data	8
2.3	PLC in accordance with IEC 61131-3	8
2.4	Physical interfaces	9
2.5	Byte order	9
3	Basic terms used in PLC programming	9
4	Requirements for using ctrlX DRIVE Technology Function	10
4.1	Hardware requirements	10
4.2	Firmware requirements	10
4.3	Software requirements	11
4.4	Requirements for communication	11
5	Notes on commissioning and application	11
5.1	Requirements for programming Technology Apps for ctrlX DRIVE	11
5.2	Installing Engineering tools on the PC	12
5.3	Package installation ctrlX DRIVE	15
5.4	Enabling the functional package "ctrlX DRIVE Technology Function"	18
5.5	Creating a project for ctrlX DRIVE	19
5.6	Example programs for ctrlX DRIVE	20
5.7	Establishing the communication to the ctrlX DRIVE target	21
6	Task system	21
6.1	Definitions of terms	21
6.2	Brief description	22
6.3	Task properties	22
6.4	Task monitoring (watchdog)	23
6.5	Task stack	25
6.6	Reading and setting the task interval time	25
6.7	Runtime behavior	26
6.7.1	Time slot method	26
6.7.2	Ensured computing time	27
6.7.3	Runtime measurement	27
7	Access options of the ctrlX DRIVE Technology Function to the ctrlX DRIVE Runtime	31
7.1	Introduction	31
7.2	Acyclic data exchange	32
7.2.1	Access via direct variables	32
7.2.2	Access via function blocks	34
7.2.3	PLC parameters for general purpose (global PLC registers)	34
7.2.4	Access to digital and analog inputs and outputs	36
8	Axis control	37
8.1	Libraries for axis control	37
8.2	Basic principles of axis control with the integrated PLC (ctrlX DRIVE Technology Function)	38
8.3	Commanding the axis	39

9	Diagnostic handling of ctrlX DRIVE Technology Function	41
9.1	Brief description and overview	41
9.2	Diagnostics handled in the drive	42
9.3	Errors that can occur at runtime due to faulty programming	42
9.4	PLC system error	43
9.5	Errors detected by functions or function blocks	43
9.6	Notes on application and programming	44
10	External PLC communication	44
11	Programming information	46
11.1	Safe programming	46
11.2	Working with retain variables	47
11.3	Libraries for ctrlX DRIVE Technology Function	54
11.3.1	General properties of the libraries	54
11.3.2	Supported libraries	55
11.3.3	Documentation on the libraries	55
11.4	Start behavior and boot project	56
11.4.1	General information	56
11.4.2	Brief description	57
11.4.3	PLC behavior after a processor exception	57
11.5	Guidelines for programming with ctrlX PLC Engineering	57
11.5.1	Overview	57
11.5.2	Compatibility of ctrlX DRIVE Technology Function with the drive firm- ware	58
11.5.3	Structuring PLC projects	58
	Multi-program technology	58
	PLC objects	59
11.5.4	Code distribution to "POU Locations"	59
11.5.5	Global data	60
11.5.6	Code optimization	61
11.5.7	Library template and style guide (PLC programming guidelines) ...	64
12	Technology Apps	65
12.1	Using Technology Apps as a self-contained function or as a library	65
12.2	Loading Technology Apps	65
12.3	Technology App ID	65
13	Diagnostic and service functions	67
13.1	Overview and introduction	67
13.2	Device editor	67
13.2.1	Device editor, general information	67
13.2.2	Communication	67
13.2.3	Applications	68
13.2.4	Log	68
13.2.5	PLC shell	68
13.2.6	Information	68
13.3	Diagnostic functions	68
13.3.1	Standard drive diagnostic functions	68
	Introduction	68
	Oscilloscope function	69

	Analog output	69
	Diagnostic trace	69
	Panel	69
13.3.2	Recording PLC variables with the oscilloscope function	69
13.3.3	PLC diagnostic functions	70
	Overview	70
	Recipe manager	71
	Visualization	71
	Monitoring functions	71
	Simulation	71
	Debugging / troubleshooting	71
	PLC control and status words	71
	Tracing PLC exceptions	73
13.4	Service functions	75
13.4.1	PLC service functions	75
	Service function in ctrlX PLC Engineering	75
	PLC behavior after a processor exception	75
13.4.2	Firmware replacement	75
14	Documentations	76
14.1	Drive systems, system components	76
14.2	Firmware/Runtime	76
14.3	Functional safety	78
14.4	Motors	78
14.5	Cables	79
15	Service and support	80
16	Glossary	81
17	Index	83

1 About this documentation

This documentation describes the ctrlX DRIVE Technology Function, the PLC firmware function for ctrlX DRIVE.

Editions of this documentation

Edition	Release date	Comment
01	2021-07-26	First edition
02	2022-07-12	• Added chapter "Working with retain variables" • Added chapter "Technology Apps" • Chapter "Task system" – Included information on "PositionLoop Event" – Included information on ensured computing time and task runtime measurement • Changes in chapter structures • In ↪ Chapter 8 Axis control on page 37, included more detailed information on how to command the axis
03	2023-07-10	• Development of individual PLC programs from AXS-V-0316 for ctrlX DRIVE possible with license "TE0" • Package installation with ctrlX PLC Engineering described from version 1.18.x • Task stack: As of device version 4.4.0.0, the maximum stack requirement can be specified using a pragma • Chapter "Free PLC parameters (global PLC registers)" was revised • Axis command overview graphic inserted
04	2024-09-13	• Modified information on symbol configuration • Added information on status tasks • Added information on asynchronous communication with UDP • Added information on how to exchange network variables of multi-axis devices between the axes • Regarding the processing of strings, reference to the optimized functions from the "CXA_Uilities" library • Tracing PLC exceptions: Added procedure for an exception after the drive was switched off • Described service function "Generate code" and "Clean all" of ctrlX PLC Engineering • Described download of missing libraries • Updated "↪ Chapter 5.2 Installing Engineering tools on the PC on page 12" • Added information on how to record PLC variables with the oscilloscope function



Feedback on this documentation

Your experience is an important part of the product and documentation improvement process.

In case of any errors or if you want to suggest changes to this documentation, please do not hesitate to contact us.

Please send your feedback to [↗dokusupport@boschrexroth.de](mailto:dokusupport@boschrexroth.de).

2 Product presentation

2.1 What is the ctrlX DRIVE Technology Function?

The ctrlX DRIVE Technology Function is the PLC firmware function that allows customized PLC programs or ready-made Technology Apps to be used in the axis processor of the ctrlX DRIVE drive system.

Main features of ctrlX DRIVE Technology Function

The runtime system of the drive-integrated PLC (ctrlX DRIVE Technology Function) is part of the drive firmware with its functions and interfaces.

The control options of the drive-integrated PLC are device-internally limited to the axis.

These are the main features of the ctrlX DRIVE Technology Function:

- ctrlX DRIVE Technology Function is perfect for economically solving logic and additional control tasks using firmware functions of the drive
- Programming functional enhancements in accordance with IEC 61131-3 based on comprehensive standard libraries and additional libraries ("technology libraries")
- Commanding the axis with access to the device control of the drive
- Direct access to all drive parameters via direct variables, functions or function blocks
- Access to digital and analog inputs/outputs of the drive via parameters

2.2 Technical data

ctrlX DRIVE Technology Function has the following resources:

Internal code memory, gross. (program, constants, administrative data...)	384 kB in the fast internal memory („area_0“, „memoryrange0“) 1 MB in the regular memory („area_1“, „memoryrange1“)
Storage of the boot project Note: The project storage is also limited by the internal code memory	7 × 64 kB compressed (P-0-1352ff)
Memory effectively usable for a boot project (code, administration, symbols...)	typically 1404 kB
Data memory (variables, instances, administrative data...)	1 MB
Retain memory	248 byte
Process input image (%I)	No process image but direct parameter access.
Process output image (%Q)	No process image but direct parameter access.
Memory storage (%M)	-
Maximum task number	3
PositionLoopEvent	available with a cycle time of 250 µs
PLC register, 4 byte, remanent	32 (P-0-1370...P-0-1385, P-0-1316...P-0-1331)
PLC register, 4 byte, volatile	32 (P-0-1270...P-0-1301)
PLC list register, 4 kB, remanent	3 (P-0-1311, P-0-1312, P-0-1389)
PLC list register, 32 kB, volatile	2 (P-0-1368, P-0-1369)
PLC text register	2 (P-0-1387, P-0-1388)
PLC register, 4 byte, volatile	40 (P-0-1390...P-0-1429)
PLC register, 2 byte, volatile	8 (P-0-1440...P-0-1447)
Semaphores (SysSemScreate())	16
Events (SysEventCreate())	8
File handles	-
Directory handles	-
Sockets	- (Ethernet programming not yet released)
Task stack (without subordinate firmware calls)	7266 byte
Task stack (with subordinate firmware calls)	4450 byte
Symbol configuration	Typically approx. 8000 (with a compact PLC project)

The internal code memory can be used completely for typical PLC projects. The symbol administration is stored in the data and code memory.

The system memory is available to the system for allocation of dynamic memory blocks for, e.g., the communication. The system memory is not intended to be used by the user.

2.3 PLC in accordance with IEC 61131-3

The PLC in ctrlX DRIVE is based on the CODESYS runtime system Control V3. The PLC is optimally integrated in the ctrlX DRIVE system environment. It processes the code programmed in IEC 61131-3 and is responsible for debugging with ctrlX PLC Engineering.

2.4 Physical interfaces

Inputs and outputs

In ctrlX DRIVE, different digital and analog inputs and outputs are available depending on the device configuration. (See also Project Planning Manual)

Engineering interfaces

ctrlX DRIVE features an Ethernet interface. The interface support PLC programming with ctrlX PLC Engineering and the configuration with ctrlX DRIVE Engineering.

Panel/USB interface

Optionally, a panel can be connected. The USB interface of the panel supports PLC programming as network adapter with ctrlX PLC Engineering and the configuration with ctrlX DRIVE Engineering.

2.5 Byte order

The AXS firmware uses the byte order "Little Endian".

3 Basic terms used in PLC programming



Some basic terms used in PLC programming are listed below; for other basic terms, refer to the documentation "ctrlX PLC Engineering, Application Manual"

Resources

Resources are objects for organizing a project, tracing variable values and configuring the project for use on the target and in the network (global variables, PLC configuration, task configuration, recipe manager).

Sampling trace

Sampling trace is not available in ctrlX DRIVE Technology Function.

PLC configuration

PLC configuration is not possible in ctrlX DRIVE Technology Function. Access to digital and analog inputs/outputs of ctrlX DRIVE is only possible via parameters, see [Access to digital and analog inputs and outputs](#)

Task configuration

It is possible but not mandatory to control project processing (control of the program) via tasks. It is recommended to create a task configuration in ctrlX DRIVE Technology Function.

Symbol configuration

The symbol configuration is used to provide symbols with certain access rights to project variables. Use these symbols to externally access variables, e.g. via an OPC server.

Recipe manager

The recipe manager provides functions to manage user-defined variable lists, so-called recipe definitions.

Task

A task is a time unit in the processing of the IEC program. It is defined by a name, a priority and a type. The type defines which condition triggers the start of the task. This condition can either be defined by time (cyclic interval, freewheeling) or by an internal event that is to trigger the task, e.g., the rising edge of a global project variable.

Function

A function has the following properties:

- A function returns exactly one piece of data (which can also contain multiple elements, such as arrays or structures) when it is executed.
- A function can be called in textual languages as an operand used in expressions.
- A function has no "memory"; i.e., the contents of variables are not stored.
- A function does not need an instance.

Function block

A function block has the following properties:

- A function block can return multiple values.
- A function block has a "memory"; i.e., the contents of variables are stored.
- A function block requires an instance to be specified.

Programs

Programs are known globally in the entire project. Programs are not instantiated. The variables of the program only exist once. Programs can be called in a task. A task can call multiple programs and a program can call functions, function blocks or other programs.

Boot project

The boot project is a project loaded on the target (in this case a drive controller of the ctrlX DRIVE range with the PLC firmware function for ctrlX DRIVE Technology Function) and started when booting up the drive controller. The boot project is automatically generated upon each download.

4 Requirements for using ctrlX DRIVE Technology Function

4.1 Hardware requirements

For ctrlX DRIVE Technology Function, a drive controller ctrlX DRIVEplus with license "TF1", "TE1", or "TX1" or a drive controller ctrlX DRIVE with license "TE0" is required; an additional hardware option is not required.

4.2 Firmware requirements

The following firmware requirements are necessary for ctrlX DRIVE Technology Function:

- Firmware AXS-V-0302 and above
 - "TF1" license when using a Technology App as a self-contained technology function
 - License "TE1", "TE0" (only from AXS-V-0316) or "TX1" when developing individual PLC programs
 - "TX1" license for enabling Technology Apps by Bosch Rexroth as a library

The functional package ctrlX DRIVE Technology Function must have been activated (see [↗ Enabling the functional package ctrlX DRIVE Technology Function](#)).

4.3 Software requirements

When installing ctrlX WORKS, the development system for PLC applications ctrlX PLC Engineering is provided for installation.

In detail, ctrlX PLC Engineering consists of the following components:

- PLC programming interface
The PLC programming interface contains the PLC editors and debuggers.
- ctrlX PLC Gateway
"ctrlX PLC Gateway" provides the communication for the programming system and for applications communicating with the control via the COD-ESYS protocol.

The device support for ctrlX DRIVE Technology Function has to be installed separately. For this purpose, install the "ctrlX_DRIVE_TF_V.x.x.x.package" in ctrlX PLC Engineering (see [↗ Making ctrlX DRIVE available as programmable device](#)). The package contains the device description file and the required libraries.



When updating ctrlX PLC Engineering, all installed libraries and device description files remain available on the computer. It is not necessary to install the ctrlX DRIVE Technology Function package again.

4.4 Requirements for communication

When exclusively using a ready-made Technology App (in the form of a ready-made parameter file) with the drive-integrated PLC ("TF1" license), it is not necessary to establish a connection between ctrlX PLC Engineering and the drive. Only a communication with ctrlX DRIVE Engineering is required to configure the drive.

If you want to use individual functions, a connection to the drive with ctrlX PLC Engineering is required for programming the drive-integrated PLC (see [↗ Physical interfaces](#)).

Establishing a connection with connected devices is supported by ctrlX DRIVE Engineering. Use the menu bar to select *"Tools → Connection"*.

5 Notes on commissioning and application

5.1 Requirements for programming Technology Apps for ctrlX DRIVE

To program Technology Apps for ctrlX DRIVE, the following requirements must have been fulfilled:

- [↗ The Engineering and programming tools must have been installed on the Engineering PC](#)
- [↗ The functional package "ctrlX DRIVE Technology Function" must have been enabled](#)
- [↗ The ctrlX DRIVE package must have been installed](#)

5.2 Installing Engineering tools on the PC

This section describes how to install the Engineering tools required for using "Technology Function". The following software is required:

- [ctrlX WORKS](#) ⁺
- [ctrlX DRIVE Engineering](#) ⁺
- [ctrlX PLC Engineering](#) ⁺

Installing ctrlX WORKS

Proceed as follows when installing ctrlX WORKS:

1.  Download the installation files of ctrlX WORKS (see [➔ "FAQ for ctrlX WORKS", "Where can I get ctrlX WORKS?"](#)).
2.  To start the installation, execute the setup file ("ctrlx-works-xxxx.exe") of ctrlX WORKS (admin rights required).
 - ➔ The user account control is displayed and you are asked whether to allow the "PackageManager" to be executed.
3.  Allow the "PackageManager" to be executed.
 - ➔ The welcome screen of the ctrlX WORKS installation is displayed.
4.  Select the installation language and confirm the dialog with "Next".
 - ➔ The terms of use of Bosch Rexroth AG are displayed.
5.  Read the terms of use. To confirm the terms of use, start the installation with "Accept".
 - ➔ The dialog for selecting the installation target directory is opened.
6.  Specify the directory for the ctrlX WORKS installation and confirm the dialog with "Next".
 - ➔ The dialog for selecting the installation options is displayed.
7.  Using "Technology Function" only requires the ctrlX WORKS option. You may deselect all other options.
Confirm your selection with "Next".
 - ➔ A list of the required software packages is identified and displayed.
8.  Start the installation with "Install"
 - ➔ The installation is started; it may take some minutes and the installation progress is displayed visually.
The installation result is displayed.
9.  To complete the installation, select "Finish".

Installing the Engineering tools

Proceed as follows to install the Engineering tools:

1. ➤ Start ctrlX WORKS.
2. ➤ On the sidebar click "Settings"
 - ➔ The "Settings" window opens. This window contains the settings for ctrlX WORKS.
3. ➤ Using "Technology Function" requires the Engineering tools ctrlX DRIVE Engineering and ctrlX PLC Engineering.
Click the "Setup Engineering Tools" tile.
 - ➔ The "Setup Engineering Tools" window opens. This window displays the Engineering tools already installed and those available for installation. If required, Engineering tools already installed may be updated in this window, insofar as a new version of these tools is available for installation. For the initial installation of ctrlX DRIVE Engineering click the corresponding tile, and in the following view click "Install".
 - ➔ The user account control is displayed and you are asked whether to allow the "PackageManager" to be executed.
4. ➤ Allow the "PackageManager" to be executed.
 - ➔ The welcome screen of the ctrlX DRIVE Engineering installation is displayed.
Note: By default, the "Run setups in the background" option is active and ensures that only the query for the "administrator mode" has to be confirmed when a setup is run. Everything else is run in the background. (Does not apply to the initial installation of a tool. In this case, it is required to select an installation directory.). Only if the "Run setups in the background" option has been deactivated, will the user interface of the tool setup (steps 5-10) appear with every update.
5. ➤ Select the installation language and confirm the dialog with "Next".
 - ➔ The terms of use of Bosch Rexroth AG are displayed.
6. ➤ Read the terms of use. To confirm the terms of use, start the installation with "Accept".
 - ➔ The dialog for selecting the installation target directory is opened.
7. ➤ Specify the directory for the ctrlX DRIVE Engineering installation and confirm the dialog with "Next".
 - ➔ The dialog for selecting the installation options is displayed.
8. ➤ In the "Functions / ctrlX DRIVE system" section, select the "Commissioning and diagnostics" option, and in "Functions / Service", select the "Device data sheets for ctrlX DRIVE" option. Other options are not required for using "Technology Function". In the "Languages" section, you may additionally select the languages to be installed. Confirm your selection with "Next".
 - ➔ A list of the required software packages is identified and displayed.

9. Start the installation with "Install".
 - ➔ The installation is started; it may take some minutes and the installation progress is shown visually. In case ctrlX DRIVE Engineering is installed for first time, a query regarding the installation of the device software "Robert Bosch GmbH Network Adapter" is displayed during the installation. If you use ctrlX DRIVE with ctrlX DRIVE Panel and would like to use it to connect the PC to the drive controller, confirm the installation of the device software with "Install". The installation result is displayed.
10. To complete the installation, select "Finish".
11. Click the tile of ctrlX PLC Engineering and in the following view click "Install".
 - ➔ The user account control is displayed and you are asked whether to allow the "PackageManager" to be executed.
12. Allow the "PackageManager" to be executed.
 - ➔ The welcome screen of the ctrlX PLC Engineering installation is displayed.

Note: By default, the "Run setups in the background" option is active and ensures that only the query for the "administrator mode" has to be confirmed when a setup is run. Everything else is run in the background. (Does not apply to the initial installation of a tool. In this case, it is required to select an installation directory.). Only if the "Run setups in the background" option has been deactivated, will the user interface of the tool setup (steps 13-18) appear with every update.
13. Select the installation language and confirm the dialog with "Next".
 - ➔ The terms of use of Bosch Rexroth AG are displayed.
14. Read the terms of use. To confirm the terms of use, start the installation with „Accept“.
 - ➔ The dialog for selecting the installation target directory is opened.
15. Specify the directory for the ctrlX PLC Engineering installation and confirm the dialog with "Next".
 - ➔ The dialog for selecting the installation options is displayed.
16. The selection of the ctrlX PLC Engineering option is mandatory. Confirm your selection with "Next".
 - ➔ A list of the required software packages is identified and displayed.
17. Start the installation with "Install".
 - ➔ The installation is started; it may take some minutes and the installation progress is shown visually. The installation result is displayed.
18. To complete the installation, select "Finish".

5.3 Package installation ctrlX DRIVE

To program a Technology App on ctrlX DRIVE, a package has to be installed in ctrlX PLC Engineering. With the installation of the package, the device description and the required libraries are installed.

This section describes how to install the package; 2 different cases are described:

- Installing the package with ctrlX PLC Engineering version 1.16.x and below
- Installing the package with ctrlX PLC Engineering version 1.18.x and above

Installing the package with ctrlX PLC Engineering version 1.16.x and below

Requirements

- ctrlX PLC Engineering ($\leq$ version 1.16.x) must have been installed.
- The package *ctrlX_DRIVE_TF_Vx.x.x.x.package* has to be available on the PC.

1.  Start ctrlX PLC Engineering.
2.  Open the “Tools → Package Manager...” menu
 - ➔ The “Package Manager” dialog is opened.
3.  Click the “Install...” button
 - ➔ The dialog for selecting the package is opened.
4.  Navigate to the directory that contains the package *ctrlX_DRIVE_TF_Vx.x.x.x.package*, and select the package. Click the “Open...” button.
 - ➔ The “Choose Setup Type” dialog is opened.
5.  Select the option “Typical setup”.
 - ➔ The standard set of device description file and libraries defined in the package is installed.

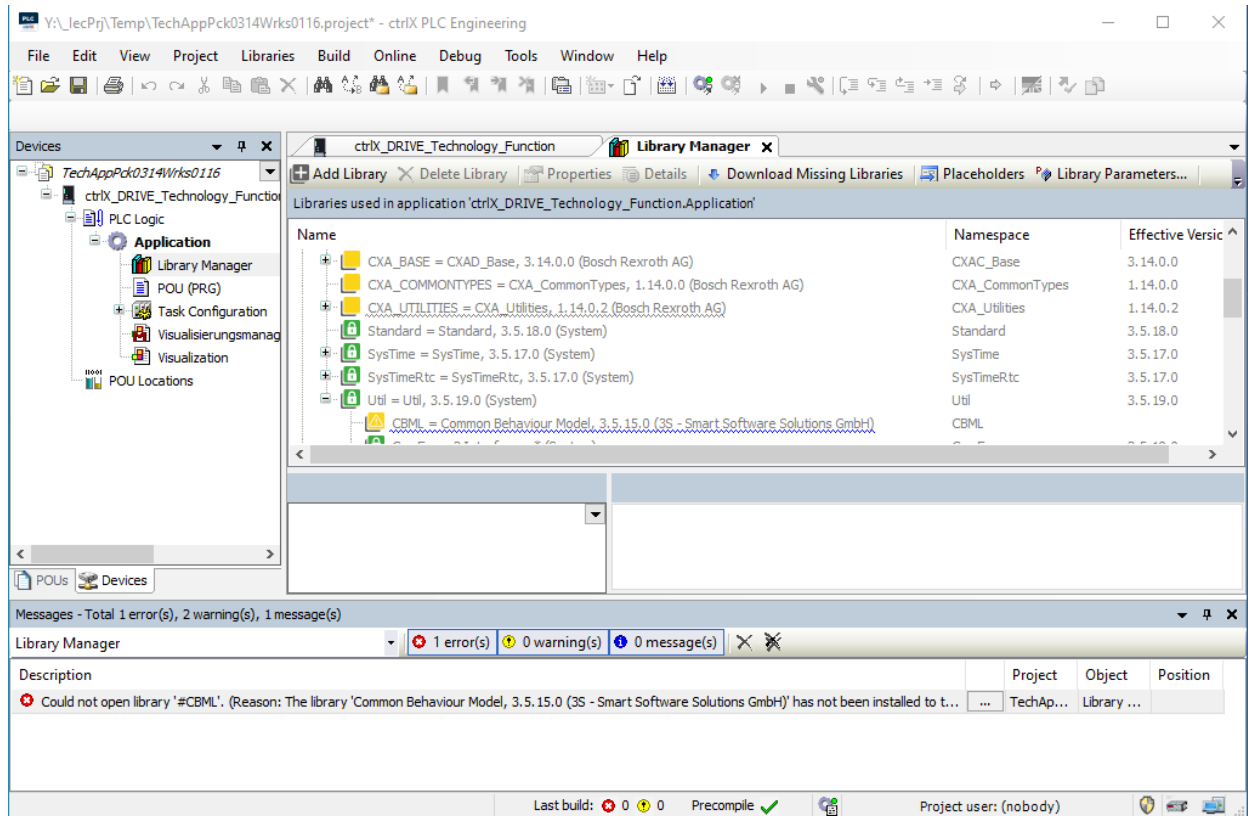
Installing the package with ctrlX PLC Engineering version 1.18.x and above**Requirements**

- ctrlX PLC Engineering ($\geq$ version 1.18.x) must have been installed.
- The package *ctrlX_DRIVE_TF_Vx.x.x.x.package* has to be available on the PC.

1.  Start ctrlX PLC Engineering.
2.  Open the “Tools → Add-on Installer” menu
3.  Allow the "User Account Control" to start the "APInstaller.GUI" program.
 - ➔ The “Add-on Installer” dialog is opened.
4.  Click the “Install file” button
 - ➔ The dialog for selecting the package is opened.
5.  Navigate to the directory that contains the package *ctrlX_DRIVE_TF_Vx.x.x.x.package*, and select the package. Click the “Open” button.
 - ➔ The confirmation dialog is opened. Confirm the installation with the “OK” button.
6.  If the package ctrlX DRIVE Technology Function does not yet have a signature, you have to confirm the missing signature by ticking the check box “I want to continue despite the missing signature(s)”. Click the “Next” button.
 - ➔ The installation of the package *ctrlX_DRIVE_TF_Vx.x.x.x.package* is started, and if necessary you will be requested in a message box to close ctrlX PLC Engineering.
7.  Close ctrlX PLC Engineering.
8.  Confirm the message window with the “OK” button.
 - ➔ The standard set of device description file and libraries defined in the package is installed.
9.  After successful installation, you can restart ctrlX PLC Engineering.

Downloading missing libraries

When an older project is edited with an updated version of ctrlX PLC Engineering or after an update of ctrlX PLC Engineering, required library versions might not have been installed. If required library versions have not been installed, this is signaled as an error during compilation and displayed in the library manager:



In this case, the missing libraries can be subsequently installed in the library manager via the Internet with the “Download Missing Libraries” command. Depending on the configuration of the company network, it might be required to create and use proxy settings [“Tools → Options → Proxy settings” (see also ctrlX PLC Engineering, Application Manual)].

5.4 Enabling the functional package "ctrlX DRIVE Technology Function"

This section describes how to enable the functional package ctrlX DRIVE Technology Function in ctrlX DRIVE Engineering.

Prerequisite

- "Technology Function" "TE1", "TE0" or "TX1" must have been licensed.

NOTE: "Technology Function" "TF1" is not sufficient for programming ctrlX DRIVE Technology Function!

If the device was not ordered with the required license, please contact our service department to have the "Technology Function" "TE1", "TE0" or "TX1" subsequently licensed.

1. ➤ Start ctrlX DRIVE Engineering.
2. ➤ Establish the connection to a device fulfilling the specified requirement.
3. ➤ Call the menu *"Commissioning → Enabling firmware functions"*.
 - ➔ The "Enabling firmware functions" dialog is opened.
4. ➤ Tick the check box of the firmware function ctrlX DRIVE Technology Function ("Technology Function" section).
5. ➤ Restart the device by switching it off and back on.

5.5 Creating a project for ctrlX DRIVE

This section describes the prerequisites and steps required to create a project for ctrlX DRIVE in ctrlX PLC Engineering.

Prerequisites

- ctrlX PLC Engineering must have been installed.
- The package *ctrlX_DRIVE_TF_Vx.x.x.x.package* has to be available.

1. Start ctrlX PLC Engineering.
2. Press the shortcut [ctrl]+[n] or select “File → New Project...” in the menu.
➔ A dialog to create a new project is opened.
3. Select the template “ctrlX DRIVE Technology Function”.
4. To confirm the selection, click “OK”.
➔ A project with "ctrlX DRIVE Technology Function" as the device is created. The program function block "PLC_PRG" with the programming language "Structured Text (ST)" is created.



Alternatively, a new project can be created using the “Standard project” template. For this purpose, select "ctrlX DRIVE Technology Function (Bosch Rexroth AG)" as the device.

5.6 Example programs for ctrlX DRIVE

Example programs for the function blocks are available in ctrlX PLC Engineering in the library manager:

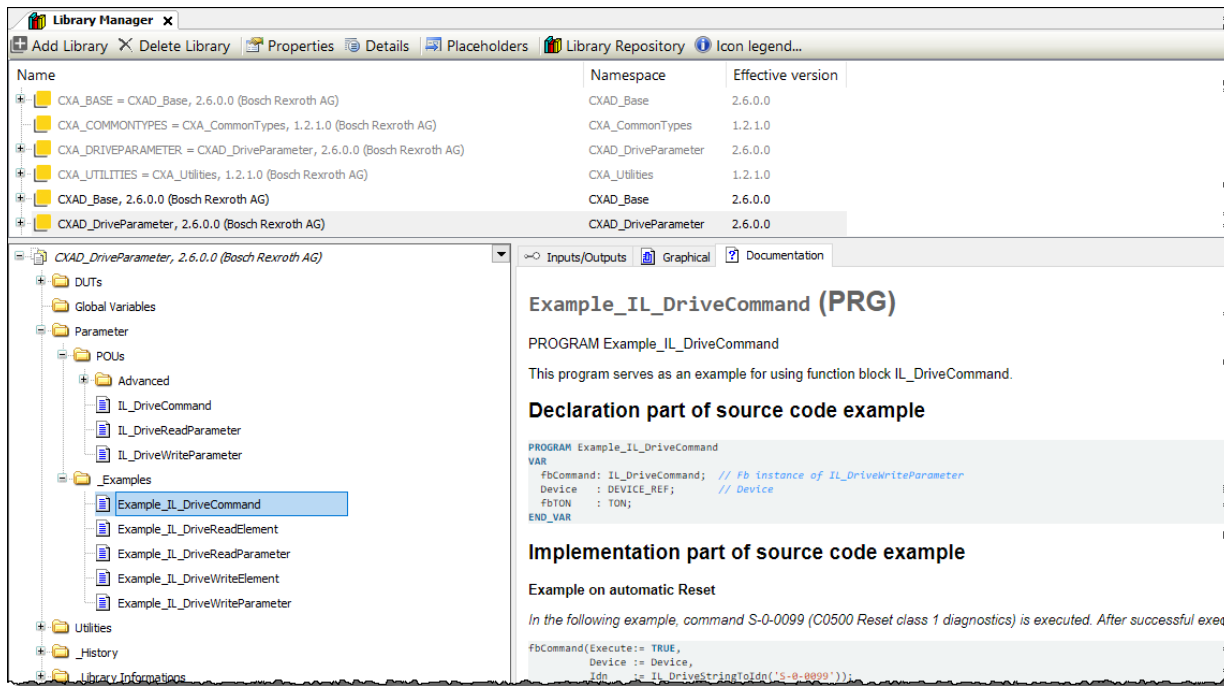


Fig. 1: Example programs for the POUs IL_DriveReadElement, IL_DriveWriteElement, IL_DriveCommand, IL_DriveReadParameter and IL_DriveWriteParameter

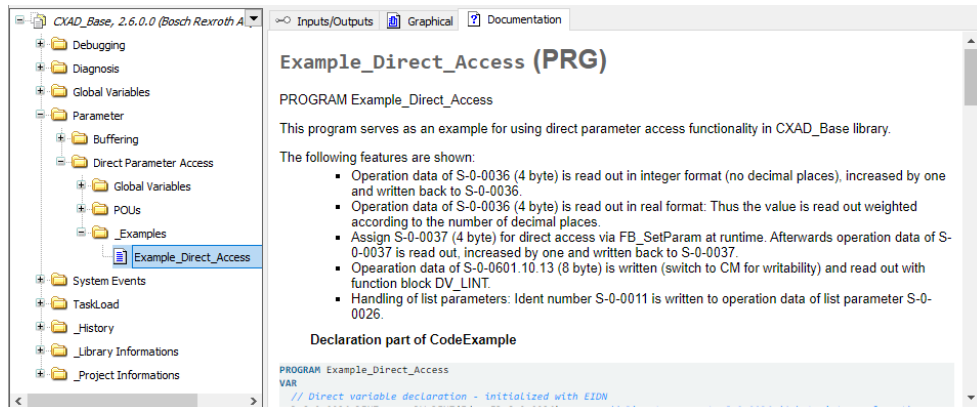


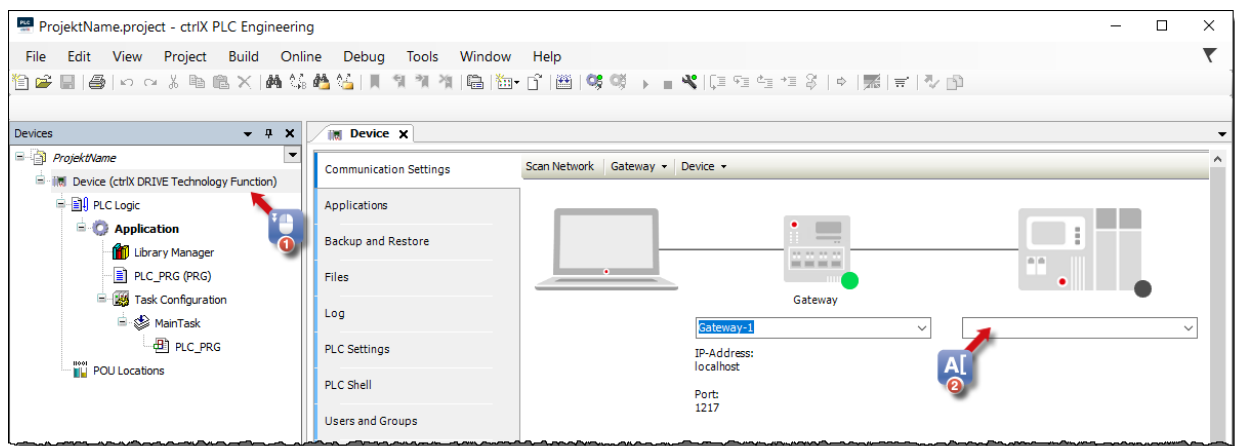
Fig. 2: Example program for acyclic access to axis parameters via direct variables

5.7 Establishing the communication to the ctrlX DRIVE target

This section describes the prerequisites and steps required to establish the communication to the ctrlX DRIVE target in ctrlX PLC Engineering.

Prerequisites: A project for the ctrlX DRIVE target must have been created in ctrlX PLC Engineering.

1. Start ctrlX PLC Engineering.
2. Open the project created for the ctrlX DRIVE target.
3. In the “Devices” view, double-click "Device (ctrlX Drive Technology Function)" (1)
 - ➔ The “Device”, “Communication Settings” window opens.



4. Enter the IP address of the axis (2) and confirm your entry with [↵].
 - ➔ If the connection to the target has been established successfully, the LED at the device symbol turns green in the “Device”, “Communication Settings” window, and the target information is displayed.
5. Download the application code to the ctrlX DRIVE target system. Log in and start the PLC (refer to "[ctrlX PLC Engineering, Application Manual](#)")

6 Task system

6.1 Definitions of terms

- **Task:** A task is a self-contained, elementary control flow within a process. Each task has its individual state with program counter, memory stack, etc.
- **Multitasking** (multi-process operation): Multitasking is the ability to perform multiple tasks simultaneously. The different processes are activated alternately in such short intervals that this creates the impression of simultaneity.
- **Scheduler:** A scheduler is an important element of a multitasking operating system. According to a given strategy, it determines the order in which ready tasks are processed. The term "scheduling" refers to the corresponding strategies and methods with which the processing order of the task is implemented.

6.2 Brief description

Task configuration

At least one task with at least one program call has to be created to ensure a defined PLC behavior.

Supported task types

ctrlX DRIVE Technology Function supports up to three IEC61131 user tasks of the following task types:

- Periodic task ("cyclic" starting from 1 ms)
- Freewheeling task
- Event task (triggering at 1-ms intervals)



ctrlX DRIVE Technology Function supports system events.

Task system and priorities

The internal timing and the sequence of the processing of the individual tasks is managed via the so-called task system:

- Preemptive multitasking is supported. This allows low priority tasks to be interrupted by high priority tasks at any time.
- A maximum of three tasks can be simultaneously "active".
- There are three priority levels [priority 0 to 2 (0=highest priority)].
 - The tasks used have to have different priority so that defined processing is obvious.
 - Tasks of high priority interrupt tasks of low priority also during processing.
 - If a low priority task is started, it may start running with a delay.
- An error message is output when a task interval value is set too high in the task configuration.



Up to three different tasks can be created.

Real-time requirements and runtime measurements

For PLC applications based on tasks with real-time requirements, task run-time measurement using the function block `MX_IECTaskGetLoad` (CXAD_Base library) is recommended. Thereby, it is possible to check whether the tasks in the PLC application provide sufficient time reserves for further development and maintenance of applications, as well as for firmware updates.

6.3 Task properties

Periodic (cyclic) tasks

Periodic (cyclic) tasks have the following features:

- Constant cycle time
- The cyclic task is started exactly once per specified period, i.e., the code is run exactly once in every time interval, provided the previous run was completed (in the case of time overflow and activated watchdog, the PLC is stopped with an "Exception" error status).



Task intervals in seconds are critical since, e.g., when stopping a task, there is a waiting time of up to twice the periodic time.



If a periodic task uses up the allocated computing time and is still active when it should already have restarted, it is not restarted before the next period. This means the system does not attempt to "repeat" the lost cycle.

Freewheeling task

A freewheeling task has the following features:

- A freewheeling task runs constantly.
- There can only be one freewheeling task and it has to be created with the lowest priority.
- A freewheeling task restarts immediately after the end of the task.

Event tasks

Event tasks have the following features:

- Event tasks are started by setting (edge) a global Boolean PLC variable. The start conditions are periodically checked. The reaction times are defined by the minimum possible PLC cycle time.
- An event task is processed exactly once after the event (edge) occurred.

Status tasks

Status tasks have the following features:

- Status tasks are started by setting (level) a global Boolean PLC variable. The start conditions are periodically checked. The reaction times are defined by the minimum possible PLC cycle time.
The PLC variable is typically deleted in the status task.
- A status task is run when the corresponding variable has the value "TRUE" (level-controlled).

PositionLoop Event

To process IEC code in clock rates faster than 1 ms, there is the "PositionLoop Event". When activated, a registered function is called cyclically in a clock of 250 µs, at the end of the XV interrupt.

This cyclic function allows simple PLC functions to be executed in the fastest possible clock. However, there are specific restrictions and rules that have to be observed for application.

Please contact our Application Support to receive advice on how to use the "PositionLoop Event".

Synchronizing a status task with a consumer connection

With information functions for process data communication in the "PositionLoop Event", a status task can be coupled to the reception of cyclic data. For this purpose, the function "MX_INT_fPcomGetConDataCounter()" has to be called in the "PositionLoop Event". The function "MX_INT_fPcomGetConDataCounter()" provides the current data counter of a command value connection. If the data counter is changed, a BOOL variable is set that is used as level trigger of a status task. In the task, the BOOL variable is reset. For further information please contact our Application Support.

6.4 Task monitoring (watchdog)

Generally, monitoring can be set individually for each task. This allows, e.g., the watchdog of a task to be set to a multiple of the periodic time.

Watchdog function

ctrlX DRIVE Technology Function supports the following watchdog function:

- Time (resolution: millisecond)
- Sensitivity:
 - Number of overtimes
 - Duration of overtime (periodic time of task × sensitivity)
- Error reaction upon watchdog:
 - A PLC exception is generated.
The PLC exception is visible as a status in ctrlX PLC Engineering and in P-0-1351, bit 3. An error text is output in P-0-1365.
The error "F6010" is generated on the drive controller; this causes a corresponding error reaction (best possible deceleration) for the axes.
 - The PLC goes to *STOP*.



After the watchdog triggers, the PLC has to be reset with Reset (cf. P-0-1350).

The error F6010 has to be additionally reset using the reset command [C0500 (S-0-0099)].

Configuring the watchdog

For each task, time monitoring (watchdog) can be configured in the task configuration.

The watchdog is active when the option has been activated. The task then is terminated with an error status ("exception") as soon as the configured watchdog time is exceeded; the specified sensitivity is taken into account for the calculation. The following cases are possible:

Multiple successive timeouts; the following applies:

Sensitivity	Exception in cycle...
0, 1	1
2	2
...	...
n	n

One-time timeout:

Exception when the cycle time of the current cycle is longer than "time × sensitivity".

Example

Time="t#10ms", sensitivity="5"

⇒ Exception as soon as the task (one-time) runs longer than 50 ms.



If required, a watchdog can be temporarily deactivated by using the "Cmplec-Task.library" functions. This is useful for cycles that, for example, may take more time due to initializations.

To switch a watchdog off/on, proceed as follows:

1. ➤ Declare a suitable variable for the task handle (RTS_IEC_HANDLE type):

```
hIecTask : RTS_IEC_HANDLE;
```

2. ➤ Use the interface functions to switch off a watchdog (and switch it back on afterwards):

```
hIecTask := IecTaskGetCurrent(0);
IecTaskDisableWatchdog(hIecTask); ... // Code
that is protected against watchdog
IecTaskEnableWatchdog(hIecTask);
```

6.5 Task stack

For PLC program processing, a so-called task stack memory is used per task to store temporary files. This task stack is limited to 4450 bytes in ctrlX DRIVE Technology Function. The stack requirement for programs and function blocks is minimal, since their data are stored in the global data area. Functions and methods, however, do not have any instance and store all data (VAR*) in the stack. It is recommended to avoid big data structures, fields, strings or create function blocks (see also [Chapter 11 Programming information on page 46](#)).

In the case of nested function calls, etc., the stack requirement increases accordingly.

If the available stack memory is exceeded, an error is signaled during the compilation of the PLC application and the project cannot be loaded: "C0297: Stack overflow detected in...(...)". In this case, the stack requirement has to be reduced by the call hierarchy.



In the case of recursive method calls, the memory requirement of the task cannot be calculated. In this case, the following command error is displayed and the check cannot be performed: "C0298: Calculation of stack usage incomplete because of recursive calls:..." Due to the limited memory size, programs with such constructs are not allowed in ctrlX DRIVE Technology Function since incalculable risks would be caused.



As of device version 4.4.0.0, an error is generated for recursive calls to methods if the memory requirement of the task cannot be calculated due to recursive calls of methods, which prevents download. If it can be predicted with certainty how much stack the calls can actually require at maximum, it is possible to specify the maximum stack requirement with the help of a pragma. Thus, the project can be compiled at your own risk: {attribute 'estimated-stack-usage' := '<number of bytes>'}.}

6.6 Reading and setting the task interval time

There are system functions that can be used to read out or modify the task interval time. Functions for writing the task interval time should only be used after consulting the Bosch Rexroth Application Support.

To read out the cycle time of a periodic task, add the "SysTaskGetCurrent()" and "SysTaskGetIntervall()" functions from the "SysTask" library in the library manager.

Beispiel:

```
VAR
    hTask: SysTask.RTS_IEC_HANDLE; // Task handle
    udiInterval: UDINT; // Interval in micro seconds
END_VAR
// Get current task
SysTaskGetCurrent(ADR(hTask));
// Get interval of the current task
SysTaskGetInterval(hTask, udiInterval);
```

NOTICE

Other functions of the "SysTask" library must not be called!

6.7 Runtime behavior

6.7.1 Time slot method

Since the tasks of the integrated PLC (ctrlX DRIVE Technology Function) are processed by the drive processor, they have to share the computing time resources with the rest of the system [position controller, velocity controller, current controller, different background tasks (such as command task)].

The integrated PLC works cyclically in time slices, regardless of the task type used. In every millisecond, the PLC can run for a certain time. In this PLC time slice, the PLC tasks run in accordance with their configuration. At the end of the PLC time slice, running PLC tasks are interrupted and continued in the next millisecond.

In the remaining time (communication time slice), other tasks with functionalities for communication and other drive services are called.

Depending on the hardware equipment, configuration and operating state of drive control, the available size of the PLC time slice varies considerably.

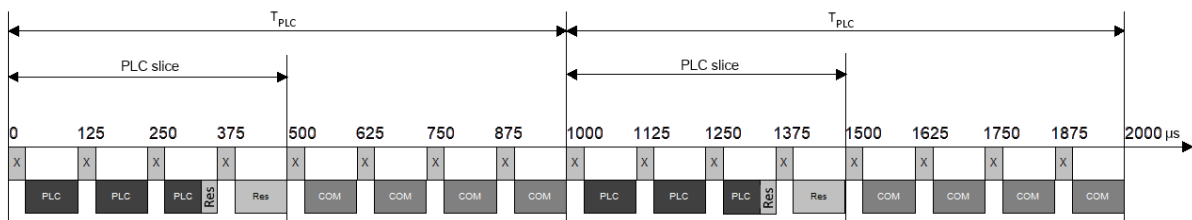
PLC cycle time

The PLC cycle time T_{PLC} of ctrlX DRIVE Technology Function defines the periodic time for cyclic tasks. The T_{PLC} consists of one or multiple PLC time slices.

PLC time slice

The integrated PLC (ctrlX DRIVE Technology Function) works cyclically in time slices, regardless of the task type used. Depending on the system load, a certain computing time is available to the ctrlX DRIVE Technology Function for each PLC time slice.

The PLC time slice (T_{PLC}) is always one millisecond. It also defines the reaction time of the event tasks. Therefore, the minimum task cycle time is also 1 millisecond.



T_{PLC}	Cycle time of an entire PLC time slice: always one millisecond
PLC slice	Part of time slice for PLC tasks: always 500 µs in one millisecond time slice
X	XV interrupt (is measured and deducted)
PLC	Ensured PLC computing time = reference value for 100% display at function block MX_IECTaskGetLoad
Res	Reserve PLC computing time
PLC+Res	Total computing time available for the PLC

The ensured computing time is a defined part of the "PLC slice" time available for the PLC. PLC tasks can also use the entire PLC time slice. If they take more than their ensured computing time in this case, the function block MX_IECTaskGetLoad shows a load > 100%.

Minimum PLC cycle time

The minimum PLC cycle time ($T_{PLC,min}$) consists of one PLC time slice; it also defines the reaction time of the event tasks. The minimum cycle time is 1 ms.

6.7.2 Ensured computing time

Over the product lifetime of ctrlX DRIVE Runtime, the available computing time (PLC time slice) will typically decrease in the case of new firmware releases, because the scope of contained drive functionality expectedly increases.

To nevertheless provide a constant computing time for the PLC over the entire product lifetime, a defined part of the entire available computing time is permanently ensured for the PLC.

This ensured computing time has been planned in such a way that it does not change between the firmware releases of ctrlX DRIVE Runtime during the product life cycle and thus is also guaranteed in future releases.

It is highly recommended not to exceed the ensured computing time so that a currently created PLC application with its tasks can also fulfill its real-time requirements with future firmware releases.

Furthermore, it is recommended not to use up the entire ensured computing time for maintenance and further development of the PLC application.

To monitor the computing time used, there is the advanced task runtime measurement (see [➔ Runtime measurement](#)).



If drive functions are activated/deactivated or changes are made in the configuration (control, cycle times, cyclic data, etc.), the available or ensured computing time for the PLC might change. The ensured computing time applies to firmware releases with an unchanged scope of activated functions and an unchanged configuration.

6.7.3 Runtime measurement

Runtime measurement in ctrlX PLC Engineering

When you have logged in with ctrlX PLC Engineering, the task runtimes are displayed in the task configuration with some statistical information.

However, the values only have a resolution of one millisecond and therefore are not suited for getting the exact runtime of a task with a cycle time of one or a few milliseconds.

Advanced runtime measurement with function block

With the function block `MX_IECTaskGetLoad` from the `CXAD_Base` library, the PLC task runtime can be measured more precisely. This is particularly interesting for tasks with short intervals (in a range of one to a few milliseconds) which obligatorily have to be processed within their periodic time.



With the library manager of ctrlX PLC Engineering, a detailed description of the function block can be found in the `CXAD_Base` library.

The time slot method, together with the ensured PLC computing time, is taken into account for measuring (see "Ensured computing time").

The actual runtime of the task is compared to its maximum ensured runtime (according to the periodic time). The result is the load of each task in percent and the unused remaining time, in relation to the ensured computing time.

In the case of multiple tasks, there are dependencies on high priority tasks. When the computing time of a high priority task is increased, both its own load and the load of the low priority task are increased, since their available computing time is reduced.

High priority interrupts, such as current controller, velocity controller, etc., are not included in the load calculation, so that the display varies depending on the occurrence of these interrupts. This affects the displayed load in such a way that to some extent it changes abruptly.

The task runtimes are not measured automatically, but the measurement has to be activated by using the corresponding function block "MX_IECTaskGetLoad" in the PLC project. This function block provides the values of the task specified at the input. Basically, the function block can be called in any task. It is useful to call it in a low-priority task, unless you want to evaluate every task cycle. The runtime measurements are activated when a project is loaded that contains at least one function block instance of the "MX_IECTaskGetLoad" type. When another project is loaded that does not contain the function block, the runtime measurement is deactivated.



A PLC task may take more than the ensured computing time. In this case, the displayed load at the "Load" output shows more than 100%. For a task with a load of more than 100% it is not guaranteed that it has sufficient processing time over the entire product lifetime.



We highly recommend using "MX_IECTaskGetLoad" for all tasks and keeping the maximum task load "LoadMax" below 100% or accordingly lower in order to allow for maintenance and enhancements of the PLC application without overloading the tasks.

Displaying the maximum load of all tasks

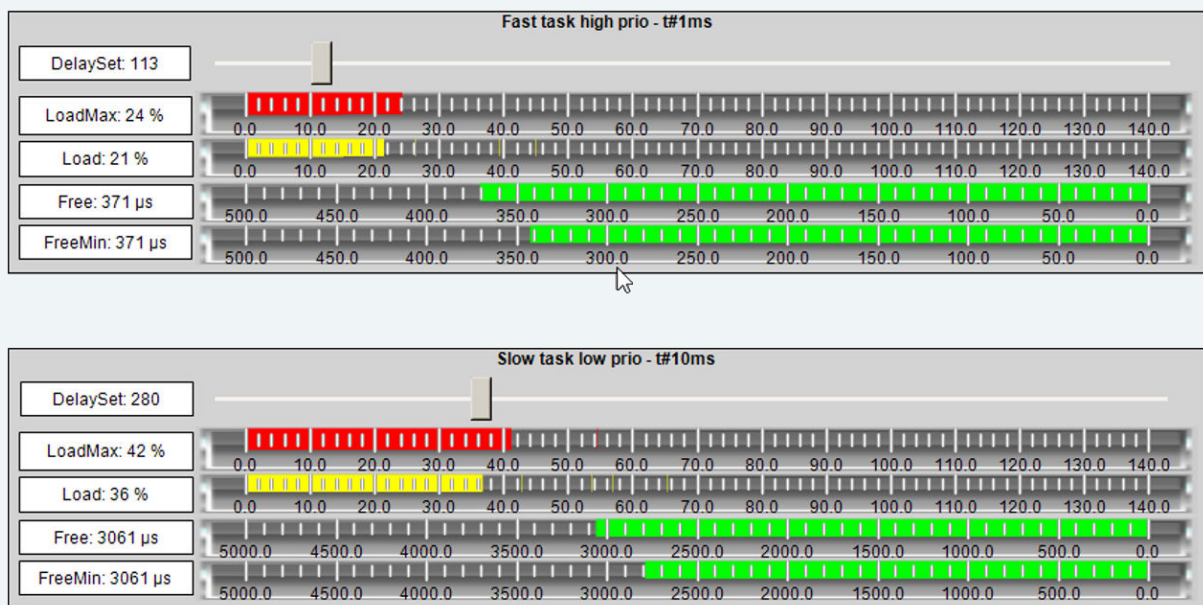
The maximum load of all tasks is displayed in P-0-1366 as a percentage value. As a prerequisite for display, the function block "MX_IECTaskGetLoad" must have been implemented for all tasks.

Writing any value to the parameter resets the maximum displayed in the parameter. This resetting does not have any influence on the displayed maximum values of the instances of function block „MX_IECTaskGetLoad“.

Examples

In the figures below, the runtimes and maximum runtimes of two tasks were measured using the function block "MX_IECTaskGetLoad" and shown as bars.

The load of the fast high priority task has an effect on the slow low priority task.



The following two examples show how the load display is calculated. Independent of the selected control clock of the drive, the XV interrupt with its cycle time of 125μs is taken into account for the calculation. All other interrupts are not included in the measurement.

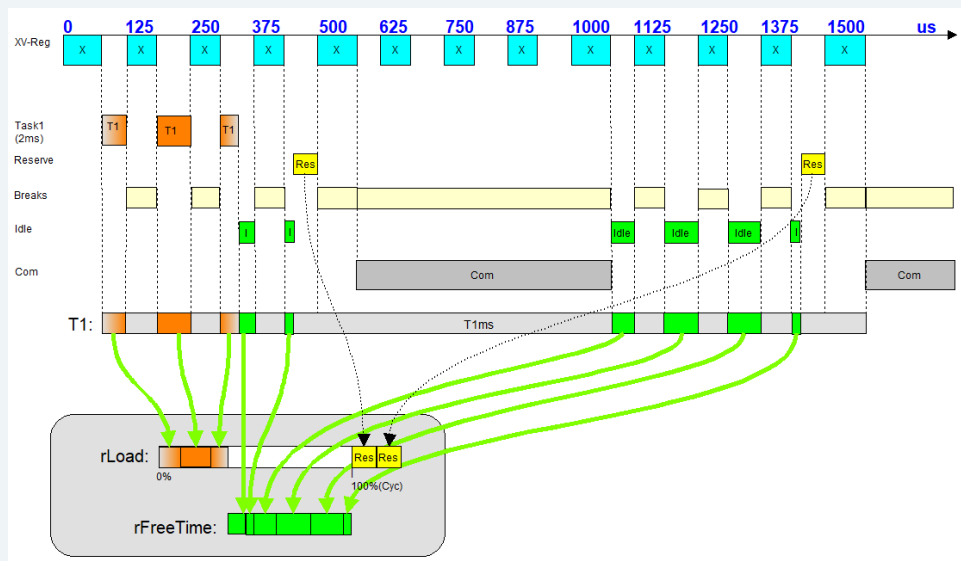
Example1: Task with periodic time $t\#2\text{ms}$ and runtime $< t\#1\text{ ms}$ 

Fig. 3: Display of load per task in percent versus its ensured runtime = possible time minus the reserve time "Res" (without taking other interrupts into account)

XV-Reg	Controller interrupt of drive control
Task1	PLC user task
Reserve	Reserve time (Res) in addition to ensured computing time
Breaks	Deducted time
Idle	Computing time of task that is not required
Com	Communication function and other drive functions
rLoad	Used computing time in relation to the ensured computing time
rFreeTime	Remaining unused computing time of the ensured computing time

The figure above shows a measurement of a task with a period duration of $t\#2\text{ms}$. The task only uses a small part of its ensured computing time. Note that the 100% mark corresponds to the total computing time ensured for this task. The computing time actually used is compared to the 100% time and displayed as percentage value at the "rLoad" output. The remaining unused computing time is displayed as a time value in μs at the "rFreeTime" output.

XV-Reg	Controller interrupt of drive control
Task1	PLC user task
Reserve	Reserve time (Res) in addition to ensured computing time
Breaks	Deducted time
Idle	Computing time of task that is not required
Com	Communication function and other drive functions
rLoad	Used computing time in relation to the ensured computing time
rFreeTime	Remaining unused computing time of the ensured computing time

Access options of the ctrlX DRIVE Technology Function to the ctrlX DRIVE Runtime

7.1 Introduction

- Direct variables for fast parameter access (without any checks)
- Parameter access function blocks to access all elements of any parameters (also list parameters)
- Parameter access function blocks for easy and user-friendly access to single parameter values

7.2 Acyclic data exchange

7.2.1 Access via direct variables

A way to process parameters easily and quickly is accessing them via direct variables.

When using direct variables, read and write access to parameters is possible in the PLC program.

Accessing parameters via direct variables provides the following advantages:

- Simple syntax (simple and clearly structured programming in PLC); no complicated function block call
- Fast parameter access (read and write access)
- Optional parameter access that can be changed during runtime

Direct variable features:

- Direct variables have to be declared in the variable part with the matching data type
- The declaration must not take place in functions or methods but in global data or instances
- In the declaration one or the required parameter has to be assigned
- Additionally, the parameter can be (re-)addressed during runtime using the "FB_SetParam()" method
- When switching between CM/PM/OM, all direct variable instances are updated internally and their writing property etc. is checked again
- No access to remote axes
- Single parameters 16 bit - 64 bit INT formats as well as REAL 32 bit are available
- List parameters 16 bit and 32 bit INT formats are available
 - "MaxElements" is returned as output already during the declaration
 - Read/write access to elements (<=MaxElements)
 - Read access to "MaxElements"
 - Read/write access to "ActElements" (<=MaxElements)
 - Automatic "Complete" with "LastCall" when writing the actual length
- During parameter access, no checks are carried out during runtime
- During read accesses to invalid list elements, 0 is returned or no action is executed when writing
- The processing speed for reading and writing list elements is only marginally slower than for single parameters
- Monitoring: Direct variables can be displayed and modified on the user interface like regular variables. This also applies to the visualization. The access is executed within the communication task context.
- For direct variables in ctrlX DRIVE, only as much memory is required as declarations are created

The following parameter types cannot be used as direct variable:

- Parameters on subsystems
- Commands
- List parameters with ring buffer format
- Access to safety technology parameters
- Specific parameters changing their attribute or other properties during runtime, such as parameters of the oscilloscope or patch function
- Parameters configured for parameter set switching

Supported data types and parameters

There are three different data types of direct variables that can be used for the parameters. Please note that the correct data type has to be used, depending on the parameter attribute. The following table shows the attributes of the available parameters and their matching direct variable type:

Data type of direct variable	Parameter attribute			
	2 byte fixed/variable	4 byte fixed/variable	8 byte fixed/variable	4 byte fixed/variable Float
DV_DINT	OK	OK	Error	Error
DV_LINT	Error	Error	OK	Error
DV_REAL	*) OK, if fixed	*) OK, if fixed	Error	OK

*) Caution: Use DV_REAL to access parameters with DEC_MV and DEC_OV formats. For parameters with scaling, DV_REAL takes the scaling exponent and the scaling factor into account. Any change in the scaling configuration only has an effect after changing to the operating mode.

In the case of display in the REAL data type, the same accuracy as in DV_DINT cannot be achieved in the case of 4-byte parameters. The REAL accuracy is approx. 7 decimal places at most, depending on the scaling.



Parameters whose data type is not contained in the table cannot be addressed with direct variables.

List of parameters supported as direct variables

All parameters that can be used as direct variables are listed in P-0-1260; the access to P-0-1260 can take several hundred milliseconds.



As with the processing of cyclic master communication data, no limit values are checked and no data are stored with direct variables.

Constants

Constants for addressing the list element (only with list parameters) with the "diListElem" input:

- Reading the maximum number of elements: MX_DV_LIST_MAX_ELEMENTS
- Reading/writing the actual number of elements: MX_DV_LIST_ACT_ELEMENTS

Declaration in the variable part

Direct variables can be declared in the variable part with the matching data type, with name and address assignment. Since the declaration takes place in the variable part, no computing power is consumed at runtime for internal parameter search and check.

Parameter selection at runtime

The parameter can also be selected at runtime.

During runtime, the parameter can be assigned using the "FB_SetParam()" method. "FB_SetParam()" has a runtime of approx. 5-10 µs.

- The parameter specified in "Idn" is searched for and access is prepared.
- The error output "uiError" is set during the declaration (PLC start or PLC reset) or after a "FB_SetParam()" call.
"uiError" returns the status; "uiError" can have the following values:
 - "0" in case of success
 - 16#1001 ERROR_ID_NOT_THERE, if the parameter does not exist or is not visible.
 - 16#7008 ERROR_DATA_NOT_CORRECT, if the parameter is not supported as direct variable or the data type does not match.
 - 16#700B ERROR_ADR_NOT_CORRECT, if the instance declaration is contained in a function.

- For lists, the maximum number of elements is displayed in "uiMaxElements" .
- The current writing property is checked and returned in the "bWritable" output.

Code examples

Examples of the variable declaration, the parameter selection at runtime, as well as of the reading and writing parameter access, are contained in the "CXAD_Base" library in the "*Parameter → Direct Parameter Access → _Examples*" directory.

7.2.2 Access via function blocks

By using function blocks, acyclic access (read and write access) to axis parameters is possible by means of custom-made function blocks. The parameter channel allows all S- and P-parameters in the drive, including the "PLC register" parameters, to be accessed via function blocks.



The parameters are directly accessed, i.e., the duration of a read or write process is very short. Exceptions: access to master communication parameters, configuration parameters of safety technology, as well as specific device parameters. In these cases, access typically takes 50-60 ms. This has to be taken into account for use in the task structure (watchdog).

Features

Features in case of access via function blocks:

- Checks during runtime (value range and write protection)
- Instance required
- Action takes place at rising "Execute" edge
- No access to remote axes
- Longer access time than for direct variables



The function blocks for editing the parameters are contained in the "CXAD_DriveParameter" library.

Addressing

Parameters are addressed via constants from the "CXAD_Base" (FP_....) library. The constants used in this library contain the Sercos-compatible address for P-/S-parameters. In the case of function blocks of the CXAD library, the "Idn" input is supplied with these constants.



A detailed description of the individual parameter access function blocks is contained in the function block documentations of the respective libraries CXAD_DriveParameter and CXAD_Base. Open the library manager in ctrlX PLC Engineering, select the function block in the respective library and open the "Documentation" tab. Programming examples are contained in the "_Examples" subdirectory.

7.2.3 PLC parameters for general purpose (global PLC registers)

PLC parameters for general purpose (global PLC registers) are drive parameters that can be used as desired in the PLC program for data management and/or communication between the ctrlX DRIVE Technology Function and external devices, inputs/outputs, sensors, etc. This is not a data channel, but specific (unassigned) drive parameters.



The global PLC registers have no direct influence on the drive, but only act in connection with the ctrlX DRIVE Technology Function.

Features

- The PLC registers are only available with activated ctrlX DRIVE Technology Function.
- Buffered and unbuffered PLC register are available.
- The content of the buffered PLC register is buffered in case control voltage fails, i.e., the PLC register contents are stored in persistent form so that the parameter contents do not get lost in case voltage fails.
- Registers with list structure are available for transmitting large data volumes.
- The writability, name, data format and other properties of the PLC register parameters can be customized using PLC functions.
- The PLC registers are available with the following characteristics:
 - 32 buffered global PLC registers (G0 .. G31) (P-0-1370 to P-0-1385 and P-0-1316 to P-0-1331)
 - 3 buffered global PLC list registers (GL0 .. GL2) with 1024 4-byte values (P-0-1389, P-0-1311 and P-0-1312)
 - 2 unbuffered global PLC text registers (AT0 and AT1) (P-0-1387 and P-0-1388)
 - 72 unbuffered global PLC registers A0..A31) (P-0-1270 bis P-0-1301) and (A32..A71) (P-0-1390 bis P-0-1429) with a data length of 4 bytes
 - 8 unbuffered global PLC registers (A72..A79) (P-0-1440..P-0-1447) with a data length of 2 bytes
 - 2 unbuffered list registers (AL*) with 8192 4-byte values (P-0-1368, P-0-1369)

Use cases for buffered global PLC registers Gxx.

The buffered global registers G0 .. G31 and GL0 to GL2 can be used for the following use cases:

- Configuring PLC functions or function blocks
- Use as non-volatile (permanent) memory for the ctrlX DRIVE Technology Function, because the contents are retained in case voltage fails

Use cases for unbuffered global PLC registers Axx

The unbuffered global registers A0 .. A31 and AL0 to AL1 can be used for the following use cases:

- Online parameterization of PLC functions or function blocks
- Command values and actual values of PLC functions or function blocks
- Communicating with the external control via the master communication interface



When the global PLC register are used as command values (command values from the higher-level control), note that the values will be set to "0" if communication fails (as is typical for inputs).

Applications for global PLC register ATx

The global PLC text registers (AT0 and AT1) are available as freely usable text parameters with a maximum of 255 characters plus closing "0" character.

The global PLC registers (AT0 and AT1) can be used for the following use cases:

- Communicating with higher-level control or HMI
- Defining freely definable status/diagnostic texts

Configuring the PLC register parameters

For the free PLC parameters (global PLC registers), the name, writability, display format, limit values and unit can be configured via the PLC program. The following register configuration functions from the "CXAD_Base" library are used for configuration:

Configuration	Register configuration function	Configurable registers
Name	MX_fSetParamName	All
Writeability	MX_fSetParamWritability	All
Display format	MX_fSetParamFormat	All except for AT0, AT1
Limit values	MX_fSetParamLimits	All except for AT0, AT1
Unit	MX_fSetParamUnit	All except for AT0, AT1



A detailed description of the individual register configuration functions is contained in the function documentations of the "CXAD_Base" library. Open the library manager in ctrlX PLC Engineering, select the function and open the "Documentation" tab. Programming examples are contained in the "_Examples" subdirectory.

The functions should be called when the PLC is started and, if necessary, before the drive is switched from "CM" to "PM" or "OM". A code example for calling these functions to configure the registers at program start and before switching the state machine ("CM", "PM", "OM") is also included in the library "CXAD_Base".

By default, PLC registers are always writable. The writability can be restricted with the function "MX_fSetParamWritability", so that they can only be read externally as actual values, for example, or in only certain drive statuses (e.g. "CM, PM") can be written. With the help of direct variables it is always possible to write the PLC registers from the PLC program.

7.2.4 Access to digital and analog inputs and outputs

Access to digital and analog inputs/outputs of ctrlX DRIVE is only possible via parameters. Access via [Direct variables](#) is recommended.

Depending on the configuration of the control section, digital and analog inputs and outputs are available in different quantities and characteristics. For the available quantity and characteristics, see the Project Planning Manual of the control section used.

▲ WARNING

Since no I/O driver with process image is available in ctrlX DRIVE Technology Function, outputs are not automatically changed upon a stop or an error of the PLC.

Digital inputs and outputs

The digital inputs are mapped as bits in P-0-0307. Access takes place directly by reading the parameter. Alternatively, it is possible to take the long way via PLC register parameters (configured by digital input assignment in P-0-0300 and P-0-0301) to simplify rewiring without modifying the program.

The digital outputs have to be linked to a parameter using P-0-0310 and P-0-0311. For access from the PLC, a PLC register parameter, such as P-0-1270, has to be configured as source in ctrlX DRIVE Engineering.

To set the output, the configured PLC register parameter has to be written in the PLC program, e.g. via direct variable access.

Analog inputs and outputs

The analog inputs and outputs have to be configured in ctrlX DRIVE Engineering.

For access to analog inputs from the PLC, a PLC register parameter, such as P-0-1270, has to be selected as **target**. Optionally, the input value can be read directly without assignment and signal preprocessing. Parameter P-0-0210, for example, can be read for this purpose. It is recommended to read the PLC register parameter or P-0-0210 by direct variable access in the PLC program.

For access to analog outputs from the PLC, a PLC register parameter, such as P-0-1270, has to be selected as **source**. It is recommended to write the PLC register parameter by direct variable access in the PLC program.

Code examples for access to digital and analog inputs and outputs

Code examples for access to digital and analog inputs and outputs are contained in the PLC library "CXAD_Base.library": Refer to "*CXAD_Base.library* → *Parameter* → *IO_Access* → *_Examples* → *Example_IO_Access*".

8 Axis control

8.1 Libraries for axis control

The CXAD_Base and CXAD_AxisControl libraries contain function blocks for axis control for the target ctrlX DRIVE Technology Function.

- CXAD_AxisControl.library
 - MX_VelocityControl()
 - MX_TorqueControl()
 - MX_PositionControl()
- CXAD_Base.library
 - MX_SetControl()
 - MX_HaltBit()
 - MX_Power()
 - MX_GetExternalControl()
 - MX_PresetOpMode()

Distinguishing the ctrlX DRIVE Technology Function to the ctrlX CORE

ctrlX DRIVE Technology Function supports functionalities used to operate the drive as an "intelligent servo axis". It does not support a versatile, stand-alone single-axis "Motion Logic Control"; this purpose requires a ctrlX CORE control.

8.2 Basic principles of axis control with the integrated PLC (ctrlX DRIVE Technology Function)

The operating state of the axis is defined by external and internal control signals (error, power, etc.). The primary external control information for commanding the axis via the master control word (P-0-0116) is as follows:

- "Drive On"
- "Drive enable"
- "Drive Halt"
- Command operation mode input

There are also drive control commands for activating complex, pre-configured commands, such as "drive-controlled homing procedure", "auto-adjustment functions", etc. Depending on the application, the external control signals are preset by an external control via master communication or by the integrated PLC (ctrlX DRIVE Technology Function).

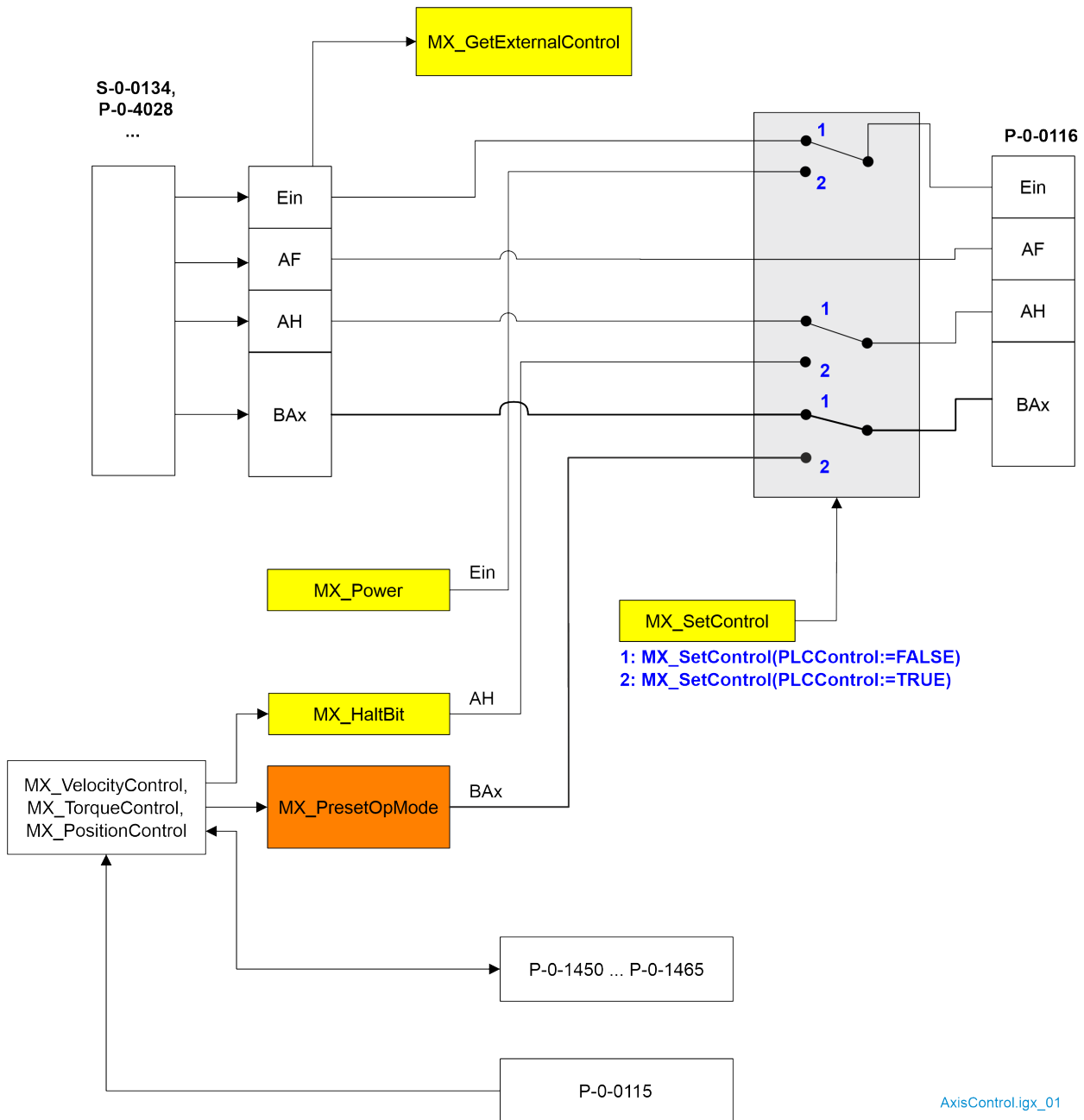
The external control as a standard has control over the axis. The external control signals are specified via the specific control word of each master communication interface. If required, the PLC can temporarily give itself axis control using the function block MX_SetControl(). This may be necessary, e.g., for an intelligent, decentralized error reaction. This means PLC is able to temporarily control the axis via function blocks.

8.3 Commanding the axis

For commanding via the control word of the axis (P-0-0116) with the PLC, temporary axis control has to be activated with `MX_SetControl()`. Afterwards, motion control of the axis with the PLC is possible.



The parameter S-0-0240, bit 0 (DC Bus Power Control) is not automatically written with values, but this has to be done via a master communication or the internal PLC.



AxisControl.igx_01

S-0-0134, P-0-4028,...	Control word of the master communication	AF	Drive enable (P-0-0116, bit 14)
P-0-0115	Status word of the axis	AH	Drive Halt (P-0-0116, bit 13)
P-0-0116	Control word of the axis		Target operation mode (P-0-0116, bit 8, 9, 11, 12)
P-0-1450..P-0-1465	Parameters of the PLC secondary modes	BAx	
Ein	Drive On (P-0-0116, bit 15)		

Controlling the axis

The PLC can get control over the axis with the function block **MX_SetControl()**. With the function blocks **MX_Power()** and **MX_HaltBit()**, the signals "Drive On" or "Drive Halt" (in P-0-0116) can be specified. But the external "Drive enable" signal always takes effect and is used for the external control to decelerate the drive, even in the case of temporary control.

Axis motions

Axis motions are carried out by controlling the operation modes of the drive.

The axis motions can be performed on a higher level of abstraction with easily operated function blocks of the CXAD_AxisControl library, or alternatively by activating the respective operation mode with MX_PresetOpMode() and writing the command value parameters belonging to the operation mode with direct variables. The latter, however, requires more programming effort but involves higher flexibility.

High-level programming level with function blocks of CXAD_AxisControl library

The function blocks MX_PositionControl(), MX_VelocityControl() and MX_TorqueControl() contained in the CXAD_AxisControl library are used to utilize the most important operation modes of the drive. They automatically activate the corresponding internal secondary PLC operation mode using the function block MX_PresetOpMode() and preset the command values of the function block inputs. If the drive still is in the "Drive Halt" state, this state can be canceled using the function block MX_HaltBit().

All function blocks of the CXAD_AxisControl library use the internal secondary PLC operation modes "Torque/force control", "velocity control" or "drive-controlled positioning" as independent instances of the equivalent operation modes with the parameters P-0-1450 to P-0-1465.

While temporary control is active, the parameters P-0-1450 to P-0-1465 cannot be written externally. This ensures that with temporary control, the operation mode settings of the external control are not illegally manipulated.



The CXAD_AxisControl library is available freely readable with all function blocks, so that it is possible to implement one's own function blocks with a differing or advanced functionality according to the example of the existing function blocks.

It is not allowed, however, to modify the existing function blocks! If required, the existing function blocks may be used though as templates for one's own differing function blocks.

Low-level programming level with MX_PresetOpMode() and parameter access

For specific tasks, that are not covered by the ready-made function blocks for controlling the operation modes, it is alternatively possible to directly activate any desired operation mode using the function block MX_PresetOpMode() and to preset all required parameters via direct variables.

9 Diagnostic handling of ctrlX DRIVE Technology Function

9.1 Brief description and overview

Functional features

The ctrlX DRIVE Technology Function is integrated in the diagnostic handling of the drive firmware. Fatal system errors are handled by the drive firmware. In this case, the drive implements a defined handling method. Other system errors caused by PLC programming are handled in the PLC.

9.2 Diagnostics handled in the drive

- Fatal system errors (such as incorrect address access or floating point exception) cause an F8xxx error reaction.
- Drive errors and drive warnings that can be identified at any time via the drive diagnostics (cf. S-0-0390, S-0-0095,...).
- Messages, warnings or errors, generated from a PLC program, that can be triggered using simple function calls:
 - 16 freely definable messages (A2211 .. A2226)
 - 8 freely definable non-fatal warnings without drive reaction (E2211 .. E2218)
 - 8 freely definable fatal warnings with drive reaction (E8211 .. E8218)
 - 8 freely definable non-fatal errors (F2211 .. F2218), still allowing a freely definable variable error reaction (P-0-0117, P-0-0119)
 - 8 freely definable fatal errors with drive reaction (F8211 .. F8218)

Involved functions

The following functions are used in conjunction with diagnostic handling:

- MX_fSetDiag
- MX_fSetDiagText



The functions for diagnostics generation are contained in the "CXAD_Base" library:

1. Open the library manager in ctrlX PLC Engineering,
2. Select the "CXAD_Base" library.
3. Directory tree "CXAD_Base → Diagnosis → POU's"

9.3 Errors that can occur at runtime due to faulty programming

Errors that can be caused by faulty programming are so-called PLC exception errors ("exceptions"). The following errors are handled:

- Task-Watchdog, Omitted Cycle Watchdog
- Division by zero
- TAN(PI/2) 32bit/64bit
- LN(<0) 32bit/64bit
- LOG(<0) 32bit/64bit
- SQRT(<0) 64bit
- EXPT(0,-1) 32bit/64bit
- Pointer accesses with incorrect alignment
- Pointer accesses to third-party data / NULL
- Range violation in case of subrange types
- Field addressing across field limits

The reaction to these errors is defined:

- Drive error F6010 is generated; the axis is decelerated according to parameterization for F6xxx error reaction.
- Display in standard diagnostic system of the drive, cf. S-0-0390, S-0-0095, P-0-0115,...
- Display by error bit in PLC status (P-0-1351). A higher-level control can thus directly react to a ctrlX DRIVE Technology Function error.
- The ctrlX DRIVE Technology Function does not contain any outputs (I/O management). Thus, there is no automatic influence on ctrlX DRIVE outputs in the case of PLC stop.

- The affected task is immediately aborted.
- All PLC variables in the affected task remain frozen to facilitate debugging!
- Other tasks run to their end and are not called anymore (like "PLC STOP"). If a task does not reach its natural end in double its cycle time, it is aborted.
- The error is also displayed on subsequent login.
- The PLC cannot be set to RUN anymore and reports an error upon the first attempt. The RUN state of the PLC then is only possible after a warm start of the PLC.

Diagnostic options

The easiest way of debugging is when it is done with a connected (even subsequently connected) programming system.

Log in with a matching project. The incorrect program line is highlighted and displayed as a breakpoint. Read the call hierarchy in the "Call list" view. The errors are also displayed as "Exceptions" in the "Log" window.

Diagnostics with ctrlX DRIVE Engineering

In the diagnostic trace, the drive error F6010 is displayed and saved with its detailed diagnostics. The error is also displayed in plain text in P-0-1365 (e.g., "EXCEPTION* DivisionByZero appliaction 'Application' task '...' (Pou: x/y)").

9.4 PLC system error

If problems occur upon the start of the PLC, drive error F6010 is triggered with a detailed diagnostics. This points to a hardware defect or an internal error of the drive firmware. In this case, make a backup of all parameters and contact our service department.

Error reaction

- Drive error F6010 is generated; the axis is decelerated according to parameterization for F6xxx error reaction.
- Display in standard diagnostic system of the drive, cf. S-0-0390, S-0-0095, P-0-0115,...
- Display by error bit in PLC status (P-0-1351). A higher-level control can thus directly react to an error of ctrlX DRIVE Technology Function.
- ctrlX DRIVE Technology Function does not contain any I/O management. Thus, there is no automatic influence on ctrlX DRIVE outputs in the case of PLC stop.
- The PLC cannot be set to "RUN" anymore and reports an error upon the first attempt.
- The error can only be cleared by restarting the drive.

ctrlX DRIVE Engineering diagnostics

In the diagnostic trace, the drive error F6010 is displayed and saved with its detailed diagnostics. The error is also displayed in plain text in P-0-1365 (e.g., `SystemError 0x00260001 Src: 0x080E5D41 call support`).

9.5 Errors detected by functions or function blocks

The individual function blocks have outputs which can be used for error handling:

- "Error": an error bit (BOOL)
- "ErrorID": an error number (ENUM type, few values)
- "ErrorIdent": an error code with extended error information

"Error"

The "Error" output is of the BOOL type. The rising edge of "Error" signals that an error has occurred during the processing of the function block.

"ErrorID"

The "ErrorID" output is of the ERROR_CODE type. If an error is present at the function block, the error classification can be taken from the "ErrorID" output.



The ERROR_CODE type is defined in the "CXA_CommonTypes" library.

Refer to the documentation "ctrlX PLC Engineering, Libraries, Reference Book", "CXA_CommonTypes" or select the "CXA_CommonTypes" library in ctrlX PLC Engineering in the library manager. In the directory tree under "DUTs/Diagnosis of POU's/ERROR_CODE".

"ErrorIdent"

The "ErrorIdent" output facilitates detailed error diagnostics. The values of the output are of the ERROR_STRUCT type.

Refer to the documentation "ctrlX PLC Engineering, Libraries, Reference Book", "CXA_CommonTypes".

9.6 Notes on application and programming

Resetting errors in the ctrlX DRIVE Technology Function

Errors can be cleared in the ctrlX DRIVE Technology Function by the "IL_Drive-Command" function block using parameter S-0-0099 (C0500).

Triggering PLC messages, warnings and errors

By calling the "MX_fSetDiag" function, the following PLC messages, warnings and errors can be started and completed in the drive:

- A2211 .. A2226
- F2211 .. F2218
- F8211 .. F8218
- E2211 .. E2218
- E8211 .. E8218

Defining a text for PLC messages, warnings and errors

A freely selectable text can be defined for the transmitted PLC diagnostics by calling the "MX_fSetDiagText" function. Switching the drive off sets the diagnostic texts to their default values again.



A detailed description on how to trigger PLC diagnostics / setting diagnostic texts is contained in the function documentations of the CXAD_Base library. Open the library manager in ctrlX PLC Engineering, select the CXAD_Base library. Directory tree: CXAD_Base/Diagnosis/POUs/MX_fSetDiag or MX_fSetDiagText. For programming examples see under CXAD_Base/Diagnosis/_Examples.

10 External PLC communication

By default, external communication of the Technology Function is carried out using PLC register parameters via the ctrlX DRIVE interfaces.

Furthermore, access via the Ethernet-based interfaces of ctrlX PLC including the symbol configuration is available.

User-programmed Ethernet communication

Using the "CXA_SocketComm" library, it is possible to program Ethernet communication yourself. From this library, only the function blocks for asynchronous communication with UDP can be used. The maximum package size of a UDP datagram is 1472 bytes.

Restriction: In the case of multi-axis devices with device-related IP communication, such as PROFINET or EtherNet/IP, the axes do not have their own IP address. In this case, only the first axis can be accessed.

Access via network variables

Network variables enable communication with PLC variables between different devices with a PLC based on CODESYS and located in a joint network.

Network variables have to be defined in variable lists [so-called "Global Network Variable Lists" (NVL)] both in the transmitter and in the receiver. Their values are transmitted via UDP broadcast or UDP unicast in the network.



Network variables do not have any deterministic real-time behavior. The transmission time from transmitter to receiver and back can take between approx. 40 ms up to approx. 100 ms and more, depending on the load.

To achieve a fast and continuous transmission clock, the transmission intervals of the transmitters should not be set smaller than 20 ms.

Global Network Variable List (NVL)

Network variable lists have to be created for transmitter and receiver at the application node using "Add object". The network variable list in the receiver can also be imported in the project of the receiver from an exported network variable list of the transmitter.

The network variable lists are displayed in the project tree and can be edited in the network variable list editor. The network variable lists in transmitter and receiver have to be compatible.

To make the network settings of the network variable list, proceed as described:

1. ➤ Open the context menu of the network variable list.
2. ➤ Select "Properties...".
3. ➤ Select "Network variables" tab.
4. ➤ Select "Settings...".
5. ➤ Configure broadcast address in the subnet used.



Network variables are always transmitted from the transmitter to the receiver! Due to the fact that each device can handle multiple network variable lists, a device can be a transmitter and also a receiver.

For more information about network variables, refer to the ctrlX PLC Engineering help.

Network variables of multi-axis devices

In the case of a multi-axis device, network variables can be exchanged between the axes. For this purpose, "255.255.255.255" has to be set as the broadcast address for each axis.



When EtherNet/IP or PROFINET is used, communication with network variables to the multi-axis device is not possible.

11 Programming information

11.1 Safe programming

By default, the following monitoring functions are activated by means of the automatically integrated check functions:

Use of pointers ("CheckPointer" function)

Pointer accesses are monitored at runtime. This prevents programs from accessing memory outside of the PLC data ranges. An access attempt to an address outside of the valid limits is detected and generates a PLC exception with F6010. Despite a check for limits, data accidentally may be manipulated due to incorrect access, the drive may show incorrect behavior or crash.

DANGER

Lethal injury or property damage caused by unintended motion or dropping of vertical axes!

⇒ When using pointers, incorrect access can have unpredictable consequences; you should be extremely cautious in this case.

Array accesses ("CheckBounds" function)

Array accesses are monitored for the limits to prevent access to invalid data. In the case of access outside of the limits, a PLC exception is generated with F6010.

DANGER

Lethal injury or property damage caused by unintended motion or dropping of vertical axes!

⇒ If there is no "CheckBounds" function available in the project, indexed access to arrays is not checked. If the index is outside of the allowed range, unwanted data are read or overwritten. This involves the risk of incorrect access having unpredictable consequences.

Subrange types ("CheckRange..." functions)

Data accesses are monitored for the range limits to prevent writing of invalid data. In the case of access outside of the limits, a PLC exception is generated with F6010.

DANGER

Lethal injury or property damage caused by unintended motion or dropping of vertical axes!

⇒ If there is no "CheckRangeSigned" or "CheckRangeUnsigned" function available in the project, access to subrange types is not checked. If the value is outside of the allowed range, this might possibly cause unwanted effects in the PLC program. This involves the risk of incorrect access having unpredictable consequences.

Division by zero ("CheckDiv..." functions)

Division by zero is checked to avoid the calculation of invalid values. In the case of a division by zero, a PLC exception is generated with F6010.



⚠ DANGER

Lethal injury or property damage caused by unintended motion or dropping of vertical axes!

⇒ If the project does not contain any functions which are used to check for division by zero, this might possibly cause unwanted effects in the PLC program. When the divisor is 0, this involves the risk of incorrect results having unpredictable consequences.

The 'no_check' pragma has the effect that no check function (POUs for implicit checks) is to be called for the POU. Since the check functions have an impact on the program processing velocity, it is recommended to apply the attribute to function blocks that have already been sufficiently checked and that have already been called often.

See also the ctrlX PLC Engineering Application Manual, ➔ [Attribute 'no_check'](#)

Access to variables via programming system or interface is only byte-consistent

When a 2-byte or 4-byte variable is read, the reading process might possibly be interrupted by a change in the variable in a PLC task. When multiple contained bytes are simultaneously changed, the wrong value might possibly be temporarily transmitted (e.g., one byte old, one byte new).

Stack check

Stack requirements are checked during compilation. This prevents a program with excessive stack requirements from being compiled. If the stack requirement could not be checked by the compiler (command error C0298), this involves the risk of data violation. Due to the risk of exceeding the stack size, a recursive method call is not allowed.

⚠ DANGER

Lethal injury or property damage caused by unintended motion or dropping of vertical axes!

⇒ Using functions with large data volumes can have unpredictable consequences. The stack requirements for function blocks are minimal, since their data are stored in the PLC data. Functions, however, have no instance and store all data in the stack. Avoid large data structures and fields or create function blocks.

11.2 Working with retain variables

Brief description

Remanent variables, so-called retain variables, can retain their values beyond the program runtime. This means that retain variables retain their values after the device was switched off and back on. Upon restart of the PLC project, the saved values are used for further processing.

One application example for retain variables is a piece counter in a production line which is expected to continue to count after a power failure. All variables – except for the retain variables – are reinitialized after the power failure and after the device was switched back on, either with their initialized values or with the standard initializations.

Persistent variables (variables that also keep their values during download) are not supported. Using the recipe manager is recommended for this purpose.

Storage location

All retain variables described here are stored in P-0-1359 and handled like other retain parameters of ctrlX DRIVE.

Properties and information

P-0-1359 can be written when the device is in the configuration mode (CM) or in the parameterization mode (PM), and the PLC is not in the *RUN* state.

In the PLC project, retain variables have to be identified by the keyword `RETAIN` in their declaration, in a function block or in a global variable list. If at least one retain variable has been declared in a function block, the complete instance of the function block is in the retain memory range.

The user data range for retain variables comprises 248 byte.

Retain variables are reinitialized in the case of “reset cold”, “reset origin” or if the program is downloaded again. In the case of “reset warm”, they retain their values.

To simplify the overview, it is recommended to define retain variables centrally in a global variable list. The retain variables can also be combined in a structure in a global variable list.

⚠ CAUTION

If the retain data (P-0-1359) were completely deleted or do not match the application, no error is displayed, but the retain data are initialized without any further message. Therefore, the application should check the retain data at the start to see whether an initialization with default values has taken place. This can be checked, for example, with a retain variable by assigning a different value than the default value to it when starting. If it previously complied with the default value, all retain data have been initialized and the application can react accordingly.

Variables:

```
VAR_GLOBAL RETAIN
  bFlagRetain : BOOL; // Flag for init of retain data
  iRem1 : INT; // User defined retain variable
END_VAR
```

```
VAR_GLOBAL
  bAllRetainDataInitialized :BOOL;
END_VAR
```

Start:

```
IF NOT bFlagRetain THEN
  bFlagRetain := TRUE;
  bAllRetainDataInitialized := TRUE;
END_IF
```

In program:

```
IF bAllRetainDataInitialized THEN
  // ToDo, if retain data are initialized
  // e.g. manual mode, calibration of machine, etc.
END_IF
```

Modifying the structure of retain variables

If the structure of retain variables is modified, an initialization takes place upon restart. The following modifications in the retain structure cause an initialization:

- Adding or removing a variable
- Re-sorting variables
- Renaming variables

Updating the project with ctrlX PLC Engineering

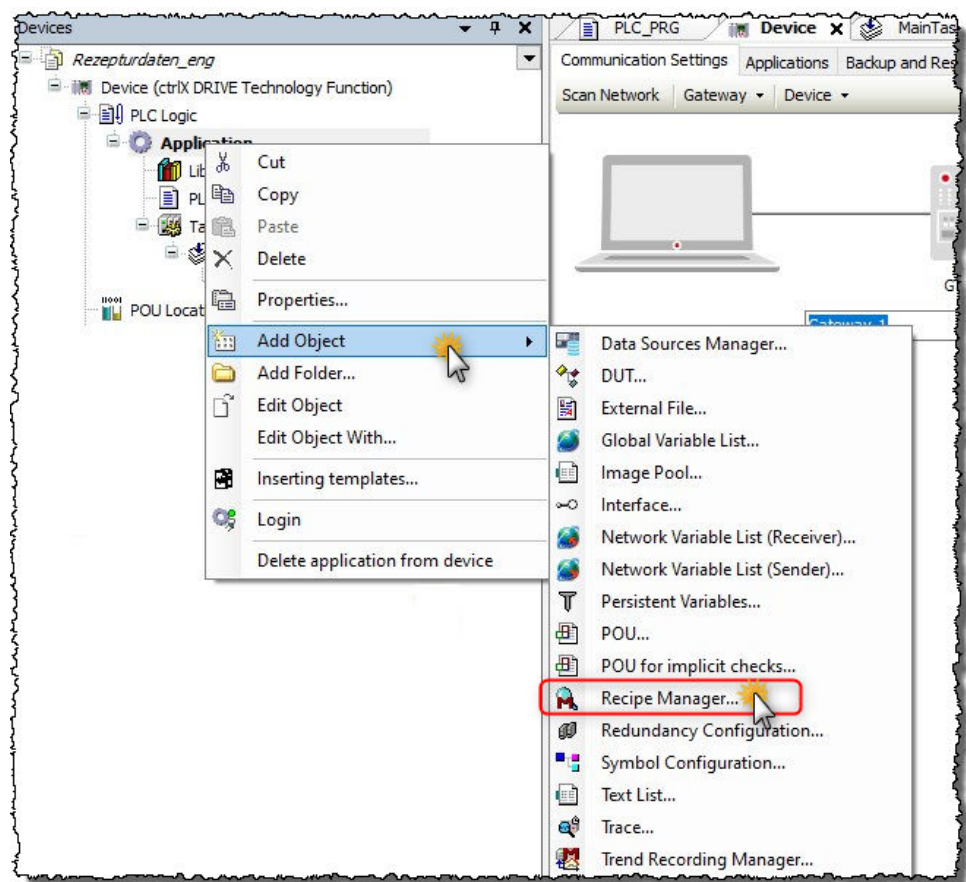
When a project is downloaded to the device, the retain variables are initialized. If the retain data are to be maintained after the download, this can be achieved by using the recipe manager.

Saving the retain data by using the recipe manager

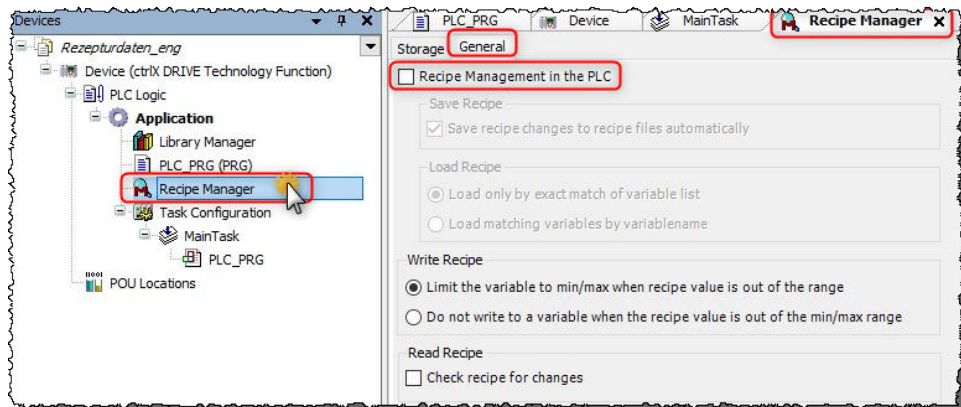
If contents of variables are to be available even after “reset origin” or “reset cold”, this can be achieved by using the recipe manager. For this purpose, a recipe is created with the variables of the retain data. The retain data can then be saved on the PC and reloaded to the device after the download.

Steps to follow for saving the retain data by using the recipe manager

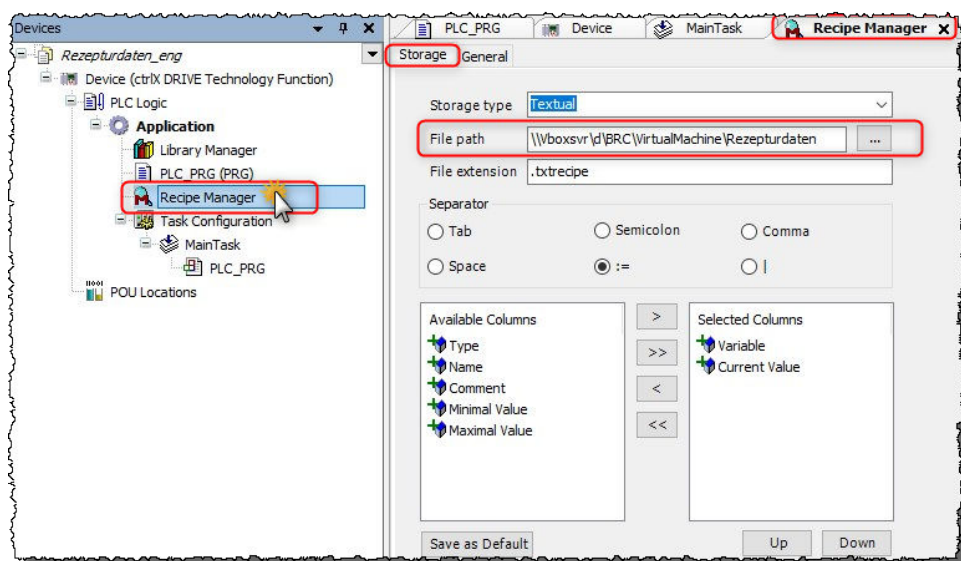
1. In the application, add a “Recipe manager” object.



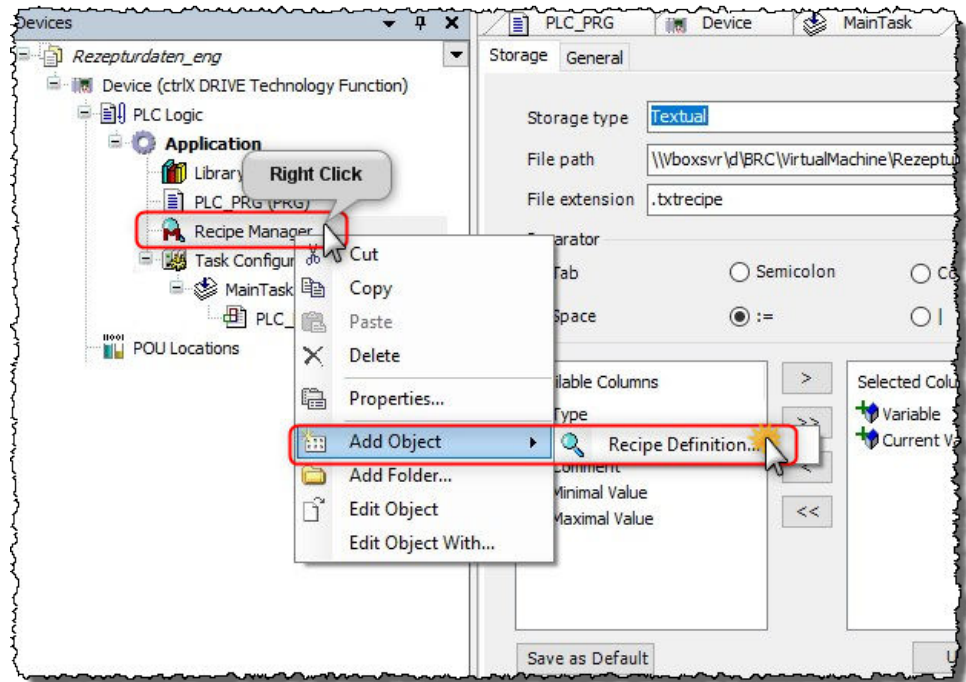
2. Do not select the “Recipe management in the PLC” option!



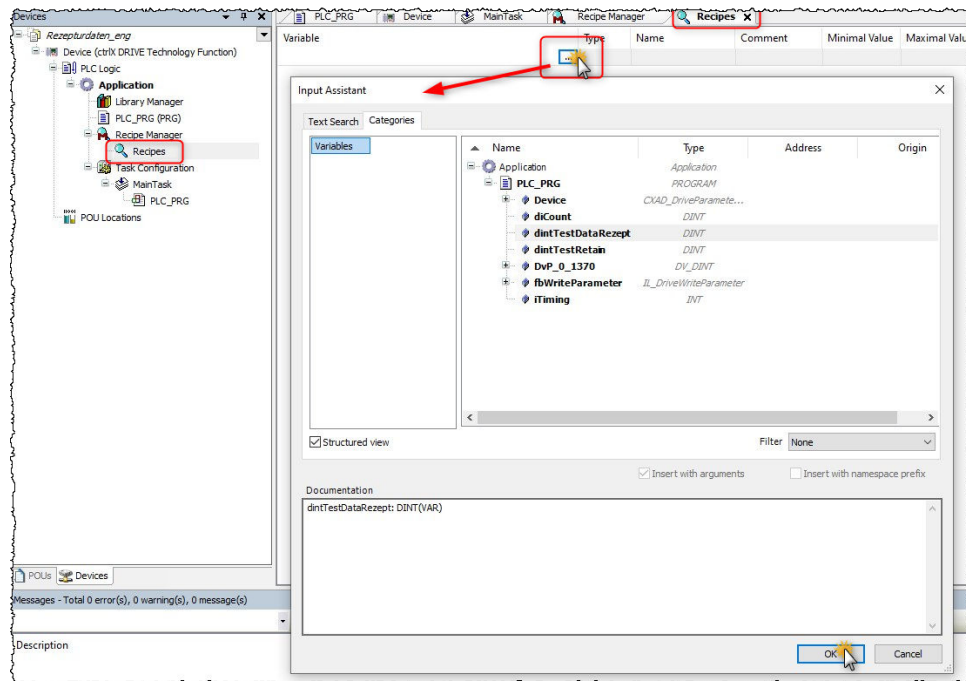
3. Select the storage location of the recipe data (variable values) on the PC.



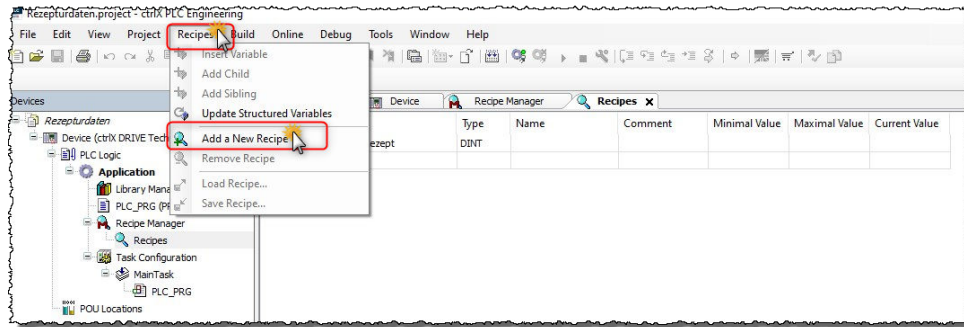
4. In the recipe manager, add a “Recipe definition” object.



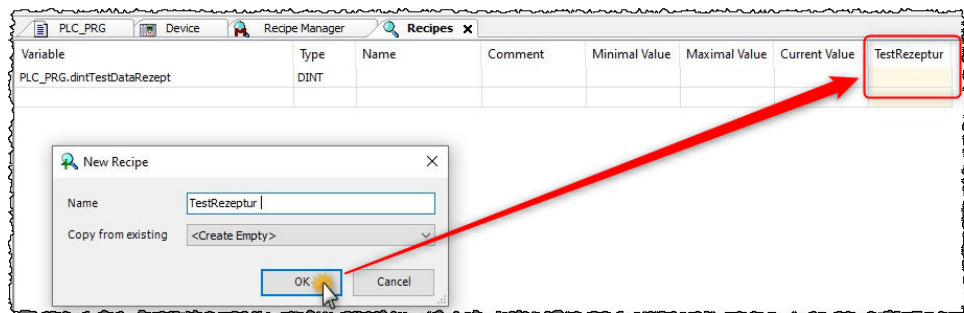
5. Add those variables to the recipe definition the contents of which are to be saved on the PC for restoring them later on.



6. Add a “Recipe” for the variables to be saved.
6.1

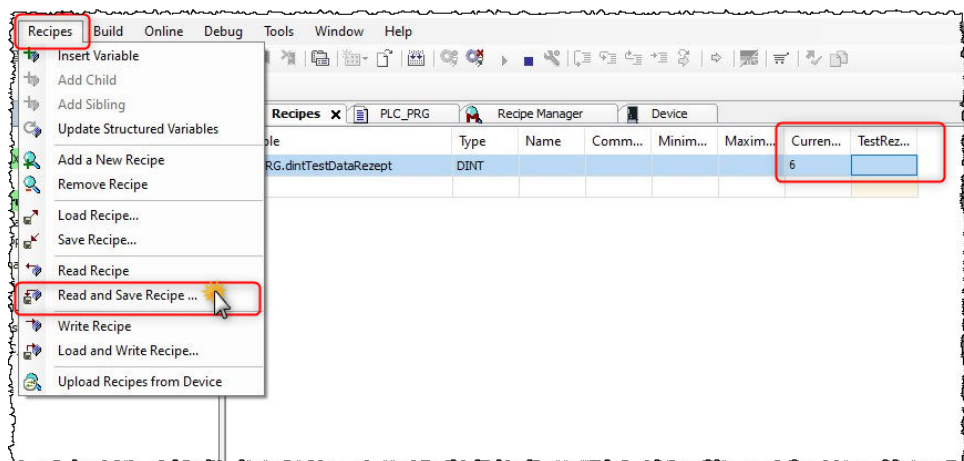


6.2

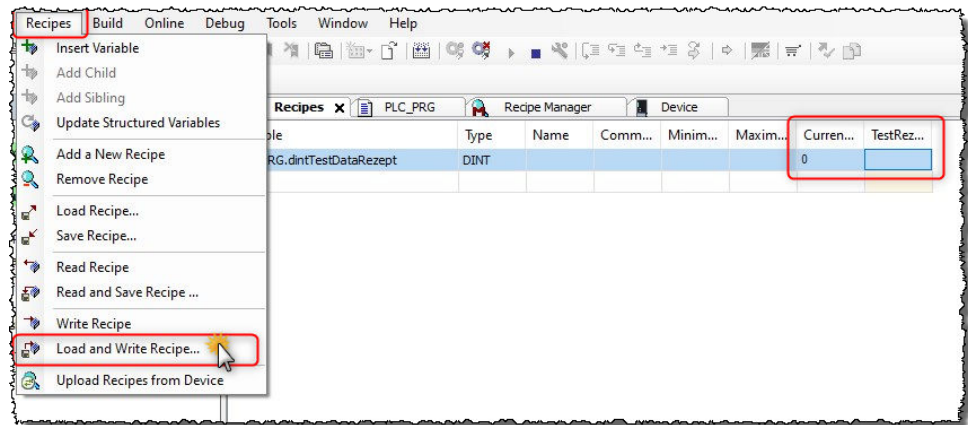


If ctrlX PLC Engineering has been connected online with the device, the variable values can be saved by means of the recipe via “Read and save recipe...”. For example, before replacing a device for servicing.

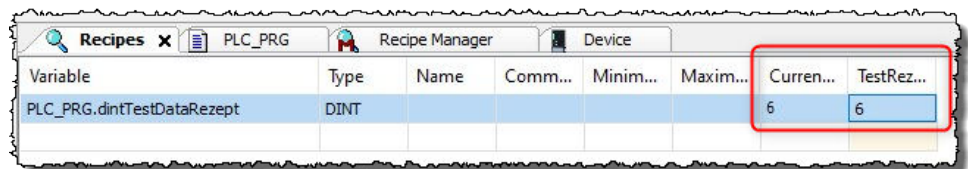
Via “Read and save recipe...”, the variable value is taken from the device to the recipe of ctrlX PLC Engineering and then saved as a file on the PC.



If the variable values saved before are to be restored later on (e.g., after replacing a device for servicing) from the file saved on the PC, this is done by “Load and write recipe...”:



Afterwards, the variables in the device have the value from the file:



Updating the project without ctrlX PLC Engineering - maintaining the retain data

Without ctrlX PLC Engineering the project can be updated by the parameters P-0-1352 ... P-0-1358. After having loaded the parameters and afterwards restarted the device, the new project becomes active. If the retain structure has not changed, the retain data are maintained on the device. If the retain structure has changed, the retain data are initialized. If the retain data are to be retained, this has to be configured. For example, the retain data can be saved in a remanent register parameter before the update, and after the program update they can be taken to the new structure from the register parameter.

Transmitting retain data to a device

If the retain data are to be transmitted to a device, this can be done with P-0-1359, if the retain structure matches. To load the parameter P-0-1359, the device has to be in the *CM* or *PM* status and the PLC in *STOP*. Thereafter, the device has to be restarted.

It is also possible to load the PLC program, together with the retain data (P-0-1352 ... P-0-1359), to a device and afterwards restart the device.

Updating a project of the same retain structure without PLC programming system or serial commissioning

If changes are made in the program only, without modifying the retain data structure, proceed as follows:

Program update with initialization of the retain data (serial commissioning / update without data):

- Switch the device to PM (in the case of update)
- Stop the PLC (STOP) (in the case of update)
- Delete P-0-1359 or entire PLC project (P-0-1350.0.6) (in the case of update)

- Load parameters P-0-1352 to **P-0-1358** to the device („Load parameters“ – without retain parameters)
- Restart the device
- Set PLC to *RUN* again

Program update without influencing existing retain data (update):

- Switch the device to PM
- Stop the PLC (STOP)
- Load parameters P-0-1352 to **P-0-1358** to the device („Load parameters“)
- Restart the device
- Set PLC to *RUN* again

Program update with prepared retain data (serial commissioning / update with data)

- Switch the device to PM (in the case of update)
- Stop the PLC (STOP) (in the case of update)
- Load parameters P-0-1352 to **P-0-1359** to the device („Load parameters“ – with retain parameters)
- Restart the device
- Set PLC to *RUN* again

Parameter

PLC retain data are stored in P-0-1359.

The PLC retain data can be read or written in one step using P-0-1359. The maximum list length (MaxLen) of P-0-1359 is 63 elements of 4 bytes each).

Diagnostics

- If the retain data memory is faulty or after a system error, the error F6010 occurs, when switching on, with detailed diagnostics 0x01260010 and the message in P-0-1365 TF-ERROR* Load of boot project was denied – PLC retain data invalid (P-0-1359).

If the error TF-ERROR* Load of boot project was denied – PLC retain data invalid (P-0-1359) occurs, the Bosch Rexroth support should be informed.

Bug fixing options:

- PLC reset cold and start, then reset with S-0-0099 or restart
– or –
- Reload PLC project, then reset with S-0-0099 or restart
– or –
- Set P-0-1359 valid by writing and restart
CAUTION: By setting P-0-1359 valid by writing and restart, the retain structure is checked, but the contents of the variables are not checked.
- If non-matching PLC retain data or no PLC retain data are found when the drive is switched on, all retain variables get their default value without an error message. The application can recognize this and react accordingly.
- When the available quantity of retain data is exceeded, a corresponding error message occurs in ctrlX PLC Engineering when the code is generated: C0103: Outside of retain memory.

11.3 Libraries for ctrlX DRIVE Technology Function

11.3.1 General properties of the libraries



This section provides an overview of the libraries for ctrlX DRIVE Technology Function.

The function blocks contained in the libraries for ctrlX DRIVE Technology Function are based on the PLCopen specification and all of them, in principle, act in the same way.

Other function blocks that are required but not specified in PLCopen have the same syntax, semantics and the following basic properties:

- The function blocks normally are called cyclically and provide information at their outputs, such as "Done", "Error", "CommandAborted".
- The processes are normally started with a rising edge at the "Execute" or "Enable" input.
- After completion or abortion, the corresponding output ("Done", "Error", "CommandAborted") is set. This output stays TRUE until "Execute" is reset. If "Execute" is already FALSE, the respective output remains in its status for one cycle (call) and then falls back to FALSE.

Other features of the libraries or user interface:

- Instance behavior described in standard IEC 61131
- Error handling ["Error", "ErrorID" and "ErrorIdent" outputs]
- Unassigned inputs are possible
- Cascading of function blocks

11.3.2 Supported libraries

When a project is created with a ctrlX DRIVE target, some libraries are automatically loaded. If required, the user can include more libraries in the project.

NOTICE

Property damage by including unsupported libraries.

Only the libraries listed below are allowed to be included.

In the case of libraries with a placeholder name, use the placeholder.

Library	Placeholder	Description	Automatically loaded?
CXAD_Base	CXA_BASE	Basic functions for ctrlX DRIVE Technology Function target	Yes
CXAD_DriveParameter	CXA_DRIVEPARAMETER	Access (read/write) to parameters of the ctrlX DRIVE Technology Function target	
CXAD_CheckRtv	CXA_CHECKRTV	Monitoring functions for implicit checks	
CXAD_AxisControl	CXA_AXISCONTROL	Functions for axis control	
CXA_CommonTypes	CXA_COMMONTYPES	System-wide data types	Yes
CXA_Uilities	CXA_UTILITIES	Auxiliary functions	
CXA_SocketComm	CXA_SOCKETCOMM	Function blocks for data transfer with TCP and UDP	No
CXAD_DEBUG	CXA_DEBUG	Debugging (for internal use only)	No, to be included by internal development staff only

11.3.3 Documentation on the libraries

The source code of the libraries mentioned under ➔ ["Supported libraries"](#) always contains the library documentation. In the case of compiled libraries (*.compiled-library), the library documentation in addition is immediately displayed in the "Documentation" tab of the library manager. (The library documentation is only available in English.) Libraries with the prefix "CXA_" also

have an online documentation with additional information. If the "Documentation" tab for a library does not display the library documentation, like for example in the case of the open library "CXAD_AxisControl", you can generate the documentation of libraries on your own in ctrlX PLC Engineering, from the source code of the CODESYS library.

To generate documentation from the source code of a CODESYS library with ctrlX PLC Engineering, the "CODESYS Library Documentation Support" package of CODESYS must have been installed in ctrlX PLC Engineering. The package contains the "CODESYS LibDoc Scripting Collection" with which the (JSON-exported) source code of a CODESYS library is read, converted and included in the "Documentation" tab of the library documentation.

To install the "CODESYS Library Documentation Support" package, proceed as described:

Prerequisites

- ctrlX PLC Engineering must have been installed.
- The package *CODESYS Library Documentation Support x.x.x.x.package* has to be available. With ctrlX WORKS V1.12 and above, the package can be found in the installation subdirectory "Additional_packages".

1. Start ctrlX PLC Engineering.
2. Open the "Tools → Package Manager..." menu
 - ➔ The "Package Manager" dialog is opened.
3. Click the "Install..." button
 - ➔ The dialog for selecting the package is opened.
4. Navigate to the directory that contains the package *CODESYS Library Documentation Support x.x.x.x.package*. With ctrlX WORKS V1.12 and above, the package can be found in the installation subdirectory "Additional_packages". Select the package and click the "Open..." button.
 - ➔ The "Check Package Signatures" dialog is opened.
5. After having checked the package signature, start the setup with "OK".
6. Read and confirm the license agreement. "Next" opens the "Choose Setup Type" dialog. Select "Typical Setup".
7. Select the ctrlX PLC target versions for which the setup is to be executed.
8. "Next" opens the "Setup - Restart" dialog. Click the "Finish" button, close the Package Manager and then all instances of ctrlX PLC Engineering. Thereafter, the setup of the "CODESYS Library Documentation Support" package is started. Follow the instructions of the setup wizard.
9. Start ctrlX PLC Engineering.

If now you only see a button instead of the library documentation in the library manager in the "Documentation" tab, then click the button and the source code of the CODESYS library is automatically read, converted and the library documentation is displayed in the "Documentation" tab.

11.4 Start behavior and boot project

11.4.1 General information

The PLC is started automatically upon the drive start and subsequently runs independently of the axis communication phases and the operating state of the axis.

11.4.2 Brief description

The drive-integrated PLC (ctrlX DRIVE Technology Function) is activated upon the drive start. It first initializes itself and checks whether a correct boot project is available. If a correct boot project is available, it is loaded. If the boot project is incomplete or corrupt, error F6010 is displayed. The cause of the error can be narrowed down via the detailed diagnostics of F6010. A more detailed diagnostics of the cause of the error is displayed in P-0-1365.

Starting procedure

When the drive is started

- the PLC operating system is initialized
- a boot project is loaded (if available)
- the boot project is started, if required

11.4.3 PLC behavior after a processor exception

A processor exception that is not intercepted can occur due to an incorrect statement. The error cause can be in the drive firmware or the PLC application. In this case, the drive is immediately decelerated and has to be switched off and on.

A processor exception can also be caused by an incorrect code of a PLC application. In particular, this can be the case if check functions for implicit checks have been deactivated in the PLC application (zero pointer checks, etc.).

Restart process after a processor exception:

Upon restart, the error F8100 "Error when initializing the parameter handling" is triggered by the drive firmware. By stopping the drive firmware, data that are to be saved persistently could not be saved before switching off.

Loading of the boot project is prevented. Additionally, the error F6010 is generated with a detailed diagnostics. This diagnostics shows that PLC retain data are invalid. Parameter P-0-1359 indicates this with the data status "invalid".

The error F8100 cannot be cleared since the drive is not ready for operation without the persistent parameter data.

To reset the error F8100, another restart of the drive is required. Following this restart, the boot project is not reloaded. Thus, error F6010 is triggered again.

The error F6010 can now be cleared by one of the following actions:

- By "reset cold" and subsequent restart
 - or –
- By writing the retain data in P-0-1359 (set status to valid) and subsequent restart
 - or –
- By loading a PLC application in ctrlX PLC Engineering

11.5 Guidelines for programming with ctrlX PLC Engineering

11.5.1 Overview

This chapter is intended to provide the basics for structuring and programming PLC projects with ctrlX PLC Engineering for the ctrlX DRIVE Technology Function target.

The description includes suggestions and recommendations for PLC programming in the form of basic guidelines. They are intended to allow software structures to be used consistently.

Basic knowledge of the PLC programming system ctrlX PLC Engineering is required.



Comply with the guidelines fully for optimal results. If this is not possible, attempt to follow the guidelines as closely as possible.

11.5.2 Compatibility of ctrlX DRIVE Technology Function with the drive firmware

When creating a ctrlX DRIVE Technology Function project, always use the device version compatible with the drive firmware. Install the package compatible with the firmware in ctrlX PLC Engineering.

The version number of the package consists of four digits. The first two digits show the compatibility with the drive firmware. The following two digits are increased in the case of compatible enhancements/bug-fixes. For a firmware, always use the device or package version with the first two digits $\leq$ the drive firmware version and the highest available third and fourth digit. If required (new functions or bug-fixes), the device version can be changed during the processing of a ctrlX DRIVE Technology Function project.



Installing a package version: see [↗ ctrlX DRIVE package installation](#)

Changing the device version: To change the device version, open the context menu "Update device..." on the device branch. If required, activate the option "Show all versions" to set the version with "Update device".

Example:

Package version: 3.2.0.0

Firmware version: FWA-XD1-AXS-V-0302N-NN-01

The following rules apply regarding the drive firmware compatibility:

- Major versions (first digit of version number) can be compatible (e.g., package version 3.2.0.0 could be compatible with firmware version AXS-V-0406). However, it is not guaranteed that packages and firmware with a different major version are always compatible. The compatibility of package and firmware versions is documented in a reference list in the ctrlX DRIVE Technology Function Release Notes. The Release Notes are provided together with the packages in the "Collaboration Room" of ctrlX AUTOMATION.
- In the case of a change in the second or third digit of the version number, an upward-compatible (or incompatible in the prototype phase) change can occur. Usually, the change consists in additional functions within a version.
- Changes in the fourth digit of the version number are compatible. The firmware interface is not extended. Use the latest available package version.

Message during download

If the package is not compatible with the firmware, an error message occurs during the download regarding "Unresolved references" or "Signature conflicts".

The error message contains a list indicating which functions / function blocks from which library could not be linked.

11.5.3 Structuring PLC projects

Multi-program technology

A main program has to be assigned to each task (see also [↗ Task system](#)). If required, this program can contain further blocks (programs and function blocks) (multi-program technology). This provides a high degree of flexibility as regards the calling and controlling of further function blocks. In addition, the order within the respective main program is therefore unequivocal.



It is possible to assign multiple programs to any task. The processing of the programs corresponds to the order assigned. However, further control of these programs is then only possible within the programs themselves, for which reason this variant is not recommended.

By naming the function blocks, the order of display in the "Devices" or "POUs" view can be influenced. It is advantageous to put the main programs at the beginning, e.g., `_prMain_<machine number>`.

Within the function block view, the arrangement of the function blocks can be structured using further sub-folders. It is therefore possible to make a functional arrangement in groups, thus giving the entire project a clearer structure.



Details in the programs/function blocks can be seen in the "POUs" or "Devices" views via double-click or `[Alt]+[Enter]`. The calling point alone determines the location in chronological processing. Where applicable, the function blocks should be named so the order can also be seen in the object tree.

PLC objects

The objects can be classified as follows:

- Device-specific objects and application-specific objects such as device configuration or PLC objects, e.g. the task configuration, are managed in the "Devices" view via the "Device tree".
- Project-wide available objects, e.g. PLC function blocks, are managed in the "POUs" view.

11.5.4 Code distribution to "POU Locations"

ctrlX DRIVE provides two memory segments with different performances for the PLC program code:

- internal fast RAM with limited size (PLC code is processed fast)
- external slower SDRAM (PLC code is processed at regular speed)

The program code can be loaded to these memory segments. If required, the distribution can be controlled, time-critical program parts can thus be edited in the fast internal memory and time-uncritical program objects (POU) in the regular external memory.

In ctrlX PLC Engineering, each program object (POU) can be stored in the desired memory segment.

In ctrlX PLC Engineering, an editor to control the code storage is provided with the "POU Locations" object.

If no specifications are made, the program objects are stored sequentially in the code areas. This means, only when the fast code area ("area_0") is filled, are the following program objects stored in the slower code area ("area_1").

In the "POU Locations" editor, the current function block position in the memory areas is displayed. The program objects (POU) can be stored or changed per node or individually in the desired area (memory segment).

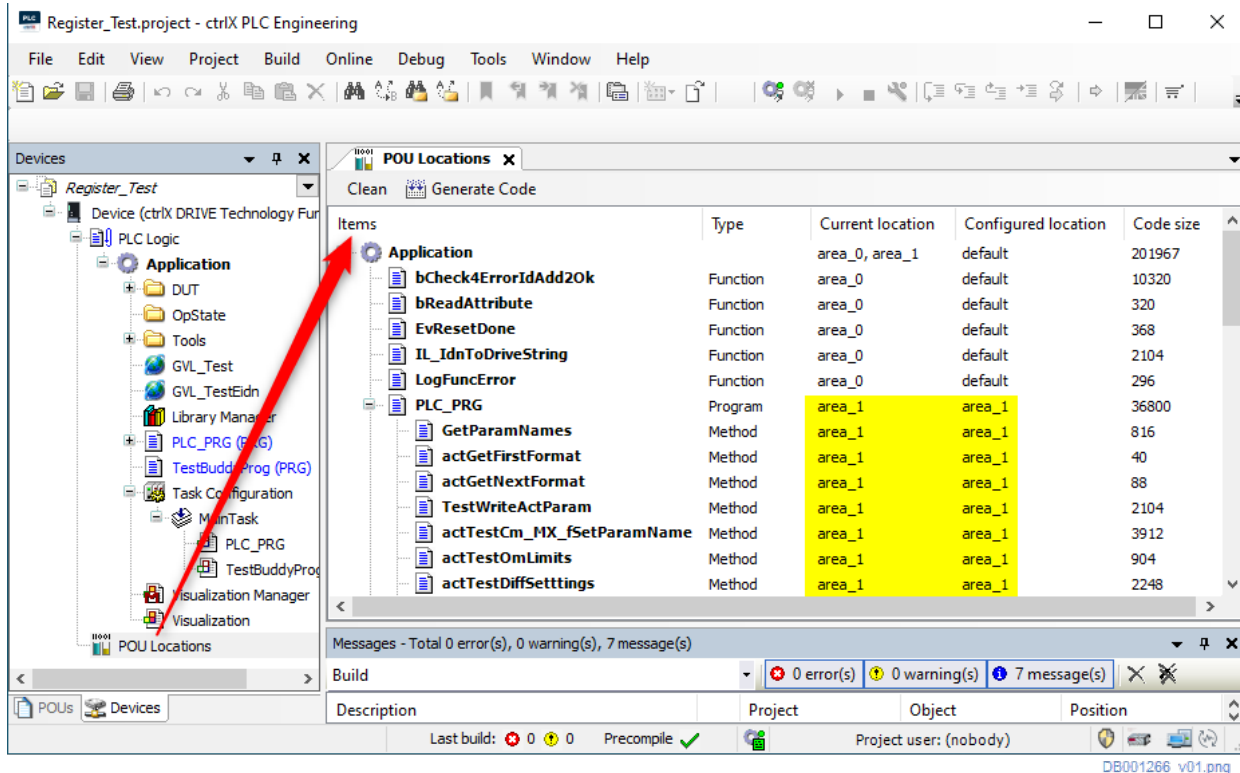


Fig. 5: ctrlX PLC Engineering, “POU Locations” branch

area_0 Fast internal memory for time-critical PLC code (default). This memory is automatically used for all program parts. If the project does not fit in this fast memory anymore, program objects (POU) are automatically moved to the second area.

area_1 Regular external memory for regular PLC code.



The operation of the editor is described in the ctrlX PLC Engineering documentation under „POU Locations“.

Parameters and diagnostics

The compressed application is distributed to the parameters P–0–1352 ... P–0–1358.

Uncompressed, 1024 kB + 384 kB = 1408 kB are available in the code memory, compressed $7 \times 64 \text{ kB} = 448 \text{ kB}$ are available in the parameters.

When the project is loaded to the drive, the compiled binary code is automatically stored in the parameters in compressed form. Depending on the degree of compression that can be achieved in the range of the code memory limit (1408 kB), the code sometimes might not fit in the parameters.

11.5.5 Global data

Accessing global data is possible in all function blocks.

Global data are normally used for cross communication between function blocks and can therefore relieve the call interface of function blocks. In addition, it is not necessary to adjust the call interface repeatedly in the case of modifications. This is to advantage in that the appearance is maintained and the user documentation of the PLC project does not have to be changed.

Global data are declared in global variable lists or in the declaration part of programming objects between the key words VAR_GLOBAL and END_VAR.

11.5.6 Code optimization

Requirements on the real-time behavior of the application

Drive control and drive-integrated PLC share the computing power of the drive, the drive control taking up a large part of the computing power of the drive. To use the available computing time in the best possible way, it is first necessary to check which demands are made on the real-time behavior of the application:

- Which functionalities have to be called in certain time intervals or in the case of certain events?
- What happens if a function is not completed at the desired point of time?

Task system

The first approach to the task configuration of the application results from the requirements on the real-time behavior of the application. Generally, a slower task should be used to separate functions with high runtimes from functions which have to run in real time. Each task requires computing time to manage and control the task system. The more tasks have been configured, the more computing time has to be spent for the management of the tasks.

Generally, only the necessary code should be processed in time-critical tasks. Too short intervals should not be used for the tasks.

Using tasks of different velocities only makes sense if relatively few actions have to be processed very fast and other actions run in a slower task (e.g., a task with $t\#1\text{ms}$ and a task with $t\#10\text{ms}$).

Besides, it is necessary to consider whether a watchdog already has to be triggered when a task is exceeded once, or whether the watchdog can be more tolerant.

As an alternative to a multi-task system, all fast actions can at first be carried out in a task and then only one slower action at a time (CASE instruction).

Beispiel

```
<Schnelle Aktionen>
CASE iCycle of
  0:
    <Langsame Aktion1>
  1:
    <Langsame Aktion2>
  2:
    <Langsame Aktion3>
END_CASE
iCycle := iCycle + 1;
IF iCycle > 2 THEN
  iCycle := 0;
END_IF
```

However, if the slower actions require more computing time, they have to be carried out in a separate task.

String processing

The processing of strings is relatively slow.

To process strings, optimized functions from the "String Functions" folder of the "CXA_Uilities" library can be used. These optimized functions have different call parameters and thus work significantly faster and require less stack memory.

Example: Instead of using the "INSERT()" function from the "Standard" library, it is recommended to use the "IL_INSERT()" function from the "CXA_Utility" library. (In the case of "IL_INSERT()", the last parameter transmitted is, for example, "sizeof(strMyText)-1".)

In the case of rapid task cycles, string processing should be avoided, if possible.

Accessing specific parameters

There are specific parameters that cannot be directly and quickly accessed (>50 ms instead of a few μ s). These parameters are, for example, parameters of safety technology or of the communication module.



Parameters that can be accessed using direct variables (see P-0-1260) usually remain unaffected.

IEC language selection

The programming language "Structured Text" (ST) is recommended for performance-optimized programming. "ST" also provides clearly structured programming.

Program code

- Bit processing is slightly slower than other computer operations. For example, this affects addressed variables (AT %M...) and bit accesses in words (wData.x).
- Division of integer variables is considerably slower than other computer operations, e.g. addition, subtraction, ...
- Division of REAL variables is slightly slower than the integer division.
- In the case of constants, multiplication by the reciprocal is preferred to division.

Using IEC constants

Using VAR CONSTANT is faster than using constant values in the form of normal variables. Variable addressing is not required, the value is directly used.

Parameter access

- Do not access parameters more often than necessary.
- Access from direct variables to parameters is significantly faster than function blocks.
- Access to list parameters requires a relatively long computing time and should either be outsourced to a slower task or carried out element-by-element using direct variables.
- Parameter access with the "Advanced" function blocks (IL_DriveReadElement, IL_DriveWriteElement) is faster than with the IL_DriveReadParameter or IL_DriveWriteParameter function blocks that are easier to handle.
- Access to string parameters requires a relatively long computing time – if necessary, execute this in a separate task.
- Access to parameters of subsystems (master communication, safety technology,...) should be executed in an individual task, since this requires a lot of computing time.

Function blocks

- Only call function blocks as long as necessary, especially cyclic function blocks.
- A constant value can already be assigned to a function block parameter at declaration.
- Outputs of function blocks can be directly read, it is not necessary to assign them to a variable.
- In the case of inputs ("VAR_INPUT") of complex data types, use "VAR_IN_OUT" instead of "VAR_INPUT" so that the data are not transmitted with every call. Please note that the function block accesses the original data.

Functions

Instead of "VAR_INPUT" inputs, better use "VAR_IN_OUT" so that the data do not have to be transmitted with every call. Please note that the function block accesses the original data.

Check function

The check function detects and avoids incorrect access. The checks are run for pointer access, array access, range limits, division by 0... Each check requires computing time. When the frequently processed code has been sufficiently checked, this automatic check (implicit check function) can be switched off on the user's own responsibility.

Switching off the implicit check function

To deactivate the checks within a POU, the following attribute can be inserted in front of the declaration:

```
{attribute 'no_check'}
FUNCTION MyFunc : INT
```

Structured programming

- Using subfunctions or subfunction blocks only provides higher performance if these functions/function blocks are used multiple times.
- By means of CASE instruction, different code branches can be systematically processed according to the status of the functionality or application.
- Transmission parameters / return values to functions and function blocks should be as compact as possible. In the case of large data quantities, access the data in global form, or transmit them via VAR_IN_OUT or REFERENCE.

Measurements

There are different functionalities for measuring the runtime:

- MX_lecTaskGetLoad() shows the percentage-based load of a cyclic task in relation to the ensured computing time.
- To measure individual actions, use the functions MX_fGetHighResTime() with MX_fHighResTimerTicks_to_us() in the CXAD_Base library.

Beispiel

```
VAR
  udiStart: UDINT;
  rDiffusec: REAL := 99999;
  rDiffusecMin: REAL;
  udiTestCycles: UDINT;
END_VAR
```

```
FOR udiTestCycles:= 1 TO 100000 DO
    udiStart := MX_fGetHighResTime();
```

```
<TestAktion>
```

```
    rDiffusec := MX_fHighResTimerTicks_to_us(MX_fGetHighResTime()-udiStart);
    rDiffusecMin := min(rDiffusec, rDiffusecMin);
END_FOR
```

"rDiffusecMin" provides the minimum runtime of the functionality.

Due to possible interruptions within the PLC time slice and cache effects in the processing of the test action, the minimum values of many calls should be used for evaluation.

11.5.7 Library template and style guide (PLC programming guidelines)

All functions, function blocks (POUs) and data types (DUTs) provided by Bosch Rexroth are sorted by topic in different PLC libraries. On the basis of PLCopen, these POUs and DUTs are standardized across systems with regard to their visual appearance and behavior.

A library template is provided in ctrlX PLC Engineering to create a new PLC application or PLC library. The template complies with the Bosch Rexroth style guide. Using the template or the style guide is mandatory for internal use at Bosch Rexroth and is recommended as a guideline to users of the ctrlX DRIVE Technology Function to create PLC libraries or PLC applications. Open the template in ctrlX PLC Engineering as follows:

1.  Start ctrlX PLC Engineering.
2.  Open the *"File → New project..."* menu.
3.  Select the "Libraries" category.
4.  In "Templates", select "Standard library".
5.  Assign a name for the new library and select a storage location.
6.  To confirm the selection, click "OK".
 - ➔ The new library is displayed in the "POUs" view.

The new library ("POUs" view) contains the following data:

- "ReadMe" text file: Describes the procedure for creating a general library from the library template
- Example templates for data types (DUTs), global variable lists and function blocks complying with the Bosch Rexroth style guide
- "_Styleguide" directory containing a PDF with system-wide programming guidelines for ctrlX AUTOMATION. The PDF can be directly opened in ctrlX PLC Engineering and, among other things, contains information about the following topics:
 - Documentation standardization
 - PLC variables and PLC types
 - Basic function blocks (standard interfaces, identifier description, standardized error handling)
 - Libraries

12 Technology Apps

12.1 Using Technology Apps as a self-contained function or as a library

PLC projects in ctrlX AUTOMATION are called "Technology App". Using Technology Apps in ctrlX DRIVE requires the functional package ctrlX DRIVE Technology Function to be activated. Also, a "TF1", "TE0", "TE1" or "TX1" license has to be available (see [➔ Enabling the functional package "ctrlX DRIVE Technology Function"](#)).

In addition to the option of freely programming the drive-integrated PLC with ctrlX PLC Engineering ("TE0", "TE1" and "TX1" licenses), the integrated PLC can be used for employing self-contained "Technology Apps" (TF1 license).

The following types of "Technology Apps" are available:

- **Technology App as a PLC program created by the user ("TE0", "TE1" license)**
- **Technology App as a self-contained technology function (TF1 license)**

The Technology App as a self-contained technology function is a compiled PLC project which is once loaded to the drive as a parameter file. The Technology App is operated like a "normal firmware function" via P-parameters and does not require any programming knowledge.

A Technology App as a self-contained technology function is made up of the following components:

- Ready-made PLC project loaded to the drive as a parameter file (P-0-1352 to P-0-1358).
 - Documentation of the function (including the parameters involved)
 - If required, a separate dialog (for ctrlX DRIVE Engineering, for example) for commissioning and operating the function.
- **Technology App as a library to be integrated (TX1 license)**

A Technology App as a self-contained technology function can be included in a freely programmed user program in the form of a library.

12.2 Loading Technology Apps

For loading Technology Apps, there is a dialog in ctrlX DRIVE Engineering. Using this dialog, the technology function stored in a parameter file (P-0-1352 to P-0-1358) can be loaded. (Loading without dialog support: Any previously loaded PLC project should be deleted before loading.)

If the retain data of the PLC (P-0-1359) also have been stored in the file, they will be used for loading. To reset the retain data, "reset (cold)" can be carried out in the PLC configuration.

12.3 Technology App ID

To clearly identify a Technology App, the list parameter P-0-1364 can be written by the Technology App.

If individual commissioning and diagnostics dialogs are supported in ctrlX DRIVE Engineering for the Technology App, the Technology App can be clearly identified with the "Technology App ID" (P-0-1364, element 0) and the "Vendor ID" (P-0-1364, element 1).

P-0-1364 is not buffered. Standard setting (default state): After the drive was switched on, all list elements of P-0-1364 have been initialized with "0".

Example of how to write P-0-1364 by the Technology App

The Technology App ID (P-0-1364) should be written in an initialization step of the Technology App. For this purpose, it is recommended to declare an individual data type (STRUCT):

```
// Technology App Identifier P-0-1364
TYPE TECH_APP_ID :
STRUCT
    uiActLength      : UINT;    // Actual length
    uiMaxLength      : UINT;    // Maximum length
    udiTechAppId     : UDINT;   // Element 0 - Technology App ID
    udiVendor        : UDINT;   // Element 1 - Vendor ID
    udiReleaseState  : UDINT;   // Element 2 - Release State
    udiVersionMajor  : UDINT;   // Element 3 - Major version
    udiVersionMinor  : UDINT;   // Element 4 - Minor version
    udiVersionSvcPack : UDINT;   // Element 5 - Service pack version
    udiVersionPatch  : UDINT;   // Element 6 - Patch
    udiReserved_1    : UDINT;   // Element 7 - Reserved
    udiReserved_2    : UDINT;   // Element 8 - Reserved
    udiReserved_3    : UDINT;   // Element 9 - Reserved
    udiReserved_4    : UDINT;   // Element 10 - Reserved
END_STRUCT
END_TYPE
```

In the declaration part, a structure variable of the "TECH_APP_ID" data type can be directly initialized with the data of the Technology App:

```
VAR
    stTechAppId: TECH_APP_ID:=
    (
        uiActLength      := SIZEOF(TECH_APP_ID)-4, // Size of TECH_APP_ID - act/max length
        uiMaxLength      := SIZEOF(TECH_APP_ID)-4, // Size of TECH_APP_ID - act/max length
        udiTechAppId     := 97816,                // Element 0 - Technology App ID
        udiVendor        := 1,                    // Element 1 - Vendor ID: Bosch Rexroth AG
        udiReleaseState  := 1,                    // Element 2 - Release State: Released
        udiVersionMajor  := 1,                    // Element 3 - Major version
        udiVersionMinor  := 2,                    // Element 4 - Minor version
        udiVersionSvcPack := 3,                    // Element 5 - Service pack version
        udiVersionPatch  := 4,                    // Element 6 - Patch
        udiReserved_1    := 0,                    // Element 7 - Reserved
        udiReserved_2    := 0,                    // Element 8 - Reserved
        udiReserved_3    := 0,                    // Element 9 - Reserved
        udiReserved_4    := 0,                    // Element 10 - Reserved
    );

    fbWriteElement: IL_DriveWriteElement; // Function block to write Technology
                                         // App Identifier (P-0-1364)
    stPrjVersion : VERSION;               // Technology App version from project
                                         // information object
END_VAR
```

If the version and the release state are to be updated via the "project information" object, it is also possible to read these data for the Technology App ID from the project information in the implementation part. For this purpose, the option "Automatically generate 'project information' POUs" must have been activated in the project information of ctrlX PLC Engineering.

In any case, the Technology App ID has to be written to P-0-1364 using the function block `IL_DriveWriteElement` (or alternatively via a direct variable). The code example below shows how the project information is applied to the structure variable `stTechAppId` and how P-0-1364 is written using an instance of the function block `IL_DriveWriteElement`.

```
// Get "Released-flag" from project information object
IF __POOL.GetBooleanProperty("Released") THEN
    // Released
    stTechAppId.udiReleaseState:= 1;
ELSE
    // Not yet released (test version)
    stTechAppId.udiReleaseState:= 0;
END_IF

// Get Technology App version from project information object
stPrjVersion:= __POOL.GetVersion();
stTechAppId.udiVersionMajor := stPrjVersion.uiMajor;
stTechAppId.udiVersionMinor := stPrjVersion.uiMinor;
stTechAppId.udiVersionServicePack:= stPrjVersion.uiServicePack;
stTechAppId.udiVersionPatch := stPrjVersion.uiPatch;

// Write Technology App Identifier (P-0-1364)
```

```
fbWriteElement( Execute      := TRUE,
                Element      := IL_DRIVE_ELEMENT.IL_OPDATA,
                Idn          := IL_DriveStringToIdn('P-0-1364'),
                SizeOfValue := SIZEOF(TECH_APP_ID),
                ValueAdr     := ADR(stTechAppId));
```

13 Diagnostic and service functions

13.1 Overview and introduction

This section describes the diagnostic and service functions that can be primarily used for troubleshooting. The options of free PLC programming and using ready-made ctrlX DRIVE Technology Function apps are explained.

Generally, the following classification is made:

- Standard drive diagnostic functions
- PLC diagnostic functions
- Service functions

13.2 Device editor

13.2.1 Device editor, general information

The device editor provides tabs for configuring a PLC device in ctrlX PLC Engineering.

The device editor opens after a double-click on the device object in the device tree ("Devices" view). ctrlX DRIVE Technology Function supports the following tabs:

- Communication
- Applications
- Log
- PLC shell
- Information



All other tabs are possibly not supported by the target ctrlX DRIVE Technology Function and cannot be used.

13.2.2 Communication

In the "Communication" tab, define the connection between ctrlX PLC Engineering and ctrlX DRIVE Technology Function.

To establish the connection, enter the IP address of the drive in the selection box to the right below the PLC device symbol, e.g. "192.168.0.9".

After the data have been entered, ctrlX PLC Engineering searches the device in the network of the ctrlX PLC Gateway.

The dot symbol at the bottom right at the Gateway or PLC device symbol provides information about the connection status to the Gateway / to ctrlX DRIVE Technology Function:

- Red: ctrlX PLC Engineering cannot establish the connection.
- Green: The connection has been established.
- Black: The connection status is unclear.



In the case of multi-axis devices with device-related IP communication, such as PROFINET or EtherNet/IP, the axes do not have their own IP address. In this case, an alternative port number after the default port 11740 has to be entered together with the IP address for each additional axis: For the second axis, for example: "192.168.0.9:11741" etc. The first axis can be addressed with "192.168.0.9" or "192.168.0.9:11740".

13.2.3 Applications

With ctrlX DRIVE Technology Function, only one application (Technology App) can be created. This application has the fixed name "Application".

In the "Applications" tab, the Technology App can be deleted or detailed information on the Technology App can be called.

13.2.4 Log

The PLC "logbook" can be found under the "Log" tab. The events recorded on the target are listed:

- Events when starting and stopping the system (loaded components with version)
- Application download and loading of the boot application

A logbook entry is displayed with the following pieces of information:

- Severity: There are five categories for the severity of the event: Warnings, errors, exceptions, information and debug messages. Each category can be shown or hidden by clicking the corresponding button in the row above the list. The buttons display the numbers of log entries of the corresponding categories.
- Time Stamp: Date and time (not supported in ctrlX DRIVE Technology Function)
- Description: Event description
- Component: Name of the affected component.

13.2.5 PLC shell

The "PLC shell" tab is reserved for the Bosch Rexroth service department. It contains extended diagnostic functions and a logging function that can be internally parameterized.

13.2.6 Information

The "Information" tab page shows general pieces of information about the ctrlX DRIVE Technology Function device:

- Name
- Manufacturer
- Categories
- Type
- ID
- Version
- ...

Normally, an image of the device is displayed.

13.3 Diagnostic functions

13.3.1 Standard drive diagnostic functions

Introduction

To display the time flows of variables from the PLC program, the standard drive diagnostic functions can be used. For example, PLC variables can be displayed for diagnostic purposes using the oscilloscope function or analog output in register parameters. Write access with direct variables is suited for this purpose.

Oscilloscope function

Using the oscilloscope function integrated in ctrlX DRIVE Engineering, time flows of drive parameters and up to 4 PLC variables can be displayed (see "➔ Chapter 13.3.2 Recording PLC variables with the oscilloscope function on page 69").

Analog output

The "analog outputs" drive function allows analog signal values to be used for commissioning and optimizing drives with appropriate measuring devices (e.g., oscilloscope, multimeter), as well as for visualizing the contents of drive parameters (see firmware Application Manual "Analog outputs").

Diagnostic trace

In the drive, all occurred errors of the drive and the PLC, together with the corresponding count of the operating hours counter, are collected in an error memory on the control section (see firmware Application Manual "Error memory (power section and control section)").

Panel

The ctrlX DRIVE panel provides the following options in conjunction with the ctrlX DRIVE Technology Function:

- PLC control: Starting/stopping/resetting ("reset origin") of the PLC project
- Reading out of the project name
- Reading out of detailed information (P-0-1365), if a PLC runtime error F6010 is present

13.3.2 Recording PLC variables with the oscilloscope function

With the oscilloscope function in ctrlX DRIVE Engineering it is possible to record PLC variables.

The PLC variables to be recorded have to be configured beforehand in the symbol configuration in ctrlX PLC Engineering.

Prerequisites:

- A connection must have been established between the PC and the drive.
 - A PLC project with PLC variables has to be available.
1. ➔ In ctrlX DRIVE Engineering, open the project the PLC variables of which are to be recorded with the oscilloscope function.
 2. ➔ In the device tree, select the "Application" object.
 3. ➔ Select the command *"Project → Add object → Symbol configuration"*.
In the following "Add symbol configuration" dialog, you can use the default settings.
Click "Add".
 - ➔ The "Symbol configuration" object is added to the device tree, and the symbol configuration editor is opened.
 4. ➔ Open the *"View"* menu of the editor and activate all variable categories. Then select the *"Build"* command in the symbol configuration editor.
 - ➔ All variables in accordance with the filters currently set in the *"View"* menu are displayed in a tree structure.
 5. ➔ Select the PLC variables, that are to be recorded with the oscilloscope function, by activating the corresponding check boxes.

6. ➤ In the main menu of ctrlX DRIVE Engineering, select the *“Build → Generate code”* command.
7. ➤ With the *“Online → Login”* command, load the application to the device.
 - ➔ The PLC variables are now available in the oscilloscope function of ctrlX DRIVE Engineering as a signal source.
8. ➤ Go to ctrlX DRIVE Engineering.
9. ➤ In ctrlX DRIVE Engineering, select the *“Diagnostics → Oscilloscope”* menu.
 - ➔ The oscilloscope function is opened in a separate window.
10. ➤ In the “Current measurement” section, click the “Signals” button.
 - ➔ The “Configure signals” dialog is opened.
 All signal sources that can be selected are listed in the left half of the dialog. In the right half, the selected signal sources are listed. It is possible to select up to 8 signal sources, thereof at most 4 PLC variables.
 ⓘ Information on the signal sources that can be selected:
 PLC variables with a data length of 8 bytes, as well as variables of the "String" type, cannot be selected for oscilloscope recording.
11. ➤ Expand the “PLC variables” branch, e.g. by double-clicking “PLC variables” or clicking “+”.
12. ➤ Select the PLC variables you would like to record by selecting the PLC variable on the left side and applying it to the list of selected signal sources with the “>” button.
13. ➤ Close the “Configure signals” dialog by clicking the “OK” button.
 It is now possible to record the PLC variables.



Information on the trigger signal:

A maximum of four channels is available for recording PLC variables. If a PLC variable is to be used as a trigger signal, only 3 more PLC variables can be recorded with the oscilloscope.

13.3.3 PLC diagnostic functions

Overview

In ctrlX PLC Engineering and ctrlX DRIVE Engineering, there are other PLC-specific diagnostic functions available (for detailed descriptions of the functions, see the respective chapter of the ctrlX PLC Engineering Application Manual):

- **Recipe manager**
 Using the recipe manager, variables can be loaded and saved in combined form.
- **Visualization**
 PLC variables can be graphically represented or changed via the ctrlX PLC Engineering user interface.
- **Monitoring functions**
 Both the runtime and the available resources are monitored.
- **Simulation**
 The PLC logic, for example, can be tested using simulation.
- **Debugging** (breakpoints, watch, single step, write)
- **PLC status information**
 PLC control and status information are contained in the parameters P-0-1350 and P-0-1351.

Recipe manager

Use the recipe manager to display the values of specific variables. It is also possible to preassign specific values to variables and transmit them altogether to the control.

Visualization

During commissioning, the integrated visualization is well suited for graphically representing PLC variables or change them via the ctrlX PLC Engineering user interface.

Monitoring functions

In the drive-internal PLC, both the runtime and the available resources are monitored in order to detect unwanted exceedance.

In the case of unauthorized access, a PLC exception is generated. These errors cause a drive error F6010 with axis deceleration.

Simulation

In the simulation mode, the connection to the drive controller is not established. When logging in, the PLC is (partly) simulated on the PC. In this case, there are no real time and no firmware function blocks. The PLC logic, for example, can be tested using simulation.

Debugging / troubleshooting

For troubleshooting and testing the created PLC programs, a debugger on source code basis has been integrated in ctrlX PLC Engineering. This allows breakpoints to be set and the tests to be carried out in individual steps. Variables can be displayed or overwritten.



- It is recommended to use a counter in each task. Thus, it is possible to easily monitor if the code is processed. Besides, it shows how often and how fast a task is running.
- The flow control as well as forcing of values is not supported in ctrlX DRIVE Technology Function.

PLC control and status words

The ctrlX DRIVE Technology Function can be controlled via the PLC control word control bits (P-0-1350), irrespective of the PLC programming system.

The processing of control commands via P-0-1350 is indicated by acknowledgement bits in the PLC status word (P-0-1351). Thus, it is possible to display the status of control commands as well as whether or not they were completed successfully.

Furthermore, the PLC status word (P-0-1351) displays the PLC (and the ctrlX DRIVE Technology Function) state.

PLC control word (P-0-1350)

By means of the control bits of the PLC control word (P-0-1350), the appropriate command to control the ctrlX DRIVE Technology Function can be started with a positive edge.

Only one command should be started at a time, do not start multiple commands simultaneously.

After a control bit has been set, wait for the 0-1 edge of the "done" or "error" bit in the PLC status word (P-0-1351). Subsequently, the control bit in P-0-1350 can be deleted again.



For the assignment of the P-0-1350 bits, please refer to the Parameter Description.

PLC status word (P-0-1351)

The PLC status word (P-0-1351) can only be read, not written. P-0-1351 is a 16-bit status parameter of the ctrlX DRIVE Technology Function.

The acknowledgement bits of P-0-1351 (bit 10="active", bit 11="done", bit 12="error") show whether a command started by the PLC control word (P-0-1350) is currently being processed ("active") and whether it was completed successfully ("done") or with an "error".

After a control bit was set, "active" shows that the command is being processed. As soon as the command execution is finished, "active" is reset and either set to "done" or "error". Sometimes, the command execution is finished immediately, and "done" or "error" is set instead of "active".

The "done" and "error" bits are reset by a falling edge of the control bit.

If the user resets the control bit before the command execution has been completed, the "done" or "error" bit is only displayed for a short time and might not be visible externally.

If the user sets another control bit while a command is processed or sets multiple control bits in parallel, the commands are processed one after the other and the acknowledgement bits are reset by clearing all control bits. In this case, the status of the command execution cannot be unequivocally identified.



For the assignment of the P-0-1351 bits, please refer to the Parameter Description.

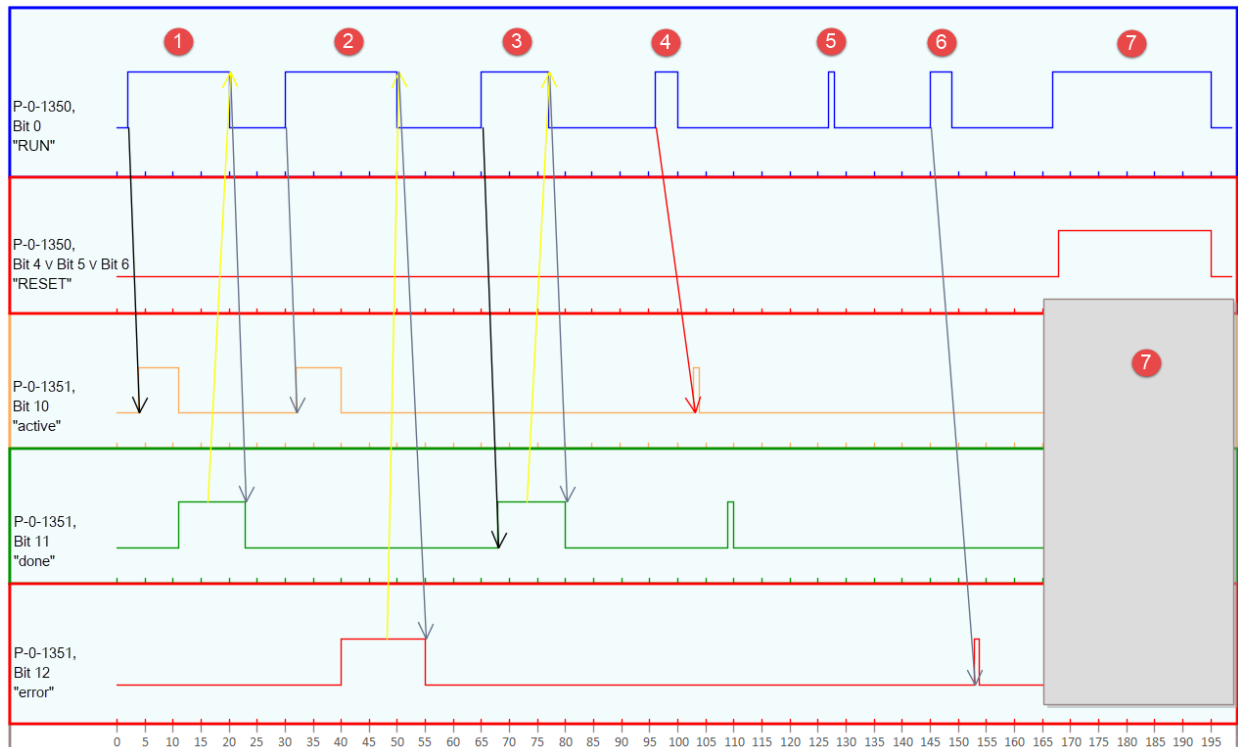


Fig. 6: Timing of the status bits

- | | |
|--|--|
| <ol style="list-style-type: none"> 1 Procedure with runtime 2 Procedure with runtime and error 3 Procedure without runtime 4 "Sufficient" pulse on control bit with subsequent acknowledgement bits 5 Pulse on control bit too short without acknowledgement bits | <ol style="list-style-type: none"> 6 "Sufficient" pulse on control bit with subsequent error bit 7 Another control bit was set during the command execution or multiple control bits were set simultaneously. The status of the command execution cannot be unequivocally identified |
|--|--|

The "error" bit is signaled in the following cases, for example:

- PLC "STOP" was executed without loaded project
- PLC "RUN" was executed without loaded project
- If PLC "STOP" or PLC "RUN" is executed with a loaded project and the "error" bit is still signaled, an internal error occurred during the command execution has occurred. In such a case, please contact our service department.



PLC "RESET" signals "done" even without a loaded project.

If PLC "RESET" or PLC "STOP" is executed and a task with a long task runtime is active at this point of time, "active" is set for a longer time.

Tracing PLC exceptions

General information

In the case of PLC exceptions, such as division by zero or exceeding of the range, the corresponding position in the PLC source code can be found very quickly with the target ctrlX DRIVE Technology Function.

Troubleshooting in the case of runtime errors

The error "F6010 PLC runtime error" is generated in the case of a PLC runtime error. In this case, the programming interface jumps to the faulty program code and highlights it with a color. The parameter P-0-1365 contains information

on the cause of error and on the affected POU (Program Organization Unit) (this information is also displayed in ctrlX PLC Engineering in the "Messages" window).

How to proceed after an exception without ctrlX PLC Engineering at the drive, before the drive is switched off

If an exception occurs and you are not logged in in ctrlX PLC Engineering at the time when the exception occurs, it is possible to log in later in ctrlX PLC Engineering. The error is displayed with a color. However, this requires that the drive was not switched off in the meantime.

In the case of error and with the parameter backup of all parameters, it is also possible to get more detailed information offline on the position of the error with the PLC code. This requires support by the Bosch Rexroth service department.

How to proceed in the case of an exception after the drive was switched off ("Replay Exception")

When an exception occurs, the position causing the exception can be displayed in the project, even after the drive was switched off. For this purpose, there is the option to trigger the error again using the command "Replay Exception" (P-0-1350, bit 8). An exception can also be triggered on another device.

Starting

Before the "Replay Exception" command (P-0-1350, bit 8) is started to again trigger an exception on another device, the following prerequisites have to be met:

- The identical PLC project (from the same compilation process) must have been loaded on the device. (Logging in should therefore be possible without download.)
- At least the parameter P-0-1362 must have been copied to the device.
- The PLC has to be in the "STOP" state.

When the mentioned prerequisites have been fulfilled, the exception can be triggered again with the "Replay Exception" command (P-0-1350, bit 8).



The execution of the "Replay Exception" command must not be executed in combination with other command bits.

When the command is executed, it is not required to be logged in in ctrlX PLC Engineering, but it is useful in order to have the position of the error displayed.

Reactions

When the "Replay Exception" command was successfully executed, the command execution is acknowledged with "Done" in P-0-1351, bit 11, and the required pieces of information are entered in P-0-1362.

When the "Replay Exception" command is executed, the drive reactions are identical to the ones of the exception.

To see that the exception was triggered by the "Replay Exception" command, a prefix text is contained in P-0-1365: "Replay: ".

If one of the prerequisites for the execution of the "Replay Exception" command is missing, the command execution is acknowledged with "Error" in P-0-1351, bit 12.

Restrictions

There are the following restrictions when replaying the exception on another device:

- Local variables of functions are not displayed with the original values.
- In exceptional cases, the exact position of the error might not be displayed. In this case, please contact the Bosch Rexroth support.

13.4 Service functions

13.4.1 PLC service functions

Service function in ctrlX PLC Engineering

ctrlX PLC Engineering makes available the service function:

- **"Download"**
refer to "Loading the ctrlX PLC Engineering project to the drive"
- The application code is generated with "Generate code". In the message window, detailed information is output on the memory assignment. Please note that you can exclude objects from compilation.
- **"Clean"**
"Clean" deletes the compiled information for the application. After a cleaning process, the application has to be recompiled and reloaded to the control.
- **"Clean all"**
"Clean all" deletes all pieces of information and again generates, among other things, the language model for all objects. This takes additional time.
Note: To complete a commissioning process, it is recommended to execute "Clean all" so that the complete project is generated again. Besides, it is useful to store an archive of the project.
"Clean all" deletes information of the last download and the last compilation process.
- **Reset**
 - "Reset warm": Resets the program; variables are initialized, retain variables retain their values
 - "Reset (cold)": Resets the program; variables and retain variables are initialized
 - "Reset (origin)": Program and boot project are deleted

PLC behavior after a processor exception

A processor exception that is not intercepted can occur due to an incorrect statement. The PLC behavior after a processor exception is described [↗ here](#).

13.4.2 Firmware replacement

In the case of firmware replacement, always make sure that the package used in the PLC project is compatible with the new firmware version (see [↗ Chapter 11.5.2 Compatibility of ctrlX DRIVE Technology Function with the drive firmware on page 58](#)).

Notes on modifications, enhancements and resolved defects are contained in the Firmware Release Notes.

14 Documentations

14.1 Drive systems, system components

Table 1: Documentations – Drive systems, System Components

Title	Type of documentation	Document type ¹⁾	Material number
ctrlX DRIVE Drive Systems	Project Planning Manual	DOK-XDRV**-X*****- PRxx-EN-P	➔ R911386579
ctrlX DRIVE DC/DC Converter XMV	Application Manual	DOK-XDRV**-XMV*****- APxx-EN-P	➔ R911413650
Security Instructions Electric Drives and Controls	Project Planning Manual	DOK-IWORKS-SECURITY***- PRxx-EN-P	➔ R911342562
Control Cabinet Air Conditioning, EMC, Design, IP Code, IndraDrive Electrics, Rexroth EFC/Fv, Sytronix	Project Planning Manual	DOK-DRIVE*-CABINET****- PRxx-EN-P	➔ R911344988

1) In the document type codes, "xx" is a placeholder for the current edition of the documentation (e.g.: PR01 is the first edition of a Project Planning Manual)

14.2 Firmware/Runtime

Table 2: Documentations – firmware

Title	Type of documentation	Document type ¹⁾	Material number
ctrlX DRIVE AXS-V-06 Functions	Application Manual	DOK-XDRV**-AXS-06VRS**- APxx-EN-P	➔ R911425358
ctrlX DRIVE AXS-V-06 (CoE) Functions	Application Manual	DOK-XDRV**-AXS-06VRS*C- APxx-EN-P	➔ R911425360
ctrlX DRIVE / eLION Diagnostic Messages of Runtime A*S-V-06RS	Reference Book	DOK-XDRV**-GEN6-DIAG**- RExx-EN-P	➔ R911424817 ²⁾
ctrlX DRIVE / eLION Parameters/Objects of Runtime A*S-V-06RS	Reference Book	DOK-XDRV**-GEN6-PARA*C- RExx-EN-P	➔ R911425449 ²⁾
ctrlX DRIVE AXS-V-05 Functions	Application Manual	DOK-XDRV**-AXS-05VRS**- APxx-EN-P	➔ R911422255
ctrlX DRIVE AXS-V-05 (CoE) Functions	Application Manual	DOK-XDRV**-AXS-05VRS*C- APxx-EN-P	➔ R911422257
ctrlX DRIVE Diagnostic Messages of Runtime AXS-V-05RS	Reference Book	DOK-XDRV**-GEN5-DIAG**- RExx-EN-P	➔ R911422251 ²⁾
ctrlX DRIVE Parameters/Objects of Runtime AXS-V-05RS	Reference Book	DOK-XDRV**-GEN5-PARA*C- RExx-EN-P	➔ R911422253 ²⁾

Title	Type of documentation	Document type ¹⁾	Material number
ctrlX DRIVE AXS-V-04 Functions	Application Manual	DOK-XDRV**-AXS-04VRS**- APxx-EN-P	➔ R911421281
ctrlX DRIVE AXS-V-04 (CoE) Functions	Application Manual	DOK-XDRV**-AXS-04VRS*C- APxx-EN-P	➔ R911421283
ctrlX DRIVE Diagnostic Messages of Runtime AXS-V-04RS	Reference Book	DOK-XDRV**-GEN4-DIAG**- RExx-EN-P	➔ R911421277 ²⁾
ctrlX DRIVE Parameters/Objects of Runtime AXS-V-04RS	Reference Book	DOK-XDRV**-GEN4-PARA*C- RExx-EN-P	R911421279
ctrlX DRIVE AXS-V-03 Functions	Application Manual	DOK-XDRV**-AXS-03VRS**- APxx-EN-P	➔ R911410073
ctrlX DRIVE AXS-V-03 (CoE) Functions	Application Manual	DOK-XDRV**-AXS-03VRS*C- APxx-EN-P	➔ R911398021
ctrlX DRIVE Diagnostic Messages of Runtime AXS-V-03RS	Reference Book	DOK-XDRV**-GEN3-DIAG**- RExx-EN-P	➔ R911409763 ²⁾
ctrlX DRIVE Parameters of Runtime AXS-V-03RS	Reference Book	DOK-XDRV**-GEN3-PARA**- RExx-EN-P	➔ R911409808 ²⁾
ctrlX DRIVE Parameters/Objects of Runtime AXS-V-03RS	Reference Book	DOK-XDRV**-GEN3-PARA*C- RExx-EN-P	➔ R911419643 ²⁾
ctrlX DRIVE AXS-V-02 Functions	Application Manual	DOK-XDRV**-AXS-02VRS**- APxx-EN-P	➔ R911398021
ctrlX DRIVE Diagnostic Messages of Runtime AXS-V-02RS	Reference Book	DOK-XDRV**-GEN2-DIAG**- RExx-EN-P	➔ R911383776
ctrlX DRIVE Parameters of Runtime AXS-V-02RS	Reference Book	DOK-XDRV**-GEN2-PARA**- RExx-EN-P	➔ R911383778

Legend:

1: In the document typecodes, xx is a placeholder for the current edition of the documentation (e.g.: RE02 is the second edition of a Reference Book)

2: This documentation is only available as an online help (on "ctrlX AUTOMATION – Product Information") and not in the PDF format

14.3 Functional safety

Table 3: Documentations – functional safety

Title	Type of documentation	Document type ¹⁾	Material number
ctrlX SAFETY "STO" (S-/P-Parameters) ctrlX DRIVE "2nd Generation"	Application Manual	DOK-XDRV**-SI-TX*****- APxx-EN-P	➔ R911383774
ctrlX SAFETY "STO" (CAN Parameters) ctrlX DRIVE "2nd Generation"	Application Manual	DOK-XDRV**-SI-TX***C**- APxx-EN-P	➔ R911426739
ctrlX SAFETY "SafeMotion" (S-/P-Parameters) ctrlX DRIVEplus	Application Manual	DOK-XDRV**-SI-MX*****- APxx-EN-P	➔ R911404905
ctrlX SAFETY "SafeMotion" (CAN Parameters) ctrlX DRIVEplus	Application Manual	DOK-XDRV**-SI-MX***C**- APxx-EN-P	➔ R911426737

1) In the document typecodes, xx is a placeholder for the current edition of the documentation (e.g.: AP02 is the second edition of an Application Manual)

14.4 Motors

Table 4: Documentations – motors

Title	Type of documentation	Document typecode ¹⁾	Material number
MS2N Synchronous Servomotors	Project Planning Manual	DOK-MOTOR*-MS2N*****- PRxx-EN-P	➔ R911347583
MS2S Synchronous Servomotors	Project Planning Manual	DOK-MOTOR*-MS2S*****- PRxx-EN-P	➔ R911410075
MS2E Synchronous Servomotors acc. to ATEX Directive 2014/34/EU	Project Planning Manual	DOK-MOTOR*-MS2E*****- PRxx-EN-P	➔ R911394140
MSK Synchronous Servomotors	Project Planning Manual	DOK-MOTOR*-MSK*****- PRxx-EN-P	➔ R911296289
MSK Synchronous Servomotors for Potentially Explosive Areas	Project Planning Manual	DOK-MOTOR*-MSK*EXGIK3- PRxx-EN-P	➔ R911312709
MKE Synchronous Motors Synchronous Servomotors acc. to ATEX Directive 2014/34/EU	Project Planning Manual	DOK-MOTOR*-MKE*GEN3***- PRxx-EN-P	➔ R911411017
MAD / MAF Asynchronous Motors MAD / MAF	Project Planning Manual	DOK-MOTOR*-MAD/MAF****- PRxx-EN-P	➔ R911295781
MLF Synchronous Linear Motors	Project Planning Manual	DOK-MOTOR*-MLF*****- PRxx-EN-P	➔ R911293635
ML3 Self-Cooled Linear Motors	Project Planning Manual	DOK-MOTOR*-ML3*****- PRxx-EN-P	➔ R911389760
MCL Ironless Linear Motors MCL	Project Planning Manual	DOK-MOTOR*-MCL*****- PRxx-EN-P	➔ R911330592

1) In the document type codes, "xx" is a placeholder for the current edition of the documentation (e.g.: PR01 is the first edition of a Project Planning Manual)

14.5 Cables

Table 5: Documentations – Cables

Title	Type of documentation	Document type ¹⁾	Material number
ctrlX Motor Cables and Connectors	Reference Book	DOK-CONNEC-XDRV*****-RExx-EN-P	➔ R911420100
Motor cables and connections with IndraDrive	Product information	DOK-CONNEC-MS2N*INDRV*-CAxx-EN-P	➔ R911401938
Rexroth Connection Cables IndraDrive and IndraDyn	Selection Data	DOK-CONNEC-CABLE*INDRV-CAxx-EN-P	➔ R911322949

1) In the document type codes, xx is a placeholder for the current edition of the documentation (e.g.: CA03 is the third edition of the Catalog documentation)

15 Service and support

Our worldwide service network provides an optimized and efficient support. Our experts provide you with advice and assistance. You can contact us **24/7 - including weekends and public holidays**.

Service Germany

Our technology-oriented Competence Center in Lohr, Germany, is responsible for all your service-related queries for electric drive and controls.

Contact the **Service Hotline** and **Service Helpdesk** under:

Tel: **+49 9352 40 5060**
 Fax: **+49 9352 18 4941**
 Email: ↗ service.svc@boschrexroth.de
 Internet: ↗ <http://www.boschrexroth.com>

Additional information on service, repair (e.g. delivery addresses) and training can be found on our internet sites.

Service worldwide

Outside Germany, please contact your local service office first. For hotline numbers, refer to the sales office addresses on the internet.

Preparing information

To be able to help you more quickly and efficiently, please have the following information ready:

- Detailed description of malfunction and circumstances
- Type plate specifications of the affected products, in particular type codes and serial numbers
- Your contact data (phone and fax number as well as your e-mail address)

Furthermore, please backup all parameters or generate a system report in ctrlX DRIVE Engineering ("Help" → "Generate system report").

16 Glossary

Application Manual

The application manual comprises the entire documentation which is used to provide information to the user of the product about the use and the safety-relevant contents for project planning, assembly, installation, mounting, commissioning, operation, maintenance, repairs and decommissioning of the product. The following terms are used for the application manual: Operating Instructions, Commissioning Manual, Instruction Manual, Project Planning Manual, Application Manual, etc.

Axis processor

The axis processor is a microprocessor in which the control algorithms for operating the actuator (e.g., a motor) are running.

Compatibility

Compatibility of a new function or functional enhancement means that, for example, a parameter file from a previous version can be used in the new firmware.

Component

A component is a combination of assembly parts with a specified function which are part of the equipment, the device or the system. Components of the electric drive and control system are e.g. supply units, drive control devices, mains choke, mains filter, motors, cables, etc.

ctrlX AUTOMATION

ctrlX AUTOMATION is the open and scalable automation platform; it breaks down the conventional boundaries between machine control, IT world and Internet of Things.

ctrlX CORE

ctrlX CORE is the product line of the compact control platform of ctrlX AUTOMATION.

ctrlX CORE is available in embedded form, drive-integrated form or in the IPC.

ctrlX DRIVE

ctrlX DRIVE is the product line of the compact modular drive system of ctrlX AUTOMATION.

ctrlX DRIVE Engineering

ctrlX DRIVE Engineering is the software used to configure and commission the ctrlX DRIVE drive system.

ctrlX DRIVE Technology Function

ctrlX DRIVE Technology Function is the PLC firmware function that allows customized PLC programs or ready-made Technology Apps to be used in the axis processor of the ctrlX DRIVE drive system.

ctrlX DRIVEplus

ctrlX DRIVE is the product line of the compact modular drive system of ctrlX AUTOMATION.

With ctrlX DRIVEplus, the drives can be extended by additional software functions and hardware.

ctrlX PLC Engineering

ctrlX PLC Engineering is the development environment in accordance with the IEC 61131-3 standard for programmable logic controllers of ctrlX AUTOMATION.

ctrlX SAFETY

ctrlX SAFETY is an umbrella term for the safety technology product lines of ctrlX AUTOMATION.

ctrlX SAFETY refers to compact safety controllers of the SAFEX-C.12 and SAFEX-C.15 types.

Other characteristics of safety technology product lines are ctrlX DRIVEplus and ctrlX SAFETYplus.

ctrlX WORKS

ctrlX WORKS is the central software of ctrlX AUTOMATION. In ctrlX WORKS, all available ctrlX devices in the network are visible.

Furthermore, ctrlX WORKS can be used to install and start app-based and web-based engineering and programming tools for typical automation tasks. It is possible to develop your own applications in any programming language, and third-party apps can be integrated in ctrlX WORKS.

Device

A device is an end product with an individual function, intended for the user and put on the market as individual commodity.

Drive

A drive (electric drive) consists of a drive controller with an electric motor.

EnDat

EnDat® is a registered trademark of -Dr. Johannes Heidenhain GmbH.

EtherNet/IP

EtherNet/IP™ is a trademark under license of ➔ [Open DeviceNet Vendor Association, Inc](#)

Installation

An installation consists of multiple devices or systems interconnected for a defined purpose and at a defined location. However, these devices or systems are not intended to be put on the market as a single functional unit.

Machine

The term machine refers to the entirety of the interconnected parts or components. At least one part or one component is movable. A machine consists of machine drive elements as well as control circuits and energy circuits, connected for a certain application. A machine is, for example, intended for processing, handling, movement or packaging of a material. The term “machine” also covers a combination of machines that are arranged and controlled to function as a unitary whole.

Manufacturer

The manufacturer is an individual or legal entity assuming responsibility for the design and manufacturing of a product which is put on the market in the individual's or legal entity's name. The manufacturer can use finished products, finished parts or finished elements or externally subcontract work. However, the manufacturer is obliged to supervise manufacturing of the product and has to be authorized to assume responsibility for the product.

Package

A package is an installable artifact that contains one or more software artifacts (1..n) which can be used on a device.

A package can contain, for example, firmware artifacts, applications, templates and recipes. Packages are customized features a customer can buy and/or install.

Patch

A patch corrects errors in the firmware.

Product

Example of a product: Device, component, part, system, software, firmware, among other things.

PROFINET

PROFINET® (Process Field Network) is the open Industrial Ethernet standard of ➔ [Profibus & Profinet International \(PI\)](#) for automation.

PROFINET® is registered trademark of PROFIBUS Nutzerorganisation e. V.

Project Planning Manual

A Project Planning Manual is part of the application documentation used to assist in the sizing and planning of systems, machines or installations.

Retain parameters

Retain parameters contain status variables; their last value is automatically saved when switching off.

Furthermore, the following aspects apply to ctrlX DRIVE Runtime: Before a device is replaced, retain parameters have to be saved and loaded to the replacement device to ensure that operation continues smoothly. Retain parameters are listed in "Process data, retain: IDN list of retain data" (0x2B6E:01 / P-0-0195 / --).

Sercos

Sercos® is registered trademark of ➔ [Sercos International e.V.](#)

State machine

The state transitions and the interaction of the control and status bits are called state machine.

A distinction is made between the *device-internal state machine* (defines the device-specific states which determine the behavior of the device) and the *state machine of master communication*.

User

A user is a person installing, commissioning or using a product which has been placed on the market.

17 Index

B

Boot project. 56

C

Cables

Documentation. 79

Configuration device. 67

ctrlX WORKS initial installation. 12

D

Device configuration. 67

Device editor. 67

Documentation

Cables. 79

Drive systems. 76

Firmware. 76

Motors. 78

Runtime. 76

System components. 76

F

Firmware

Documentation. 76

H

Helpdesk. 80

Hotline. 80

L

Libraries

General properties. 54

Supported libraries. 55

M

Motor

Documentation. 78

P

Panel. 69

PLC programming

Global data. 60

R

Runtime

Documentation. 76

S

Service hotline. 80

Start behavior. 56

Structuring PLC projects

Multi-program technology. 58

Support. 80

T

Task monitoring. 23

Task properties. 22

Task system. 21, 65

Technical data. 8

Technology App

Using. 65

Technology Apps

ID. 65

Loading. 65

W

Watchdog. 23

Bosch Rexroth AG
Bgm.-Dr.-Nebel-Str. 2
97816 Lohr a.Main
Germany
Tel. +49 9352 18 0
Fax +49 9352 18 8400
www.boschrexroth.com/electrics



R911409771 04