

TM213

Automation Runtime



Requirements

Training modules	TM210 – Working with Automation Studio
Software	Automation Studio 4.4 Automation Runtime 4.44
Hardware	X20 controller and X20 I/O modules ETA210 or ETAL210 + ETAL690 www.br-automation.com/eta-system



Table of contents

1 Introduction.....	4
1.1 Learning objectives.....	5
1.2 Symbols and safety notices.....	5
2 Automation Runtime.....	6
2.1 Automation Runtime structure.....	8
2.2 Automation Runtime diagnostics capability.....	11
2.3 Connectivity and access & security.....	11
3 Installation and commissioning.....	13
3.1 Changing and updating Automation Runtime.....	14
3.2 Configuration ID and partitioning.....	15
3.3 Project installation.....	16
4 Memory management.....	19
4.1 Logical partitioning of the Flash memory.....	19
4.2 Used RAM.....	20
4.3 Tools for determining memory information.....	21
4.4 Global and local variables.....	24
5 Runtime performance.....	30
5.1 Starting Automation Runtime.....	30
5.2 Program initialization.....	36
5.3 Executing cyclical programs.....	37
5.4 Cycle time and tolerance.....	42
5.5 Settings when transferring programs.....	46
6 I/O handling.....	49
6.1 I/O configuration and I/O mapping.....	50
6.2 Handling I/O images.....	52
6.3 Error handling for I/O modules.....	58
6.4 reACTION Technology.....	59
7 Summary.....	61

Introduction

1 Introduction

Automation Runtime is a deterministic real-time operating system that serves as the software platform for B&R's entire product range. Automation Runtime provides services for freely configuring and troubleshooting the target system and for executing programs.

In addition, Automation Runtime features a modular structure, configurability and the ability to quickly execute applications repeatedly within a precise time frame. This makes it possible to achieve optimal product quantities while guaranteeing quality and precision at runtime.



Figure 1: Automation Runtime: A software platform for the entire B&R product range

This training module will provide a general overview of Automation Runtime and its features and configuration options.

1.1 Learning objectives

This training module uses selected examples illustrating typical applications to help participants learn how to configure Automation Runtime for their own applications in Automation Studio.

- Participants will learn what demands are placed on a real-time operating system in the field of automation.
- Participants will learn about the features and functionalities available in Automation Runtime.
- Participants will get an overview of integrated server and client functions and diagnostic options.
- Participants will learn how a uniform runtime system can benefit integrated automation solutions.
- Participants will learn how Automation Runtime handles memory management and variable scope and how variables are stored in memory.
- Participants will learn about the startup and runtime behavior of B&R controllers.
- Participants will learn how Automation Runtime I/O management works.
- Participants will learn about the interrelationships involved in multitasking.
- Participants will learn about Automation Runtime's configuration options.

1.2 Symbols and safety notices

Unless otherwise specified, the descriptions of symbols and safety notices listed in "TM210 – Working with Automation Studio" apply.

Automation Runtime

2 Automation Runtime

The fundamental idea of integrated automation and the free scalability of automation solutions results in challenges for the configuration tool as well as the runtime system it's based on.

Requirements for Automation Studio and Automation Runtime

Automation Studio is the project development environment used specifically for B&R automation components. This includes controllers, motion control components, safety modules and HMI applications. Clearly organized project structuring options and the ability to manage multiple configurations ensure that teams can work together efficiently and all machine variants can be represented in a single project.

Users are able to choose from a wide range of programming languages, diagnostic tools and editors to assist them at every stage of engineering. Standard libraries provided by B&R and the integrated IEC programming languages allow for a highly efficient workflow. Extensive simulation options enable applications to be configured and tested independently of the hardware.

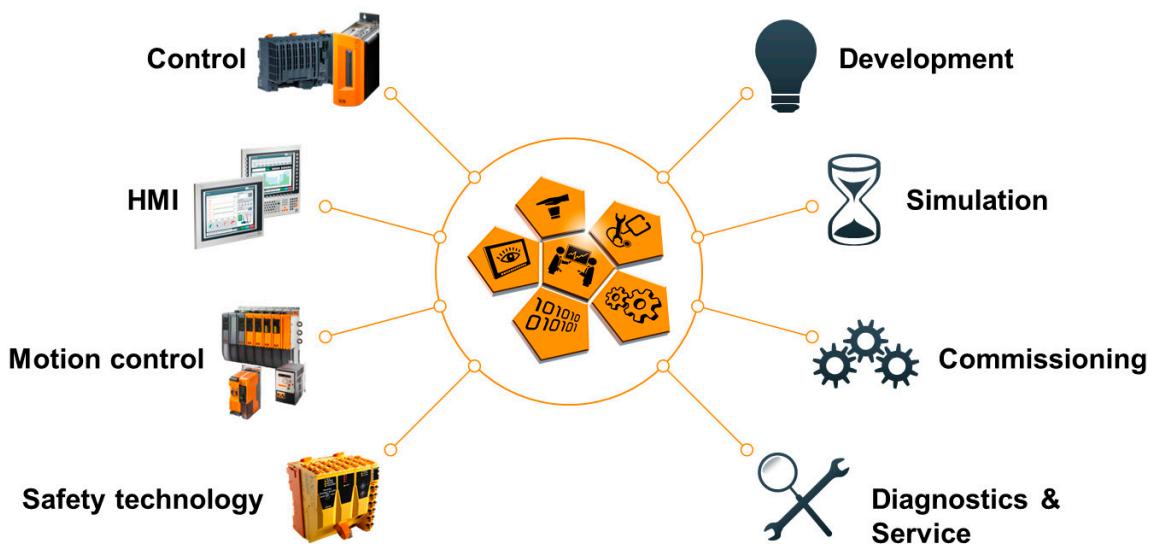


Figure 2: Automation Studio: one engineering tool for the machine's entire lifecycle

Automation Runtime characteristics

The characteristics of Automation Runtime result from the challenges faced by the configuration tool and the required runtime system functions. Automation Runtime is fully integrated into B&R target systems. It supports all hardware platforms and makes application creation hardware-independent.



Figure 3: Automation Runtime - The platform for a scalable system

Automation Runtime allows application programs to access I/O systems, interfaces, fieldbuses, networks and storage devices. Comprehensive configuration options for timing as well as client and server applications offer the flexibility required for modern control systems.

Automation Runtime provides a number of important features:

- Runs on all B&R target systems
- Makes the application hardware-independent
- Guarantees deterministic behavior with a cyclic runtime system
- Allows the configuration of 8 different task classes and cycle times
- Guarantees a response to timing violations
- Provides configurable tolerance limits for all task classes
- Offers extensive function libraries in accordance with IEC 61131-3
- Provides access to networks and bus systems
- Contains integrated client and service interfaces
- Offers an OPC UA client and server interface as well as user management
- Provides comprehensive diagnostic functions



Real-time operating system \ Method of operation

Automation Runtime

2.1 Automation Runtime structure

Automation Runtime provides the user with a deterministic, hardware-independent, multitasking tool for creating applications. It manages hardware and software resources and offers complete diagnostics.

The IEC library functions in Automation Runtime make programming faster and easier while helping to avoid errors.

Automation Runtime meets the highest demands for deterministic behavior and speed.

To take advantage of this performance advantage in the application, an abstraction layer is put over the real-time OS. This ensures the user that no adjustments will need to be made to the application if a different operating system is used. The uniform programming interface always remains the same.

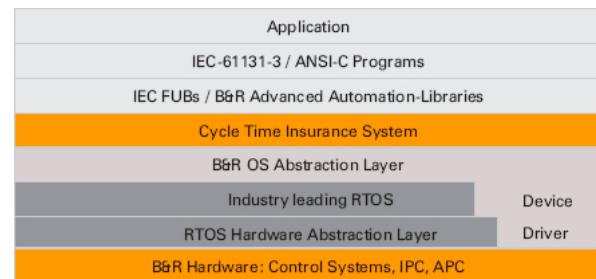


Figure 4: Automation Runtime architecture

Programs, tasks and task class

A program that is assigned to a task class and executed on the target system is also called a task.

Tasks are executed cyclically in a task class.

Cycle time and priority are determined via the task class.

Object Name	Version	Transfer To S
CPU		
└ Cyclic #1 - [10 ms]		
└ mainlogic	1.00.0	UserROM
└ brewing	1.00.0	UserROM
└ heating	1.00.0	UserROM
└ feeder	1.00.0	UserROM
└ conveyor	1.00.0	UserROM
└ Cyclic #2 - [200 ms]		
└ Cyclic #3 - [500 ms]		
└ Cyclic #4 - [1000 ms]		
└ Cyclic #5 - [2000 ms]		
└ Cyclic #6 - [3000 ms]		
└ Cyclic #7 - [4000 ms]		
└ Cyclic #8 - [5000 ms]		

Figure 5: Task class system with cyclic tasks in task class #1



2.1.1 Automation Runtime target systems

Automation Runtime can run on many different hardware platforms. For X20 controllers, Power Panels and Automation PCs, PC-based hardware platforms are used.

Automation Runtime Embedded

Automation Runtime Embedded can be implemented on all target systems where Automation Runtime is run as an independent operating system. The X20 operating system, Power Panels and PC-based hardware platforms are suitable for Automation Runtime Embedded.



Figure 6: Automation Runtime on all types of controllers

Automation Runtime simulation - ARsim

ARsim is a Windows-based Automation Runtime system that isn't real-time capable but essentially corresponds to the functionality of all other target systems. ARsim can be used to fully simulate all controllers, HMI applications and drives.



Figure 7: Automation Runtime simulation on the PC

Automation Studio allows you to switch directly from real hardware configurations to Automation Runtime Simulation. Switching over is done using the Automation Studio toolbar. Automation Runtime Simulation is started automatically and a connection is established. Tests on the PC are made possible quickly and easily due to seamless integration of the simulation.



Figure 8: "Activate Simulation" button



Automation Runtime Windows - ARwin

ARwin is a target system capable of running on all Windows operating systems. A real-time operating system imposes itself over the Windows operating system, which assumes complete control of computer resources. The host operating system itself is handled as a low-priority task on the real-time operating system.



Figure 9: Automation Runtime - real-time applications on Windows

B&R Hypervisor

With B&R Hypervisor, it is possible to run Automation Runtime on a target system parallel to Linux or Windows. B&R Hypervisor makes it possible to cleanly partition a system's hardware resources and assign one or more logical CPUs, memory areas and peripheral devices to different operating systems running on a target system. Each operating system is logically separated from the others at runtime.

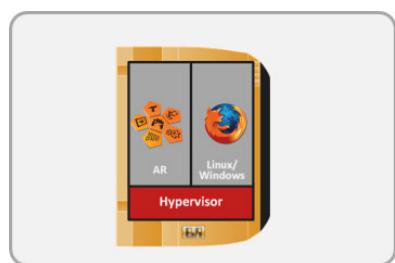
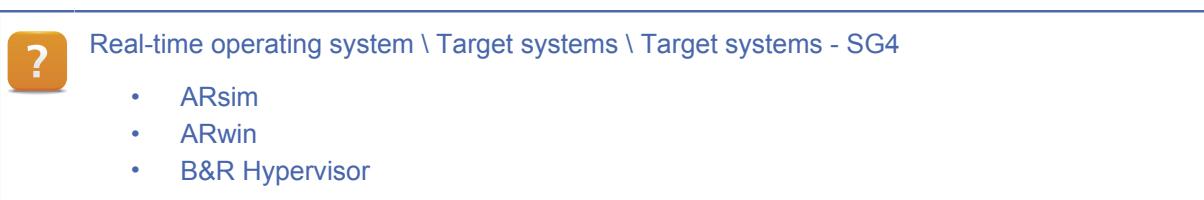


Figure 10: B&R Hypervisor - Multiple operating systems on one Automation PC

Automation Runtime



2.1.2 Server and client functions

Numerous client and server functions are available on Automation Runtime. Different network services and protocols are supported.

The individual protocols are highlighted in color in the image. The protocols that are already enabled for a new Automation Studio configuration are highlighted in green.

Those protocols that can be additionally enabled and configured via configuration entry are highlighted gray.

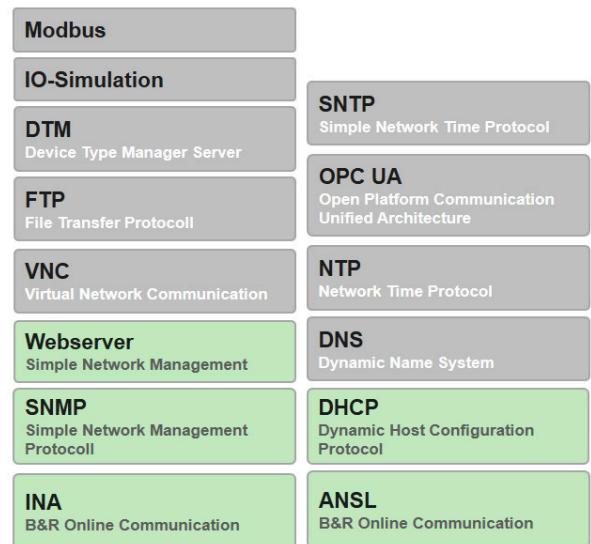
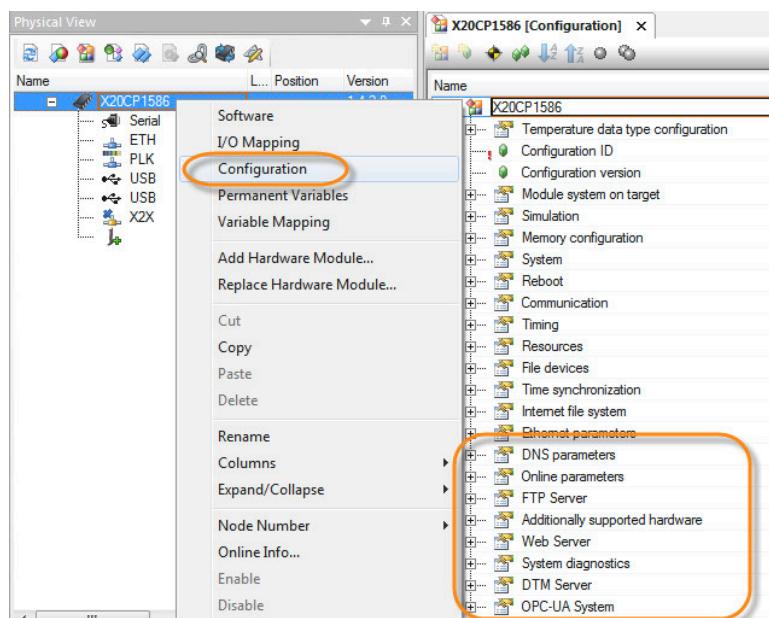


Figure 11: Configurable Automation Runtime server and client functions



Open the configuration entries from the shortcut menu for the CPU and the Ethernet interface. Every protocol has a standalone configuration entry that is used to enable and configure the corresponding protocol.

Many of the functions offered can be diagnosed and configured at runtime using libraries.

Figure 12: Open CPU and Ethernet configuration via the shortcut menu



Programming \ Editors \ Configuration editors \ Hardware configuration \ CPU configuration \ SG4

Communication \ Ethernet \ AR configuration \ Interface configuration (SG4)

Programming \ Libraries

- Configuration, system information and runtime control
- Communication

2.2 Automation Runtime diagnostics capability

Independent of the implemented hardware platform, Automation Runtime supports many diagnostic tools that provide information about the current system state. A list of different diagnostic access options for Automation Runtime is provided below.

The use of the integrated diagnostic tools is documented in Automation Help. Additional executions and exercises can be found in the "TM223 - Automation Studio Diagnostics" training module.



Figure 13: Automation Runtime - complete diagnostics integrated



Diagnostics and service \ Diagnostic tool

- Status bar
- Information about the target system
- System Diagnostics Manager
- Monitor mode \ Online comparison
- Logger
- Profiler

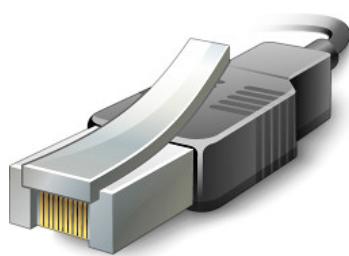
Diagnostics and service \ I/O and network diagnostics

Programming \ Libraries \ Configuration, system information, runtime control

2.3 Connectivity and access & security

The B&R solution is completely open for seamless integration in existing networks and allows direct integration in devices from other manufacturers. B&R components can be implemented as either fieldbus masters or slaves.

An OPC UA server and client enable the publication and exchange of process data. The user roles system helps with the limitation of read and write access. An encrypted connection is enabled using Transport Layer Security (TLS).



Support of devices from other manufacturers

Automation Studio allows fieldbuses and also devices from other manufacturers to be directly integrated. Master and slave devices are added to Automation Studio – like all other B&R components – in the Physical View or in the System Designer.



Communication \ Fieldbus systems

Project management \ Hardware management \ 3rd-party devices / 3rd-party device manager

Access & security - User management system

Automation Studio supports configuration of a user system, a role system and a certificate management system as well as management of SSL/TLS configurations. These configurations are managed in the "Access & Security" package in the Configuration View.



Programming \ Access & Security \

- Transport Layer Security (TLS)
- User role system

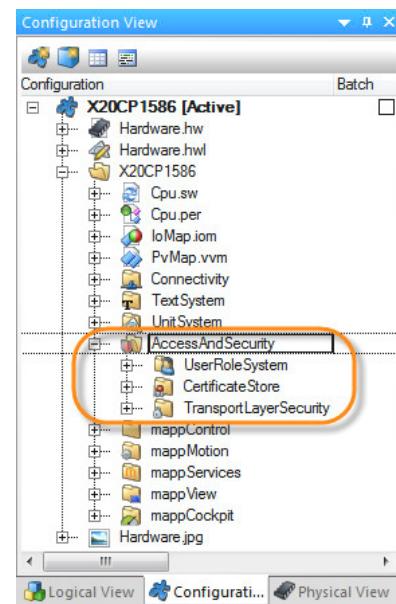


Figure 14: User management system in the Configuration View

OPC UA server and client

Automation Studio supports the configuration of an OPC UA server. Read/write access for the individual OPC UA codes is managed using the user role system. OPC UA client functions are offered via a library.



Communication \ OPC UA

Programming \ Libraries \ Communication \ AsOpcUac

Automation software \ Example projects \ Communication with OPC UA client function blocks

3 Installation and commissioning

Every target system starts Automation Runtime and loads the Automation Studio project from flash memory. Depending on the target system used, the storage device may be a CompactFlash card, CFast card or integrated flash memory.

There are different options for installing Automation Runtime. Automation Runtime can be installed online, offline or via USB.

Regardless of which method is selected, Automation Studio provides support during the installation process.

During initial installation, the flash memory of the controller is partitioned according to the requirements of the application.

When installing Automation Runtime, settings like the I/O configuration and the interface configuration are included.

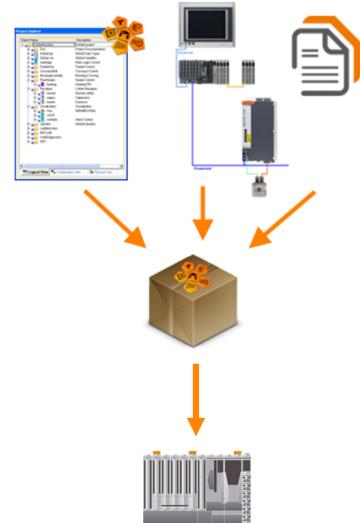


Figure 15: Installation and commissioning



The exception with PC-based target systems is Automation Runtime for Windows (ARwin). In this case, Automation Runtime is installed the first time using a special setup tool.



Hardware \ X20 system \ X20 modules \ CPUs

- X20(c)CP1301, X20CP1381 and X20CP1382 \ Operating and connection elements \ Programming the system flash memory
- X20(c)CP158x and X20(c)CP358x \ Programming the system flash

Project management

- Project installation
- Simulation \ ARsim \ Creating an executable project structure

Real-time operating system \ Target Systems \ Target systems - SG4 \ ARwin \ Installation / Configuration

Installation and commissioning

3.1 Changing and updating Automation Runtime

If new software versions are available for the runtime environment, then these software versions can be added via the **Upgrades** dialog box. The upgrade dialog box is opened by selecting **<Tools> / <Upgrades>** from the menu.

The Automation Runtime versions are changed in the Automation Studio project for the currently active configuration.

Software versions for the runtime environment can be modified by selecting **<Project> / <Change Runtime Versions...>** from the main menu.



Changing the Technology Package version (mapp Motion, mapp Services, etc.) affects all configurations in the project, since the libraries linked with them are managed in the Logical View.

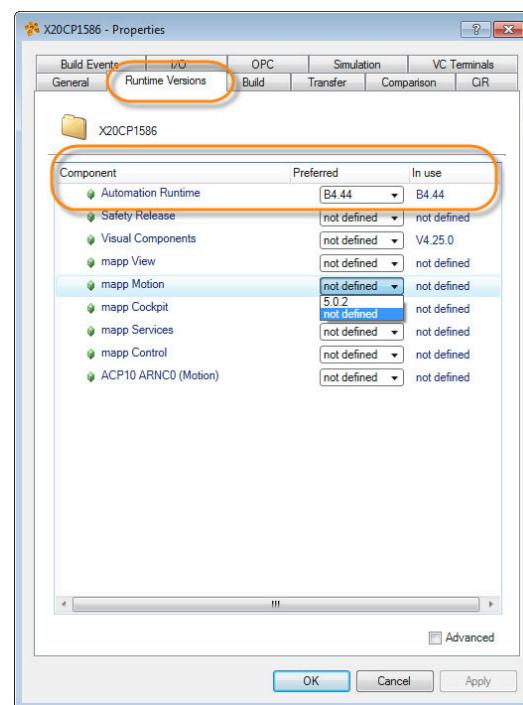


Figure 16: Changing runtime versions



Project management

- Workspace \ Upgrades
- Changing runtime version

Real-time operating system \ Version overview

Upgrading from Automation Studio

If Automation Runtime is already installed on the target system, the project can be transferred along with the new Automation Runtime version. Alternatively, the creation of installation media, such as a CompactFlash, CFast and USB flash drive, is supported. Further information is described in the following section (["Project installation" on page 16](#)).



Figure 17: Display of version differences in the transfer dialog box; restart required

When transferring the project to the target system, any Automation Runtime version conflicts are detected and displayed in a dialog box.

Upgrading without Automation Studio

If it is not possible to update using Automation Studio, the Runtime Utility Center can be used to create a complete image for installation. For example, it can then be transferred to a USB flash drive.



Project management \ Project installation \ Performing project installation \ Export RUC

3.2 Configuration ID and partitioning

Before project installation, the settings for the configuration ID and the partitioning of the flash memory should be checked in the project. It is also possible to define a configuration version. The settings are opened via the shortcut menu for the CPU.

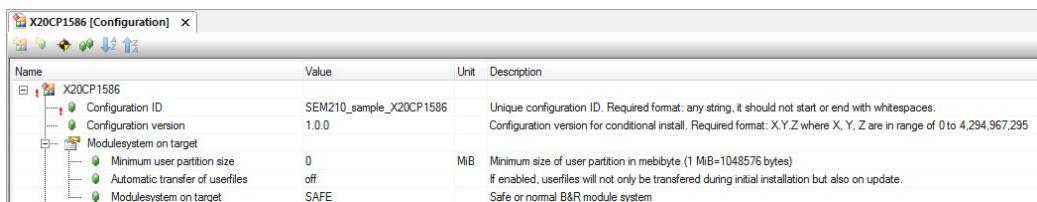


Figure 18: CPU settings: settings for configuration ID and partitioning

Configuration ID

With Automation Runtime 4.25 and later, a unique configuration ID is assigned to each configuration in the Automation Studio project. The configuration ID serves as a unique identifier of the project and is preset to "<AS Project name>_<Configuration name>". It is necessary to assign a unique configuration ID in the project. This allows the system to differentiate between an update transfer (same ID) or initial transfer (different ID) during project installation.



Figure 19: Comparison of the configuration ID in the transfer dialog box

Installation and commissioning

Partitioning

The flash memory of a controller is organized as a file system. Depending on the selected partition option, a distinction is made between the normal and secure file system. For the normal file system, a partition is created in the flash memory, on which Automation Runtime and the application are saved.

For the secure file system on the other hand, Automation Runtime and the application are stored on different partitions. The partition sizes for Automation Runtime and the application are calculated automatically.¹

A user partition can be added to the normal and secure file system. On this partition, the user can save things like recipe data at runtime. The memory size is manually determined for the user partition.



Programming \ Editors \ Configuration editors \ Hardware configuration \ CPU configuration \ SG4

Project management \ Project installation

- Performing project installation \ Settings \ System configuration settings
- FAQ

3.3 Project installation

It is possible to transfer Automation Runtime over an online connection. In order to do this, the online interface must be configured correctly and be in the right mode. (["Starting Automation Runtime" on page 30](#)). Alternatively, the Automation Studio project, including Automation Runtime, can be transferred via offline installation and by creating a project installation package.

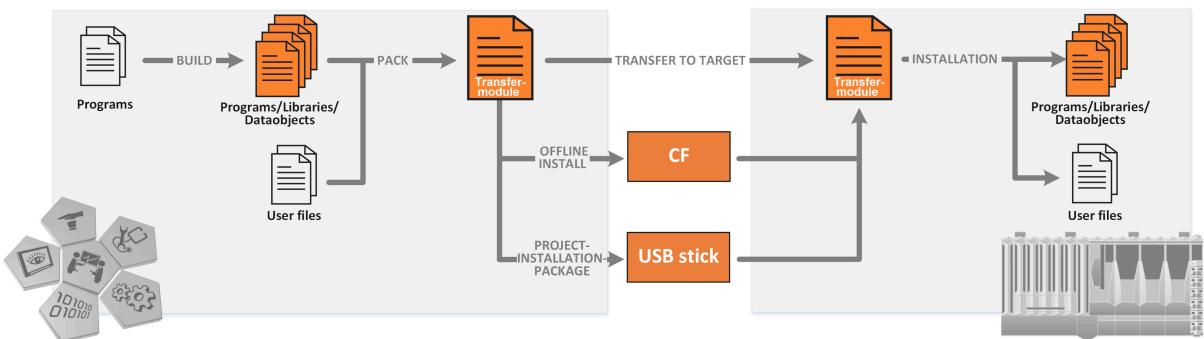


Figure 20: Schematic diagram of project installation options

Establishing a connection

A connection to the controller is established using the "Browse for target system" feature in Automation Studio. This function searches the network for B&R controllers. The connection settings of the controllers can be temporarily changed in the search dialog box.



Programming \ Building and transferring projects \ Establishing a connection to the target system \ Ethernet connections \ Browse for targets

¹ The partition sizes for Data1 and Data2 are set at a ratio of 2:1. This is necessary so that sufficient storage space is available for the transfer module during project installation.

Online installation

Automation Runtime can be transferred or online installation performed once the connection has been established.



Programming \ Building and transferring projects \ Online services \ Transfer Automation Runtime \ Transferring to SGx target systems \ Installing via an online connection

Project management \ Project installation

- Scenarios \ Online commissioning

Offline installation

For offline installation, Automation Studio is used to generate an installation medium (CompactFlash or CFast card). This is then inserted into the target system. When the supply voltage is switched on, Automation Runtime and the application are started.



Project management \ Project installation

- Scenarios \ Offline commissioning

Project installation package - USB installation / Network installation

Automation Runtime and the application software can be included in a project installation package and transferred to the target system on a USB flash drive, CompactFlash card or via a DHCP server.

If a USB flash drive containing a project installation package is inserted into the target system, then the update can be triggered by restarting the system or using the ArProject library.



When installing using a project installation package, it may be necessary to set the device to BOOT mode, depending on the device configuration and operating status. In BOOT mode, the mechanism for installing a project installation package from a USB flash drive or the network is always active.



Project management \ Project installation

- Scenarios \ Offline commissioning
- Overview \ Transfer \ Project installation package

Programming \ Libraries \ Configuration, system information, runtime control \ ArProject

Device-specific installation

A distinction is roughly made between devices with integrated flash memory and devices with CompactFlash. The device-specific Automation Runtime installation, initial startup and a description of the reset button can be found in the respective user's manuals.

Installation and commissioning



All target systems have the option of deliberately starting in BOOT mode. Depending on the target system, this is done using a reset button, the mode selector switch or node number selector switch. The reset button can be used to restart the system and change the operating mode. The mode selected with a mode selector switch is applied after restarting. Further information about the possible operating modes and actions can be found in the data sheet of the respective target system.



Programming \ Build & transfer \ Online services \ Transfer Automation Runtime

Hardware \ Power Panel \ Power Panel C70 \

- Installation \ Commissioning
- Device description \ Operating and connection elements \ Button for reset and operating modes

Hardware \ X20 system \ X20 modules \ CPUs \ X20(c)CP1301, X20CP1381 and X20CP1382 \ Operating and connection elements \

- Programming the system flash memory
- Button for reset and operating mode

Exercise: Check and update the Automation Runtime version

The goal of this exercise is to explore the options for updating Automation Runtime on the target system. Automation Runtime upgrades are searched for, the settings of the Runtime versions are checked in the project, Configuration ID and partitioning are determined and the appropriate mechanism for updating the target system is selected.

1) Search for Automation Runtime upgrades

<Tools> / <Upgrades> menu

2) Check and adjust Runtime versions

<Project> / <Change Runtime Versions...> menu

3) Specify configuration ID and partitioning

CPU configuration (configuration ID, module system on the target system)

4) Select the appropriate online/offline installation or project installation package for the target system

Notes on this are offered in the data sheet under section "Programming the system flash".

5) Perform installation

For an online installation: Force initial transfer

4 Memory management

Memory on an Automation Runtime target system is divided into RAM and ROM.

Parts of these areas are used exclusively by Automation Runtime at runtime; the rest is available for the application.

4.1 Logical partitioning of the Flash memory

During the build process for Automation Runtime, an Automation Studio project generates modules that can be executed. These are transferred to the flash memory during project installation. CompactFlash, CFast and integrated flash memory are used as storage mediums.

A target memory is automatically assigned to every module in the software configuration. Modules that belong to Automation Runtime are transferred to the system ROM. Modules that belong to the application are transferred to the user ROM. This is a purely logical subdivision. The data transferred is saved on the same data storage device.

Object Name	Version	Transfer To	Size (bytes)	Source
CPU				
Cyclic #1 - [10 ms]				
Cyclic #2 - [20 ms]				
Cyclic #3 - [50 ms]				
Cyclic #4 - [100 ms]				
LampTest	1.00.0	UserROM	328	LampTest
Cyclic #5 - [200 ms]				
Cyclic #6 - [500 ms]				
Cyclic #7 - [1000 ms]				
Cyclic #8 - [10 ms]				
Data Objects				
Nc Data Objects				
Visualisation				
Binary Objects				
Library Objects				
runtime	3.08.0	UserROM	34832	
Source Objects				
Configuration Objects				
lomap	1.00.0	UserROM	6276	
sysconf	3.08.0	SystemROM	58948	
ashwd	1.00.0	SystemROM	1388	
arcconfig	1.00.0	SystemROM	1960	
asfw	1.00.0	SystemROM	192336	

Figure 21: Target memory for software objects in the software configuration

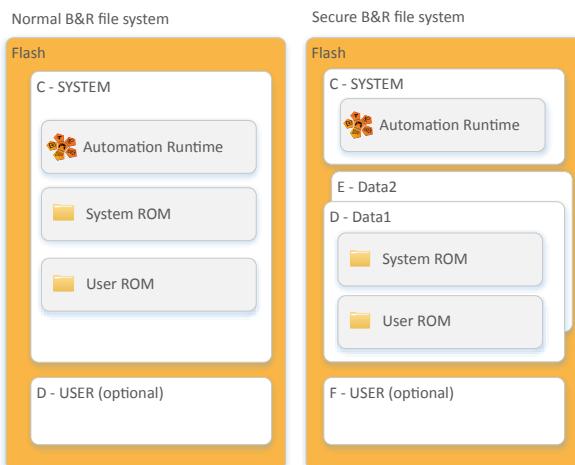


Figure 22: Comparison between a normal and secure B&R file system

For the normal B&R file system, modules from the system ROM and user ROM are stored in different folders on the C partition.

For the secure B&R file system, Automation Runtime is stored on the "C" partition. The modules in system ROM and user ROM are stored on the D partitions and a copy of this data is stored on partition E.

System modules are separated from the application by this mechanism.

It is possible to configure an additional partition for managing files generated at runtime. During project installation, data is transferred to the user partition using the transfer settings.

Memory management

See: "Settings when transferring programs" on page 46

The settings for the normal or secure B&R file system are defined in the CPU configuration.

Name	Value	Unit	Description
X20CP1586			
Configuration ID	SEM210_sample_X20CP1586		Unique configuration ID. Required format: any string, it should not start or end with whitespaces.
Configuration version	1.0.0		Configuration version for conditional install. Required format: X.Y.Z where X, Y, Z are in range of 0 to 4,294,967,295
Modulesystem on target			
Minimum user partition size	0	MiB	Minimum size of user partition in mebibyte (1 MiB=1048576 bytes)
Automatic transfer of userfiles	off		If enabled, userfiles will not only be transferred during initial installation but also on update.
Modulesystem on target	SAFE		Safe or normal B&R module system

Figure 23: Partitioning configuration for the flash memory in the CPU configuration



Real-time operating system \ Method of operation \ Memory \ Memory types

Project management \ Configuration View \ Software configuration

Project management \ Project installation \ Performing project installation \ Settings

- System configuration settings
- Installation settings

4.2 Used RAM

DRAM is used as a fast read/write memory for the execution of Automation Runtime and the application.

Alternatively, the target systems have battery-backed RAM memory that is used, for example, to retain data after a restart.



The memory types and memory sizes used in the target system are documented in the data sheet of the selected CPU.



Figure 24: DRAM, SRAM and FRAM

DRAM

During startup, Automation Runtime, all configurations and the tasks are copied into DRAM and executed there. This is necessary because access to the DRAM is quicker than flash memory.

In DRAM, a task also requires the configured local or global variable memory. Initialization of this memory is carried out automatically.

SRAM and FRAM

SRAM (static RAM) is buffered by a battery. This allows data to be retained even after a power failure. Of course, the backup battery must be functional.

Newer controller models no longer require a backup battery. FRAM saves data on a nonvolatile electronic memory type, as opposed to SRAM and DRAM.

Real-time clock

The real-time clock is backed up in different ways depending on the device. The duration that it is protected as well as the clock drift in relation to the ambient temperature are described in the respective data sheet of the CPU being used.

See also:

- ["Initializing variable memory" on page 27](#)
- ["Restarting and power failures" on page 33](#)
- ["Using retain variables" on page 34](#)



Information about the service interval of the backup battery and instructions regarding changing it are documented in the data sheet of the respective CPU.



Real-time operating system \ Method of operation \ Memory \ Memory types

4.3 Tools for determining memory information

Automation Studio unites many diagnostic tools that, among other things, provide information about the system state. Diagnostic tools for calculating memory information will be introduced below.

Online info

Selecting <Online> / <Info> from the main menu shows general information about the amount of available memory. This option is not available for all target systems. In the online info, you can also read and set the status of the backup battery as well as the current time and date on the controller. The change of time and date are logged in the "System" Logger module.

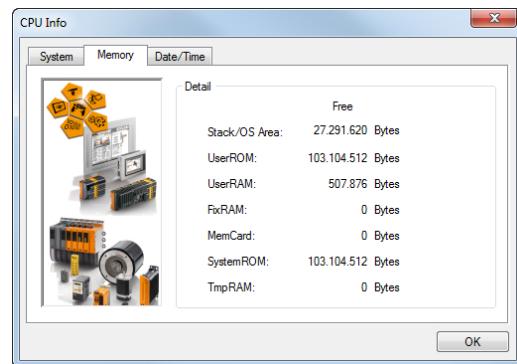


Figure 25: Reading information from the CPU

Memory management

System Diagnostics Manager (SDM)

System Diagnostics Manager (SDM) is an integral component of Automation Runtime. Selecting <Tools> / <System Diagnostics Manager> from the main menu opens SDM in a browser window.

Information about memory usage is shown in the category "System \ Memory". In the "Software" category, you can find information about the modules that have been loaded on the target system.



Figure 26: Memory allocation being shown in SDM

Online software comparison

With the online software comparison, the modules configured in Automation Studio are compared with the modules installed on the target system.

This online software comparison is launched by selecting <Online> / <Comparison> / <Software> from the main menu.

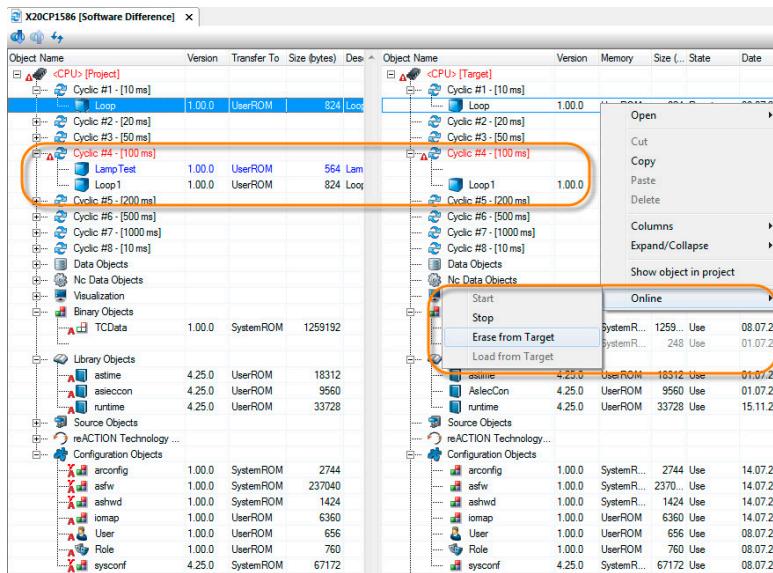


Figure 27: Opening the online software comparison window

The software comparison window displays all of the modules in the project as well as on the controller. The "Size (bytes)" column displays the target memory of the module on the target system. It is possible to compare the version, size, target memory and build date of modules.

Options are displayed in the shortcut menu for a module. Modules can be deleted or stopped individually.²

Library functions

Using function block MEMxInfo() from library AsHW, the free memory on the target system can be calculated in the application.

For example, you get direct access to the file system on the flash memory with the FileIO and MpFile libraries.

² Starting with Automation Runtime B4.33, modules installed with a project installation or project update can only be deleted with a project installation or project update. Deleting modules on the target system is only possible for data objects generated at runtime.



Diagnostics and service \ Diagnostic tool

- Information about the target system
- System Diagnostics Manager
- Monitor mode \ Online software comparison

Programming \ Libraries

- Configuration, system information and runtime control \ AsHW
- Data access and data storage \ FileIO

Services \ mapp Services \ MpFile: File management system

Exercise: Set the time and date

It's necessary to set the time and date properly on the controller in order to interpret system events correctly. This can be done in the online info dialog box. The change is logged in the "System" Logger module.

- 1) Change the time and date with <Online> / <Info>.
- 2) Check result in the "System" Logger module

The Logger is opened by selecting <Open> / <Logger>.

In the "System" Logger module, the entry "WARNING: Time / Date changed" is now displayed.

Logger Entries: 13 (Errors: 2, Warnings: 7, Informations: 2, Success: 2)						
Object Name	Visible	Continuous	Severity	Time	ID	Area
Online	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1 Warning	2016-07-14 15:18:35.000000	2075	WARNING: Time / date changed
System	<input checked="" type="checkbox"/>	<input type="checkbox"/>				clock set: 0 [sec]

Figure 28: The "System" Logger module displays the change of time and date.

Exercise: Check the amount of memory available on the target system

The project, which was created while working with the training module "TM210 – Working with Automation Studio", is already installed on the target system.

Determine the memory allocation on the target system using one of the methods described previously. It should be determined what memory types exist on the used target system and how they are used. You must also check which modules have been loaded on the target system.

- 1) Calculate free DRAM memory with the online info
- 2) Calculate free flash memory with the System Diagnostics Manager
- 3) Use System Diagnostics Manager to check which programs have been loaded on the target system
- 4) Use the online software comparison to check which programs have been loaded on the target system

Memory management

4.4 Global and local variables

Variables are symbolic programming elements whose structure and size is determined by their data type. During the build, a memory location is assigned to the variables by the compiler.

A variable's scope and properties determine its behavior during startup and runtime.



Programming \ Variables and data types

4.4.1 Local variables

Local variables are defined in the scope of a program. Direct access to these variables from other programs is not possible.

A local variable is managed in a .var file at the same level as the program.

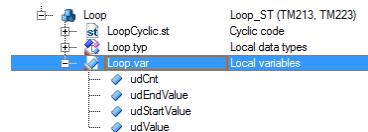


Figure 29: Local variables in the "Loop" program

If local variables from a program must be transferred to local variables of another program, this is carried out using a variable assignment file in the Configuration View.

Exercise: Create a "Loop" program using local variables

A new Structured Text program named "Loop" is created.

Declare four variables named "udCnt", "udValue", "udStartValue" and "udEndValue" of data type UDINT.

Create a loop in the cyclic section of the program. This loop will be explained and used in later exercises.

- 1) Add a new ST program with the name "Loop".
- 2) Open the "Loop.var" file and add the variables.
- 3) After saving the "Loop.var" file, the variables in the program "LoopCyclic.st" can be used.

```
PROGRAM _CYCLIC

    FOR udCnt := udStartValue TO udEndValue DO
        udValue := udValue + 1;
    END_FOR

END_PROGRAM
```

Loop:Loop.var [Variable Declaration]	
Name	Type
* Local variables of program Loop	
udCnt	UDINT
udStartValue	UDINT
udEndValue	UDINT
udValue	UDINT

Figure 30: Variable declaration for program "loop"

Exercise: Multiple assignment of a program in the software configuration

After completing the last task, move the same program twice from the Logical View to the software configuration using drag-and-drop.

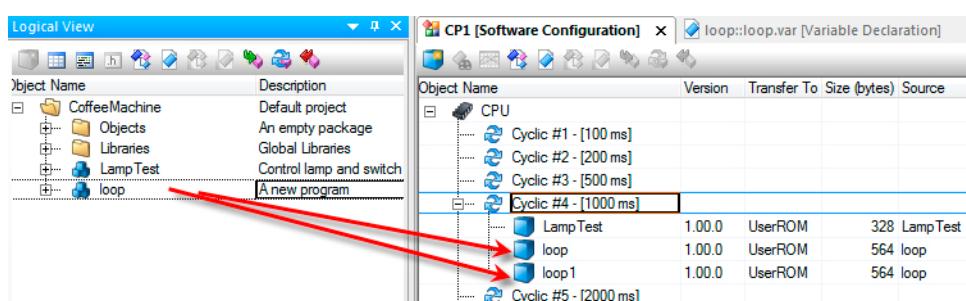


Figure 31: Multiple assignment of programs

- 1) Open the software configuration
- 2) Switch to the Logical View in the Project Explorer
- 3) Move the "Loop" program to the software configuration using drag-and-drop



The Watch window for both instances of the "Loop" program can be opened by selecting it from the shortcut menu in the software configuration or in the Logical View. A selection dialog box appears when the Watch window is opened via the Logical View with regard to which instance the Watch window should be opened.

The Watch window can also be opened by pressing the key combination <CTRL + W>.



All variables are added in the Watch window for tasks "Loop" and "Loop1". Both tasks only have local variables. Changing the variables in the Watch monitor has no effect on the other respective instance of the "Loop" program.

The screenshot shows two side-by-side 'Watch' windows. The left window is titled 'Loop1::pvm [Watch]' and the right is 'Loop::pvm [Watch]'. Both windows show a table of variables:

Name	Type	Scope	Value
udCnt	UDINT	local	421
udEndValue	UDINT	local	420
udStartValue	UDINT	local	0
udValue	UDINT	local	24632
udCnt	UDINT	local	391
udEndValue	UDINT	local	390
udStartValue	UDINT	local	0
udValue	UDINT	local	142664

Figure 32: Variable monitor for the "loop" and "loop1" tasks



Variables that are included in the variable file but not used in the program are not available on the target system.

A corresponding warning is output during the build process.

Warning 424: Variable <variable name> is declared but not being used in the current configuration.



Programming \ Variables and data types \ Scope of declarations

Diagnostics and service \ Diagnostics tools \ Watch (variable monitor)

Memory management

4.4.2 Global / Package-global variables

Global variables are displayed at the top level of the Logical View. They can be used throughout the entire Automation Studio project. They can be used in every program.

A global variable is managed at the top level in the Global.var file. Additional .var files can also be created to provide a better structure for the project.

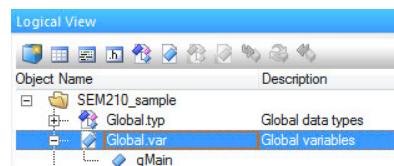


Figure 33: Global variables in the Logical View

Variables that were declared within a package are package-global variables. These are only visible within the respective package and all subordinate packages.



Global variables should only be used if it is necessary to exchange data between several programs. Another alternative is to connect local variables in different programs together using variable mapping.

Exercise: Create a global variable named "gMain" and use it in the "Loop" program

Enter a new variable in the global variable declaration "Global.var" with the name "gMain" and data type UDINT.

This variable should be incremented cyclically in the "Loop" program.

```
gMain := gMain + 1;
```

- 1) Open the "Global.var" file.
- 2) Add the "gMain" variable with data type UDINT.
- 3) Once the "Global.var" file has been saved, the variable in "LoopCyclic.st" can be used in the program.

If the variable "gMain" is added in the Watch window for tasks "Loop" and "Loop1", the value change in both tasks is visible.

Name	Type	Scope	Value
udCnt	UDINT	local	391
udEndValue	UDINT	local	390
udStartValue	UDINT	local	0
udValue	UDINT	local	1693370
gMain	UDINT	global	0

Name	Type	Scope	Value
udCnt	UDINT	local	421
udEndValue	UDINT	local	420
udStartValue	UDINT	local	0
udValue	UDINT	local	191769
gMain	UDINT	global	5973

Figure 34: Writing to global variables

Programming \ Variables and data types \ Scope of declarations
 Programming \ Editors \ Configuration editors \ Variable mapping editor

4.4.3 Initializing variable memory

During startup, all variables which haven't been given an initial value are automatically initialized with the value "0". An initialization value can be specified in the variable declaration in the "value" column.

Name	Type	& Reference	Constant	Retain	Replicable	Value
udCnt	UDINT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	0
udStartValue	UDINT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	1000
udEndValue	UDINT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	0
udValue	UDINT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	0

Figure 35: Variable initialization in "Loop.var"

Initialization via the variable declaration replaces, for example, the following program lines in the initialization subroutine "LoopInit.st":

```
PROGRAM _INIT
  udEndValue := 1000;
END_PROGRAM
```

Exercise: Initialize the "udEndValue" variable

Configure variable "udEndValue" in the "Loop" program with an initial value of 1000. Monitor the value in the Watch window.

Memory management

- 1) Open the "**Loop.var**" variable declaration.
- 2) Set the value for the "**udEndValue**" variable in the "**Value**" column.



In the variable monitor for the "**Loop**" and "**Loop1**" tasks, variable "**udEndValue**" is initialized with the value 1000. During runtime, this value can be changed to any other value.



Programming \ Editors \ Table editors - General \ Declaration editors \ Table editor for variable declaration

4.4.4 Using constants

Constants are variables whose values never change while the program is running. They are implemented in the programming instead of fixed numerical values. This makes programs easier to read and maintain.

Name	Type	Constant	Retain	Replicable	Value
*COPYRIGHT - Bemecker + Rainer					
gMain	UDINT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	100

Figure 36: Declaring constants

Exercise: Global variable "gMain" as a constant

Configure the global variable "**gMain**" as a constant with a value of 100.

- 1) Open the "**Global.var**" variable declaration.
- 2) Set the value in the "**Value**" column for the "**gMain**" variable.
- 3) Define the "**gMain**" variable as a constant.



An error message appears during the build, because there is write access to "**gMain**".

Write access to constants in programs is not permitted. In order for the program to operate without error, the "**gMain**" variable must not be defined as a constant.

LoopCyclic.st (Ln: 19, Col: 8) : Error 1138: Cannot write to variable 'gMain'.

It makes more sense to initialize the variable "**udStartValue**" as a constant with the value "0". This variable is defined as read-only in the program.



The cross-reference list is used to determine read and write access of variables in programs. A cross-reference list is generated by selecting **<Project> / <Create cross-reference>** from the main menu.



Programming \ Variables and data types \ Variables \ Constants
Project management \ Workspace \ Output window \ Cross reference

Runtime performance

5 Runtime performance

This section describes the runtime behavior of the application on the target system. Startup behavior, program initialization as well as the cyclic program process are explained.

5.1 Starting Automation Runtime

Automation Runtime boots when the target system is switched on. The following tasks are executed before execution of the cyclic programs.

- Hardware check
- Check if hardware firmware upgrade is required
- Checking the BR modules
- BR modules copied from ROM to DRAM
- Retain variables copied to DRAM
- Variable memory set to initialization value
- Execution of initialization subroutine
- Activation of cyclic programs

If no errors occur during the boot phase, the target system starts in RUN mode.

ANSI: tcpip/RT=1000 /SDT=5 /DAIP=10.43.15.34 /REPO=11159 /ANSL=1 /PT=1... | 4PPC70.0573-20B 14.25

RUN

Figure 37: Power Panel C70 in RUN mode



Real-time operating system \ Method of operation \ Boot behavior

Real-time operating system \ Method of operation \ Module / data security

5.1.1 Automation Runtime operating states

The boot process of Automation Runtime can be divided into the following four operating states. Once an operating state has been reached successfully, the check for changing to the next operating state occurs.

In the "RUN" operating state, Automation Runtime starts the application created with Automation Studio and executes it cyclically.

Certain events lead to the target system canceling startup and staying in the corresponding operating state for diagnostic purposes. The current operating state is read using the LED status indicators on the CPU and the status bar in Automation Studio. Details are logged in the "System" Logger module.

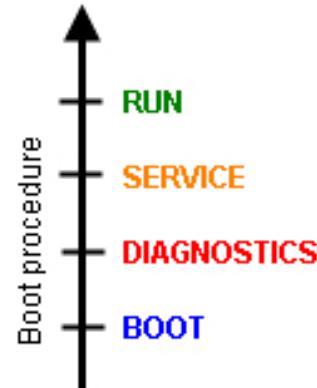


Figure 38: Operating states of Automation Runtime

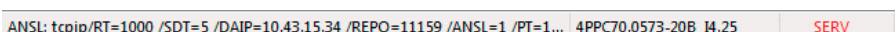
Operating state	Conditions that lead to this system state
BOOT	<ul style="list-style-type: none"> • No CompactFlash inserted • No operating system available on the CompactFlash/CFast card³ • Node number switch to "00", mode selector switch to "BOOT", reset button ⁴ 
	Figure 39: Status bar - BOOT mode
DIAGNOSTICS	<ul style="list-style-type: none"> • Fatal system error • Node number switch to "FF", mode selector switch to "DIAG", reset button 
	Figure 40: Status bar - DIAGNOSTICS mode
SERVICE	<ul style="list-style-type: none"> • Division by zero⁵ • PageFault • Cycle time violation • Missing hardware modules • CPU halted by Automation Studio 
	Figure 41: Status bar - SERVICE mode
RUN	<ul style="list-style-type: none"> • No error 
	Figure 42: Status bar - RUN mode

Table 1: Overview of Automation Runtime operating states



Real-time operating system \ Method of operation \ Operating status

Hardware \ X20 system \ X20 modules \ CPUs

- X20(c)CP158x and X20(c)CP358x \ Operating mode switch
- X20CP1381-RT and X20CP1382-RT \ Operating and connection elements \ Button for reset and operating mode

³ A requirement for this is that the implemented system has Automation Runtime by default. PC-based systems don't have Automation Runtime by default. In this case, a connection to the target system can't be established. Offline installation must be performed.

⁴ Depending on the device used, the node selector switches, the mode selector switch or the reset button are used for setting the operating mode. The description can be found in the respective user's manual.

⁵ Unlike Intel target systems, division by 0 does not result in a processor exception on ARM target systems, and instead results in 0.

Runtime performance

5.1.2 Automation Runtime boot phases

During startup of the controller, the intermediate steps will be followed. These states are also indicated in the Automation Studio status bar. These phases are usually very short.

System state	Description
STARTUP	Initialization procedures relevant to the operating system are performed during this phase.
FIRMWARE	Firmware is updated during this phase. See: "Hardware versions and firmware updates" on page 32
INIT	The application's initialization subroutines are executed during this phase.

Table 2: Overview of Automation Runtime boot phases



The phases before the RUN state (STARTUP, FIRMWARE, INIT) are indicated by a blinking green R/E LED on the X20 system.



Real-time operating system \ Method of operation \ Boot phases

Hardware \ X20 system \ X20 modules \ CPUs \ X20(c)CP158x and X20(c)CP358x \ X20 CPUs - Status LEDs

5.1.3 Hardware versions and firmware updates

In Automation Studio, each hardware module has its own device description file, a suitable image for System Designer and firmware (optional). When Automation Runtime starts up, the firmware on the hardware module is compared with the firmware installed on the system and, if necessary, updated.

Automation Studio is provided with one hardware version per hardware module. Additional hardware versions can be installed using menu option <Tools> / <Upgrades>.

In the System Designer, you can select from the installed hardware versions in the "Version" column. When installing new hardware versions, they are not automatically updated and must be selected manually.

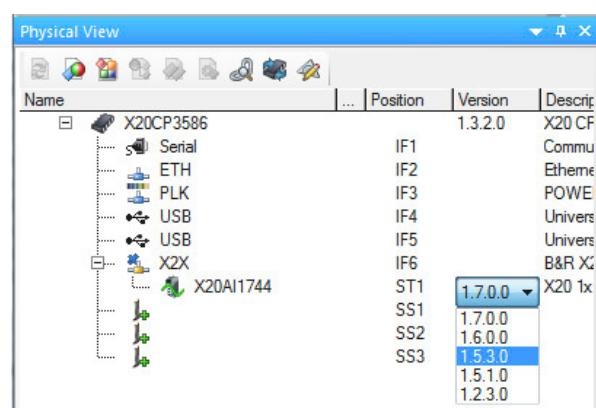


Figure 43: Selecting a hardware version in the Physical View



Project management \ Hardware management \ Hardware versions

5.1.4 Restarting and power failures

The restart behavior of target systems when changing operating states and after reset is documented in sections "Operating mode and node number switches", "Operating mode switch" and "Reset" in the respective data sheet for the controller used.

In addition, the restart behavior is configured in the CPU properties after a reset and power failure.

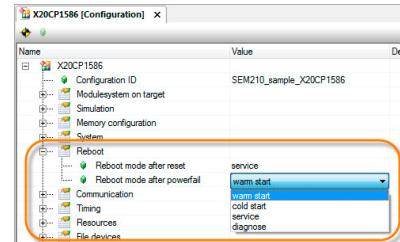


Figure 44: Configuring the restart behavior in the CPU settings

Warm restart

A restart (warm restart) is triggered by the following actions:

- PowerON after power failure
- Pressing the reset button
- Changing a system configuration and transferring it to the target system
- Performing a warm restart with Automation Studio
- Performing a warm restart with SYSreset() function



The restart behavior after a reset or power failure is configured in Automation Studio.

Retain variables keep their values after a warm restart.

Cold restart

A cold restart is triggered by the following actions:

- Restarting after exchanging a CompactFlash card
See offline installation and initial transfer description
- Pressing the reset button
- Performing a cold restart with Automation Studio
- Performing a cold restart with SYSreset() function
- Restarting if the backup battery for the SRAM is defective



The restart behavior after a reset or power failure is configured in Automation Studio.

During a cold restart, retain variables with the value "0" or the initial value from the variable declaration are reinstalled. Permanent variables retain their original value.



Programming \ Libraries \ Configuration, system information, runtime control \ SYS_Lib \ Function blocks and functions \ System functions

Power-on handling

SYSROM and USERROM are located on the CompactFlash, CFast card or internal flash memory when power is not on. The remanent variables (RETAIN) and permanent variables and the USERRAM are located on the battery-backed RAM. Automation Runtime is started when the controller is booted. All the necessary configurations and application programs are copied to DRAM. Remanent and permanent variables are copied from battery-backed RAM to DRAM.



Real-time operating system \ Method of operation \ Module / Data security \ Power-on handling

Power-off handling

Many B&R target systems are equipped with a power failure logic. This allows the system to enter a defined state in the event of a power failure. The following tasks are performed in event of a power failure:

- Access to data objects located in USERRAM is locked.
- The remanent variables (RETAIN) are copied into the battery-backed SRAM.



Real-time operating system \ Method of operation \ Module / Data security \ Power-off handling

Exercise: Startup and operating states

First check the configuration for the restart behavior of the implemented target system. Different operating states should be entered using the mode selector switch, reset button and via Automation Studio. In the data sheet of the target system used, the functions of mode selector switches and reset buttons are documented. Check the "System" logger module after restarting in a certain operating state.

5.1.5 Using retain variables

All boot phases are run through during startup of the target system. In order for the values of process variables to be retained after a warm restart, they must be stored in the remanent memory. During startup, the data is automatically restored.

Retain variables

In order to store variables in nonvolatile (remanent) memory, the following requirements must be met on the target system and in the variable declaration:

- Target system with battery-backed RAM - see data sheet
- Configure the variable by enabling the "Retain" option

Name	Type	& Reference	Constant	Retain	Value
OperatingHours	UINT	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	0
ProductCounter	UDINT	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	0

Figure 45: Configure retain variables in the variable declaration



Different target systems have varying amounts of remanent memory available. Information about size and availability can be found in the respective data sheet or in Automation Help.

Examples of the use of retain variables:

- Operating hours counters
- Type counters and data for system efficiency
- Condition and status of the machine at the time of a power failure
- And much more...



Alternatively, data that is only read or written once when the program is started or when a change is made can be stored on the flash memory.

The function blocks in the MpRecipe library from the mapp Services Technology Package are ideal for this purpose.



Real-time operating system \ Method of operation \ Memory

Programming \ Editors \ Table editors - General \ Declaration editors \ Table editor for variable declaration

Hardware \ X20 system \ X20 modules \ CPUs

- X20(c)CP158x and X20(c)CP358x \ Technical data
- X20(c)CP1301, X20CP1381 and X20CP1382 \ Technical data

Services \ mapp Services \ MpRecipe: Recipe management

Permanent variables

In order for retain variables to be retained after a cold restart, they must be added to the permanent variables in the Configuration View (CPU.per).



Only **global** retain variables can be configured as permanent variables.

Examples of data to be stored permanently:

- Operating hours counters
- All data that isn't written to a flash memory but must also be retained after a cold restart.



It is the responsibility of the user to manage the values of variables in the permanent memory.

If the state of permanent variables are not based on a proper foundation, this can result in undesired program behavior. This is especially important to remember when replacing the CPU or backup battery.



Data storage on a Technology Guard can be used as an alternative to permanent variables, which are always connected to a certain controller. Access to the data storage of a Technology Guard can be gained via the AsGuard library.



Programming \ Variables and data types \ Variables \ Variable remanence

Programming \ Editors \ Configuration editors \ Permanent variables

Automation software \ Technology Guarding

Programming \ Libraries \ Configuration, system information, runtime control \ AsGuard

Exercise: Use RETAIN variables

The "udValue" variable is increased with every loop in the "Loop" program. During a warm restart, the last value of the "udValue" variable must be retained.

- 1) Declare "udValue" variable as RETAIN
- 2) Transfer program - Check value "udValue" in the Watch window
- 3) Perform warm restart
- 4) Check the result

5.2 Program initialization

An initialization subroutine can be managed for every program. For example, variables can be initialized through calculations in an initialization subroutine.

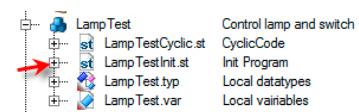
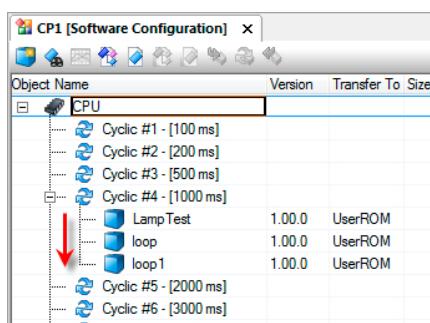


Figure 46: Program initialization

Running the initialization subroutine

Before the first cyclic program is started, all initialization subroutines are executed once in the configured order. As long as the initialization subroutines are executed, this is indicated in the status bar ("INIT") as well as by the LED status indicators on the CPU.



In this example, the initialization subroutines would be executed in the following order:

- 1) Initialization subroutine for the "LampTest" task
- 2) Initialization subroutine for the "Loop" task
- 3) Initialization subroutine for the "Loop1" task

Because initialization subroutines are not subject to cycle time monitoring, a violation will not be triggered by longer initializations.

Figure 47: Executing initialization subroutines

Function calls in an initialization subroutine

When using function blocks, it is important that the function returns a result the first time it is called.



Function blocks that must be called multiple times must be called in the cyclic section of the program. Information about the use of function blocks in the init subroutine can be found in the respective documentation.



Real-time operating system \ Target systems \ Target systems - SG4 \ Runtime behavior - SG4 \ Starting Init SPs - SG4

5.3 Executing cyclical programs

A program is assigned a certain execution time, or task class, in the software configuration.

The programs assigned to the software configuration are identified as tasks. The programs that are assigned in the software configuration are only executed after the transfer.

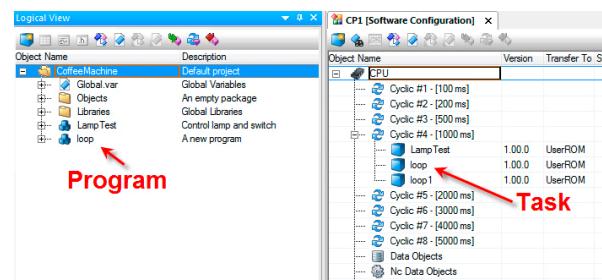


Figure 48: Programs and tasks

The tasks are assigned to configurable task classes and executed one after the other. The execution time of the tasks may vary. However, the times when the tasks start as well as when the I/O system is accessed are deterministic.

5.3.1 Task classes and cycle times

A task is executed cyclically in the time defined by its task class i.e. its cycle time.

Up to eight configurable task classes are provided to help optimize a task for its particular requirements. Every task class contains tasks with the same cycle time, priority and tolerance.

In this example, the task class #4 contains three tasks. A cycle time of 100 ms is configured.



Not all tasks need to run within the same task class. Control tasks that must be executed quickly should be assigned to a task class with a smaller cycle time. For slower processes, a task class with a correspondingly higher cycle time is selected.

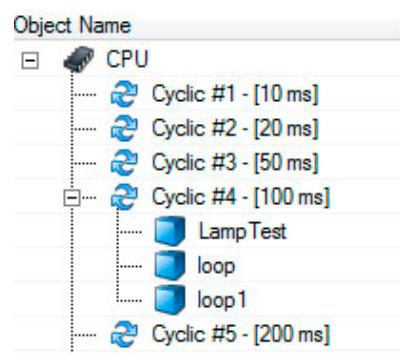


Figure 49: Task class #4 with three tasks

Ignoring the time it takes to execute the tasks, the program sequence would look like this:

Runtime performance

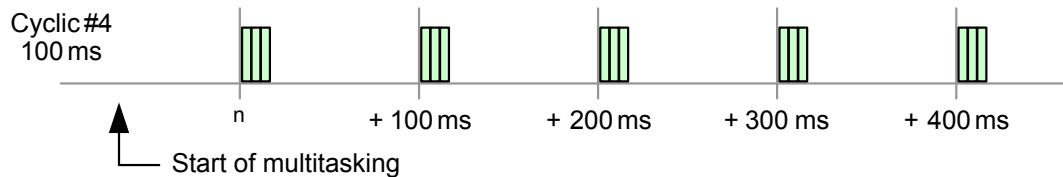


Figure 50: Tasks are called again during each cycle.

This means that the tasks in this task class are executed every 100 milliseconds. In order to do that, the sum of the execution times of the assigned programs can't exceed the configured cycle time.



Real-time operating system \ Target systems \ Target systems - SG4 \ Runtime behavior - SG4
 \ Start of cyclic tasks

5.3.2 Cycle time and priority

The priority of a task class is determined by the number of the task class.

The lower the number, the higher the priority of the task class.

Moving a task between task classes changes its priority and cycle time.

Task class #8 has the lowest priority in the system. (see: "[Task class with high tolerance](#)" on page 45)

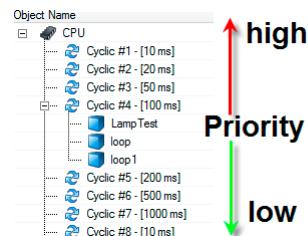


Figure 51: Task class priority

If the "Loop" task is moved from task class #4 to task class #1, it will be executed every 10 milliseconds.



The execution time of a task is not influenced by moving it to a different task class. It just changes the number of times it is called in a given period of time.

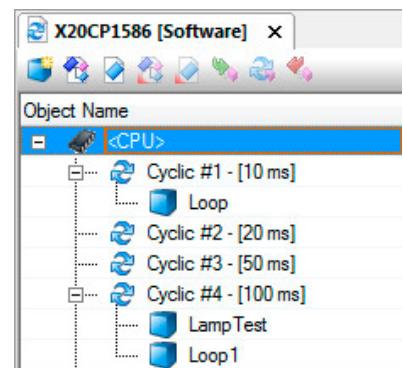


Figure 52: Different task classes

The "Loop" task is now executed every 10 ms. The tasks "LampTest" and "Loop1" in task class #4 are executed every 100 milliseconds, as before.

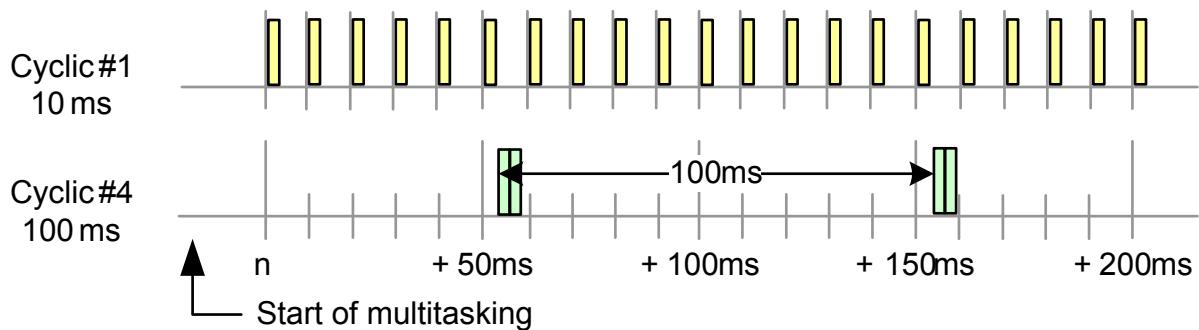


Figure 53: Executing tasks in different task classes

Exercise: Move the "Loop" task to task class #1

The "Loop" task in the software configuration should be moved to task class #1.

In the Watch window, observe the changes to the cycle times of "Loop" and "Loop1", which reference the same program.

- 1) Open the software configuration
- 2) Move the "Loop" task from task class #4 to task class #1
- 3) Open the Watch window for both tasks.
- 4) Monitor the "udValue" variable.



The "Loop" program is executed 10 times as frequently as "Loop1". As a result, the value of the "udValue" variable is increased more often by this factor.

Name	Type	Scope	Value
udCnt	UDINT	local	2
udEndValue	UDINT	local	1
udStartValue	UDINT	local	0
udValue	UDINT	local	288
gMain	UDINT	global	158

Name	Type	Scope	Value
udCnt	UDINT	local	2
udEndValue	UDINT	local	1
udStartValue	UDINT	local	0
udValue	UDINT	local	28
gMain	UDINT	global	158

Figure 54: Cycle times in different task classes

5.3.3 Idle time

During operation, Automation Runtime performs other system tasks in addition to the execution of tasks. They provide the user with useful functions. For example, a constant checksum verification process is performed and online communication is available.

The speed at which these tasks are executed can vary depending on processor performance.

The time when Automation Runtime is not executing any cyclic tasks is referred to as **idle time**. Automation Runtime uses this idle time, for example, for the following tasks:

Runtime performance

- Online communication
- Visual Components application
- Access to the file system

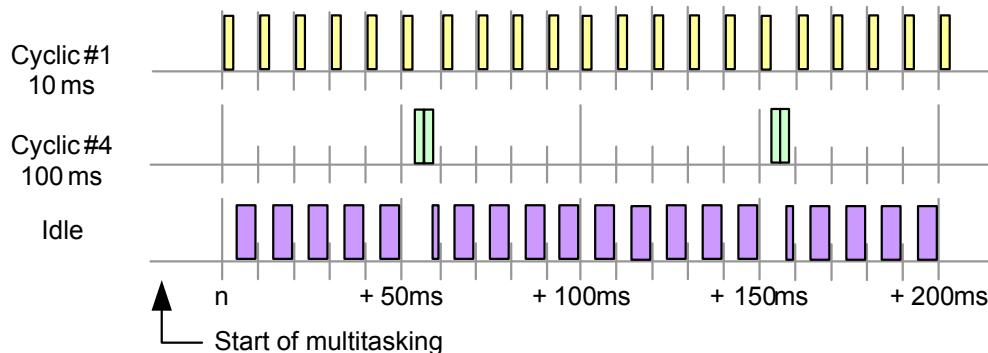


Figure 55: Idle time in the cyclic system

The basic task class for the idle time is established in the CPU configuration. The configured idle time of the system is provided in the context of this task class.

The **Profiler** can be used to determine the idle time. The **System Diagnostics Manager** can also be used to display the average system load.

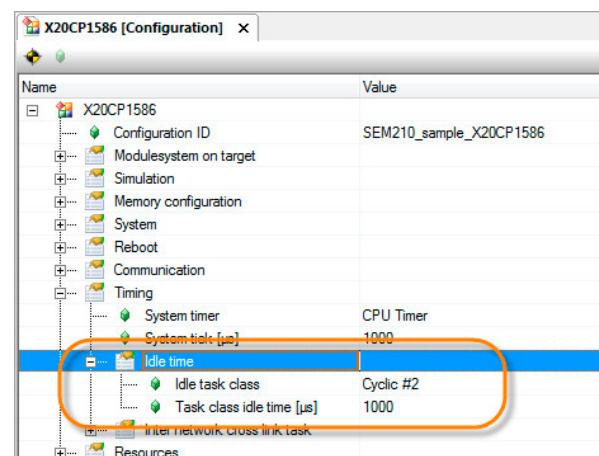


Figure 56: CPU configuration: Basic task class for idle time



If the idle time is not sufficient, the required amount of idle time can be increased by moving programs in slow task classes or by adjusting the task class cycle time.



Real-time operating system \ Method of operation \ Runtime performance \ Scheduling \ Idle time

5.3.4 Starting task classes

Not all task classes are started simultaneously after the multitasking system has been started.

The starting point is always half of the task class cycle time.

This means, for example, that the 100 ms task class will be started after 50 ms. Distributing the start time of all task classes makes better use of processor performance and allows, for example, output jitter to be kept to a minimum.



Real-time operating system \ Target systems \ Target systems - SG4 \ Runtime behavior - SG4
 \ Start of cyclic tasks

5.3.5 Task interrupted by another task

Task class priority makes it possible for a task of higher priority to interrupt a task of priority task class that takes longer.

Using the Watch window, the limit of the variable "udiEndValue" can be changed in such a way in tasks "Loop" and "Loop1" that the task "Loop1" is interrupted exactly **two times** by the task "Loop".

The schematic diagram shows how the timing can look in this case. In order to be complete and promote better understanding, the image was updated with the timing of task class #3.

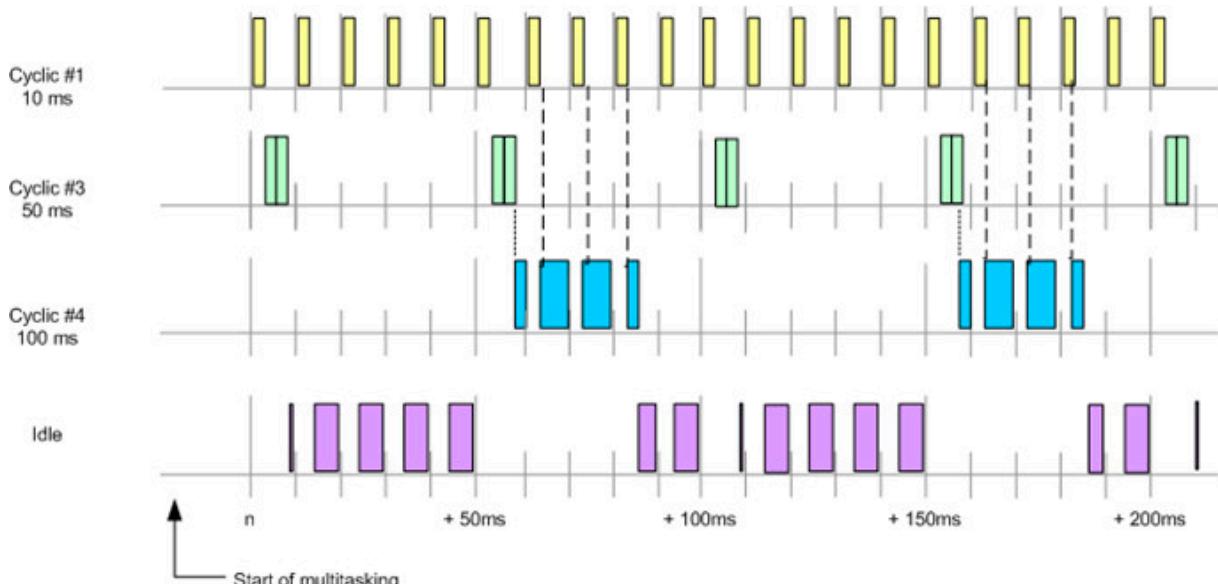


Figure 57: Multitasking with task class #1, #3 and #4; Task class #4 is interrupted by task class #1



For task class #4, a consistent I/O input image is available over the entire execution time. That's why the tasks in this task class are not influenced by the interruption.

5.4 Cycle time and tolerance

Each task class has a predefined cycle time. All of the tasks in a task class must be executed within this cycle time.

The cycle time of a task class is defined when a project is created and can be changed by the user later depending on requirements.

If the sum of the runtimes of the tasks in a task class exceeds the configured cycle time, a cycle time violation occurs. A configurable tolerance delays the cycle time violation. A restart is performed.

The cycle time and tolerance can be configured in the properties of the task class. Open the Properties window from the shortcut menu for the task class.

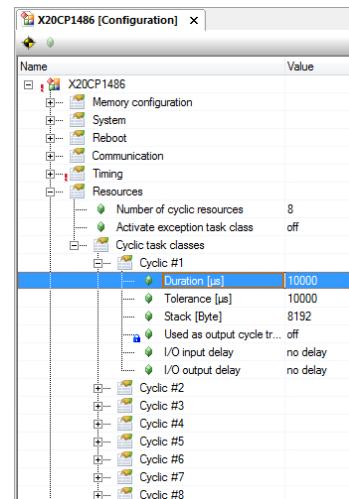


Figure 58: Configuring the cycle time and tolerance



If the configured cycle time of a task class is exceeded, the start of the next cycle is moved backwards by the set tolerance. This can lead to problems in the timing of the application. This behavior is avoided by setting the tolerance to the value "0".



Programming \ Editors \ Configuration editors \ Hardware configuration \ CPU configuration

Exercise: Check the system load using the "Loop" program

In the "Loop" task, which runs in task class #1, an increased runtime must be triggered by changing the value of the "udEndValue" variable.

Using the **System Diagnostics Manager**, the system load caused by the tasks and the available idle time are monitored.

- 1) Set the "udEndValue" variable to 50000

Determine the system load using System Diagnostics Manager

- 2) Increase "udEndValue" variable in steps

The results in the System Diagnostics Manager must be analyzed between the respective steps.

In the System Diagnostics Manager, the average system load is displayed over a selectable time period. In the graphic, the mean and the maximum can be read.

Details about the runtime behavior of a certain task can only be obtained with the Profiler. Additional exercises are planned in training module "TM223 - Automation Studio Diagnostics".



Figure 59: Display of the CPU load in the System Diagnostics Manager



Diagnostics and service \ Diagnostic tool

- System Diagnostics Manager (SDM)
- Profiler

5.4.1 Cycle time violation

Automation Runtime monitors the timing when executing the task classes. A cycle time violation is triggered if the execution time of the tasks exceeds the configured cycle time. If a cycle time violation is detected, the target system starts up in SERVICE mode.

```
ANSL: tcpip/RT=1000 /SDT=5 /DAIP=10.43.15.34 /REPO=11159 /ANSL=1 /PT=... 4PPC70.0573-20B I4.25 SERV
```

Figure 60: CPU in SERVICE mode

In this example, the cycle time violation is the result of changing values in the Watch window. When a cycle time violation occurs, all values in the Watch window are "frozen" and initialized with 0 after restarting in SERVICE mode.

Use the Logger to analyze why the system rebooted in SERVICE mode.

The Logger window can be opened by selecting **<Open> / <Logger>** from the main menu or using the keyboard shortcut **<CTRL> + <L>**.

Runtime performance

In the image, a cycle time violation in task class #1 was entered as the cause of error.

Logger Entries: 5 (Errors: 1, Warnings: 1, Informations: 2, Success: 1)					
Object Name	Visible	Continuous	Severity	Time	ID
Online	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1 Error / Syst...	2016-07-15 10:36:05,762000	9124
System	<input checked="" type="checkbox"/>	<input type="checkbox"/>	2 Success	2016-07-15 08:32:52,867000	3157279
User	<input type="checkbox"/>	<input type="checkbox"/>	3 Warning	2016-07-15 08:32:29,762000	30028
Fieldbus	<input type="checkbox"/>	<input type="checkbox"/>	4 Information	2016-07-15 08:32:22,689000	31280
Safety	<input type="checkbox"/>	<input type="checkbox"/>	5 Information	2016-07-15 08:32:19,926000	9200
Connectivity	<input type="checkbox"/>	<input type="checkbox"/>			

Figure 61: Analyze the cause of the error in the Logger: Cycle time violation in task class #1



The cause of the cycle time violation can be determined using the Profiler. Exercises for this are included in training module TM223 – Automation Studio Diagnostics.



When the target system is restarted – either by turning the power OFF/ON, or by performing a warm restart or reset – it boots in RUN mode.



Diagnostics and service \ Diagnostic tools \ Logger window

Exercise: Increase the value of the "udEndValue" variable until a cycle time violation occurs

The goal of this exercise is to raise the "udEndValue" variable in the "Loop" program incrementally until a cycle time violation occurs. The target system is restarted. The controller starts in SERVICE mode. Next an analysis with the Logger has to be performed.

- 1) Increase the value of "udEndValue" until a cycle time violation occurs
- 2) Analyze entry for the cycle time violation in the Logger.

5.4.2 Reacting to cycle time violations in the application program

In a live production system, changing to SERVICE mode without an option for a response is undesired. Events like a cycle time violation can be reacted to in an exception program.

The exception task class is enabled in the properties of the CPU configuration under the Resources category.

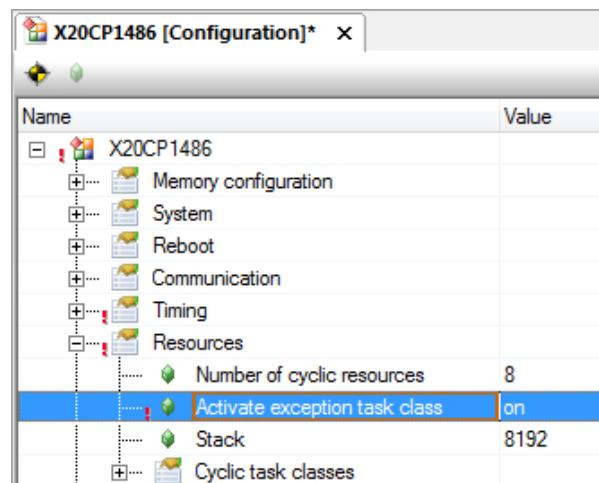


Figure 62: Enabling the exception task class

A program is added from the Logical View into the software configuration in the exception task class.

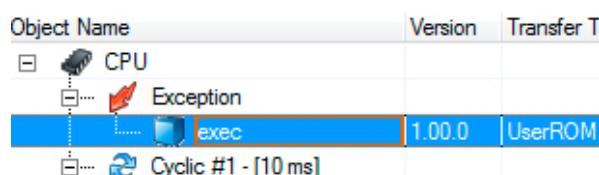


Figure 63: Assign program in exception task class

An exception number is specified in the properties of the exception task. The exception number is specified by the system and can be looked up in Automation Help.

A cycle time violation (exception no. 145) occurring at runtime will call this task. Depending on requirement, you can react to the cycle time violation in the exception task.

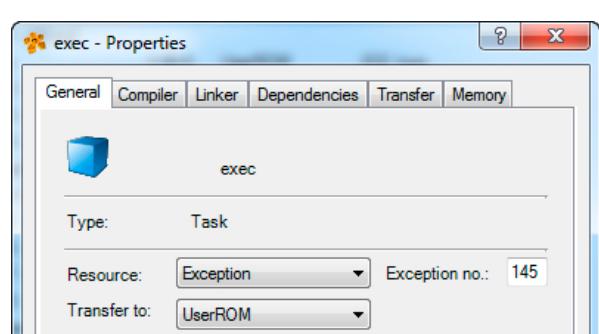


Figure 64: Configuring the exception task



Programming \ Editors \ Configuration editors \ Hardware configuration \ CPU configuration \ SGx \ CPU properties - Resources
 Real-time operating system \ Target systems \ Target systems - SG4 \ Runtime behavior - SG4 \ Exception task class - SG4

5.4.3 Task class with high tolerance

Tasks that are executed as quickly as possible but with a low priority should be executed in task class #8.

Runtime performance

If possible, this is called every 10 ms, but can be interrupted by any higher priority task class. The configured tolerance determines that task class #8 must be executed to completion at least once every 30 seconds.

Tasks that should be assigned to task class #8:

- File access from application program
- Complex, non-time-critical calculations

5.5 Settings when transferring programs

Online installation

Programs and configurations are transferred after changing to the target system. A restart is required when changing a system configuration.

When transferring program changes, you have to decide whether this occurs during development or on the production system.

Depending on the use case, the transfer settings for project installation can be adjusted. The image shows the standard values for project installation.

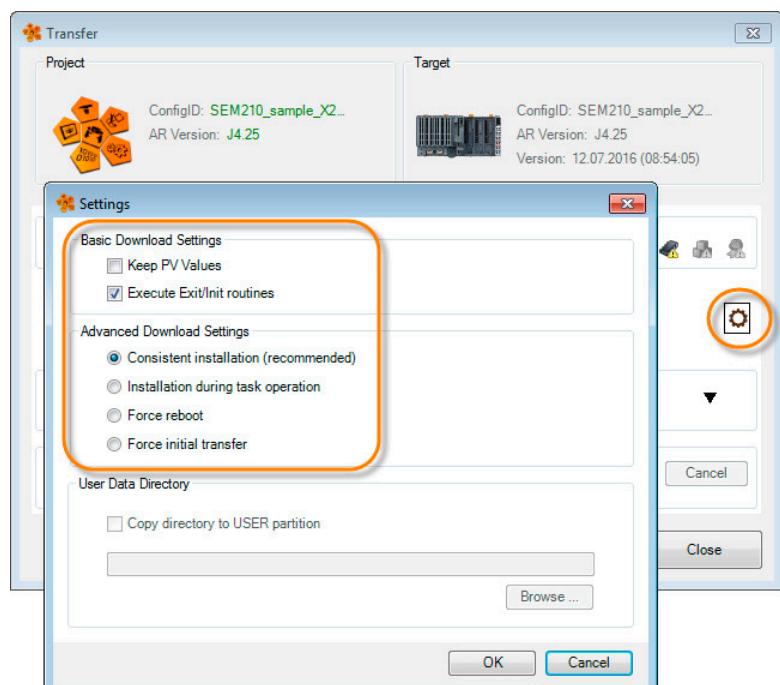


Figure 65: Configuration of the transfer settings for project installation

Keep PV values

This setting defines whether the values of process variables are retained over the installation procedure for existing tasks and tasks updated by the transfer.

Execute Exit/Init routines

If resources (memory, interfaces) were used in the initialization or cyclic subroutine, then these resources can be freed up again using the Exit subroutine.

Consistent installation

All task classes are halted during installation. This guarantees the consistency of all tasks that run on the target system.

Installation during task operation

Only those tasks are halted that are being installed during the installation procedure. The other tasks continue to be executed cyclically and are unaffected by the installation.

Force restart or initial transfer

Via the force restart option, the target system is restarted after the project transfer. An initial transfer is executed with the force initial transfer option.

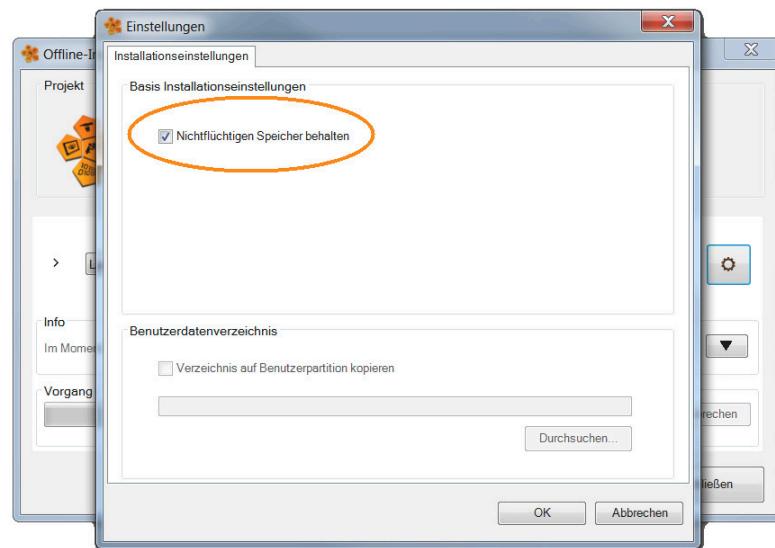
User directory

If a user partition has been defined, user data is copied to the user partition via this option.

See: ["Configuration ID and partitioning" on page 15](#)

 Project management \ Project installation

- Scenarios \ Online update
- Performing project installation \ Settings \ Installation settings



Offline installation

During offline installation, the installation settings allow nonvolatile memory to be retained. This means that the values of retain variable and permanent variables are retained, regardless of whether the CPU is performing a warm or cold restart.

Figure 66: Installation setting for offline installation

Exercise: Adjust transfer settings

In the initialization subroutine of the "loop" task, the limit for the loop is set to the value 1000. The goal of these exercises is to experience effects of the extended transfer settings. Change the limit of the loop in the variable monitor at runtime. Then change something in the "loop" program. Changing a comment is enough. Compile and transfer the change. Using the transfer settings, the limit previously set in the variable monitor must be retained.

- 1) Change "udEndValue" variable to an arbitrary value
- 2) Execute the program change in the "loop" program
- 3) Start compilation and transfer
- 4) Adjust transfer settings
- 5) Check if the value of "udEndValue" is retained

Runtime performance

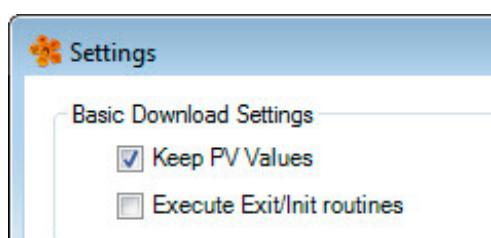


Figure 67: Settings for preserving the values of process variables; the execution of init/exit programs is prevented

6 I/O handling

A central function of a controller is to transport input and output states deterministically and quickly from and to I/O terminals in the application.

Automation Runtime I/O Management is designed to meet the highest requirements for **response times** and **configurability**.

The I/O scheduler transfers the input image from the I/O terminal before the beginning of the task class and the output image at the end of the task in a task class.

The image shows the basic sequence of I/O handling in Automation Runtime. The I/O scheduler makes a consistent input image available to the task class. This is indicated with the green arrow. At the end of the last task in the task class, the output data is transferred to the I/O scheduler. This is indicated with the red arrows.

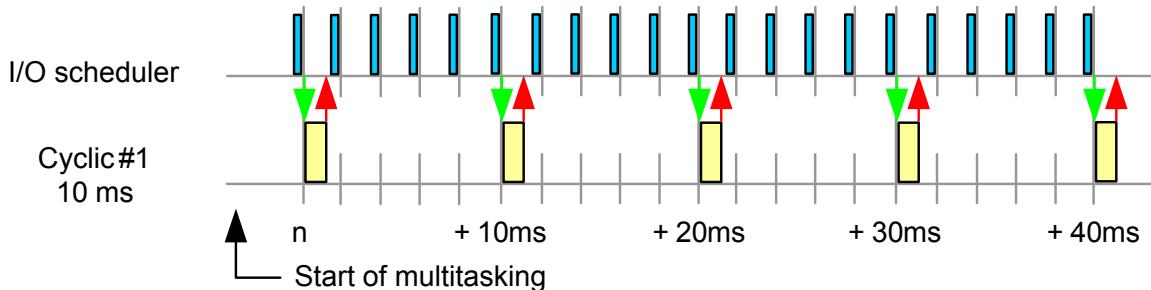


Figure 68: I/O scheduler: Provide input image; write the output image at the end of the execution time of the tasks.

In the I/O mapping, process variables are connected with I/O module channels. The I/O scheduler is controlled by the I/O mapping.

By default, task class #1 is set to 10 milliseconds.

The default configuration for the POWERLINK and X2X interfaces is a cycle time of 2 milliseconds.

The task class cycle time can be adjusted to the I/O cycle time in the CPU configuration.



Programming \ Editors \ Configuration editors \ Hardware configuration \ CPU configuration

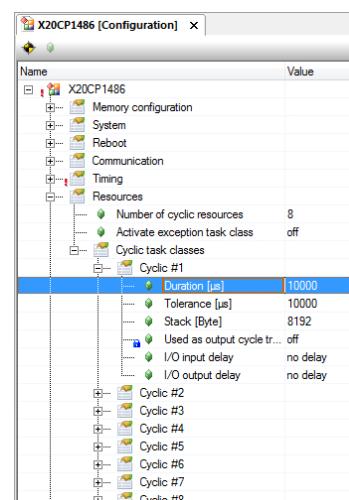


Figure 69: Configuring the cycle time for task class #1



Real-time operating system \ Target systems \Target systems - SG4 \ I/O management - SG4

Real-time operating system \ Target systems \Target systems - SG4 \ I/O management \ Method of operation

6.1 I/O configuration and I/O mapping

6.1.1 I/O configuration

Module-specific properties are configured in the **I/O configuration**.

The offered functionality of I/O modules continues to open up many implementation options and operating modes in which they can be used. The behavior for module monitoring, individual configurations and function models, among others, are adjusted in the I/O configuration.

The I/O configuration is opened via the shortcut menu for the desired I/O module. The configuration options of an I/O module are documented in the respective data sheet under section "Register description".

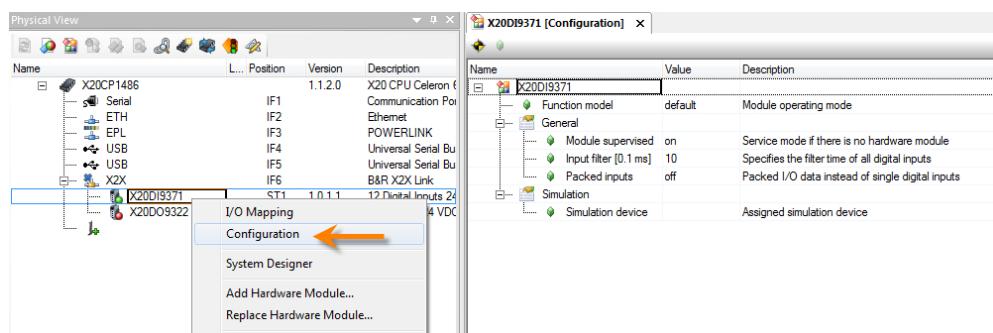


Figure 70: I/O configuration of a digital input module



Programming \ Editors \ Configuration editors \ Hardware configuration \ I/O configuration

6.1.2 I/O mapping

The **I/O mapping** defines which data from the I/O image is assigned to a process variable.

Each variable in a .var file that is used in a program can be assigned to a channel of an I/O module, regardless of its scope.

The I/O mapping is opened via the shortcut menu for an I/O module.

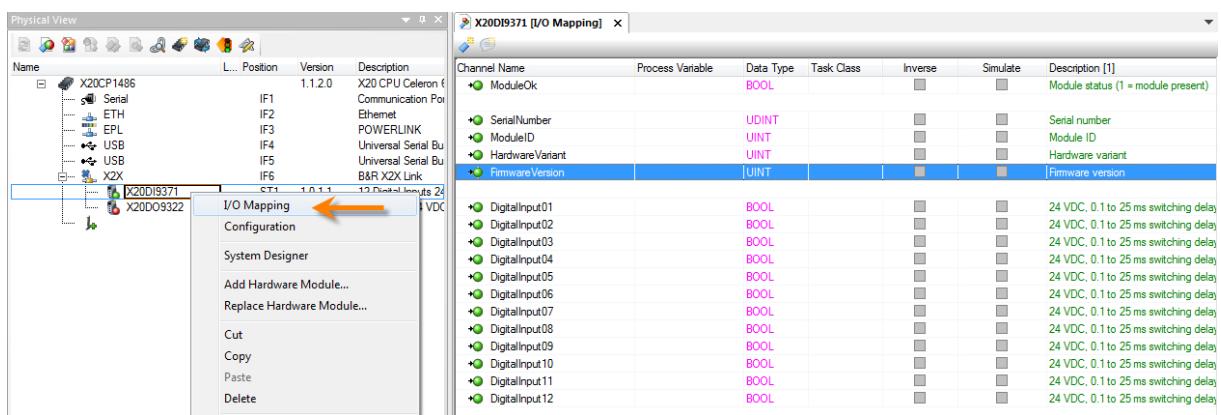


Figure 71: I/O mapping of a digital input card



For global variables, the channel can be assigned a task class in which the data of the I/O image should be transferred.

If "Automatic" is set and the project built, Automation Studio will automatically determine the fastest task class where the variable is being used.



Programming \ Editors \ Configuration editors \ I/O mapping

I/O handling

6.2 Handling I/O images

The I/O scheduler provides a consistent I/O image for the execution of all tasks in a task class and starts the task class at the exactly configured time. This ensures that the inputs at the beginning of the task class and the outputs at the end of the task class are transferred.



Each task class uses a separate I/O image. The input states remain consistent during the entire task runtime.

Cycle of I/O data

The provision of I/O data occurs in the set I/O cycle time of the bus system used.

The configuration is opened via the shortcut menu for the respective fieldbus.

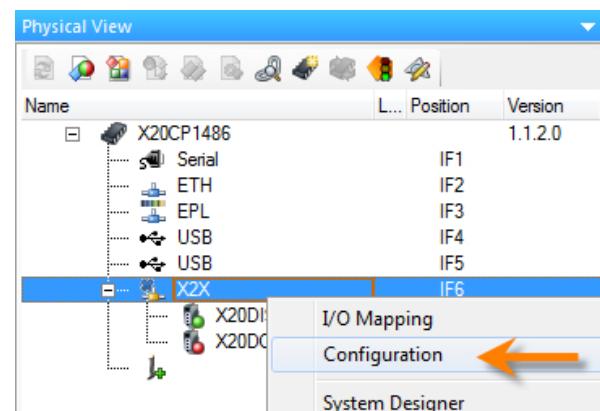


Figure 72: Opening the properties of the X2X Link interface

The I/O cycle time configured in the properties is used as the basis for copying I/O data to the I/O image. This means that the inputs are provided as an I/O image and the output image is described in the set X2X cycle time.

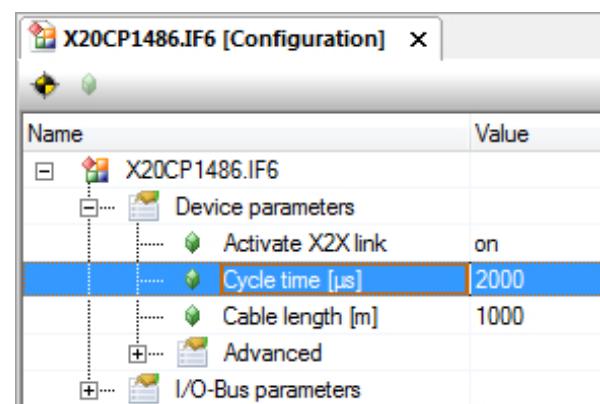


Figure 73: Configuration of the X2X cycle time



Network Analyzer makes it possible to check whether the configured POWERLINK and X2X Link cycle times are permissible. It is opened by selecting <Open> / <Network Analyzer> from the main menu.



Diagnostics and service \ Diagnostic tools \ Network analyzer

The graphic shows that an input image is provided every 2 milliseconds. For the application, this means that the input data is not older than the configured cycle time when the task class starts; the output data will be written after this time has passed at the latest.

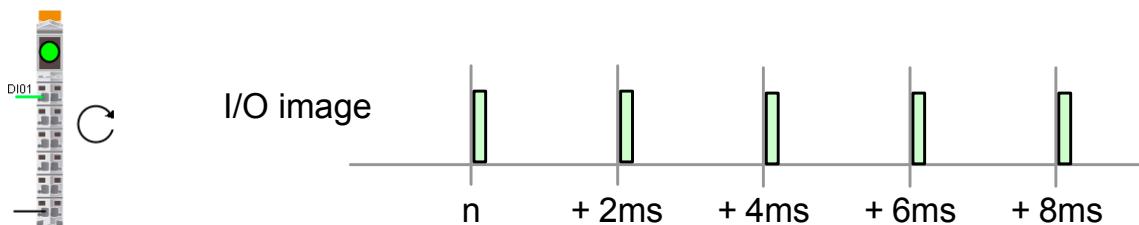
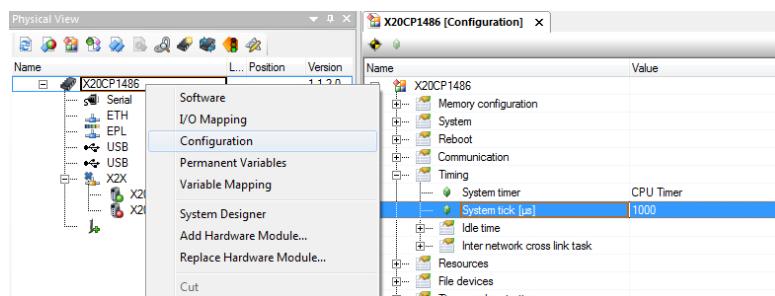


Figure 74: Reading the input image

I/O data in the task class

The I/O scheduler controls when the I/O image for the task classes is provided and the task classes are started.



By default, the I/O scheduler is opened with a system tick of 1000 μ s.

The system tick is set in the CPU configuration in the "Timing" category.

Figure 75: Configurable system tick

The graphic shows that the I/O scheduler is called twice as often as the I/O image is updated.

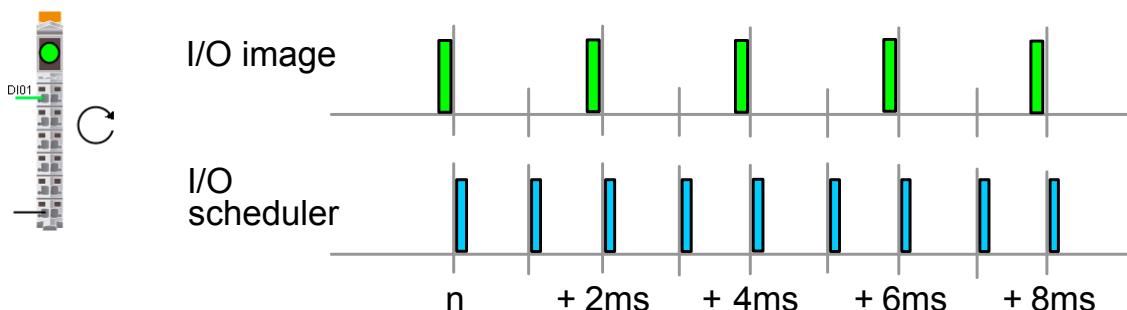


Figure 76: Reading the input image; Calling the I/O scheduler

I/O handling

The cycle time of the fieldbus can be used as a system timer in the timing configuration. By default, the fieldbus cycle time is applied in a 1:1 ratio for the I/O scheduler. I/O cycle and I/O scheduler now work synchronously in the set ratio.

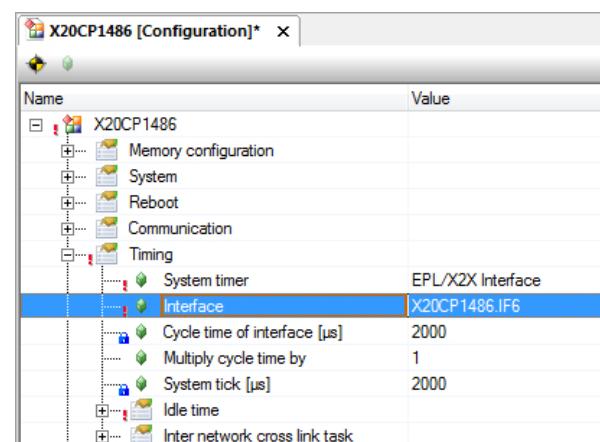


Figure 77: Synchronizing the system tick

By using the fieldbus timer as a system timer and multiplying the cycle time by a factor of 1, the I/O scheduler is called with the configured I/O cycle time.

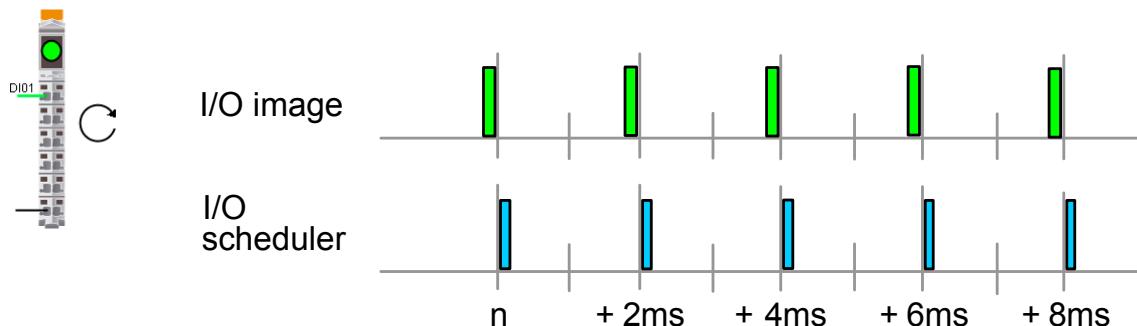


Figure 78: Synchronizing the I/O image and I/O scheduler

As is established in the I/O mapping, the I/O data from the input image is assigned to the configured process variables.

For the tasks of a task class, this results in the following when the I/O scheduler is called in a cycle time of 2 milliseconds:

Task class #1

The I/O scheduler starts task class #1 every 10 ms and provides it with the input image for the assigned variables (green arrow).

At the end of the tasks in task class #1, the variables of the assigned outputs are written to the output image (red arrow).

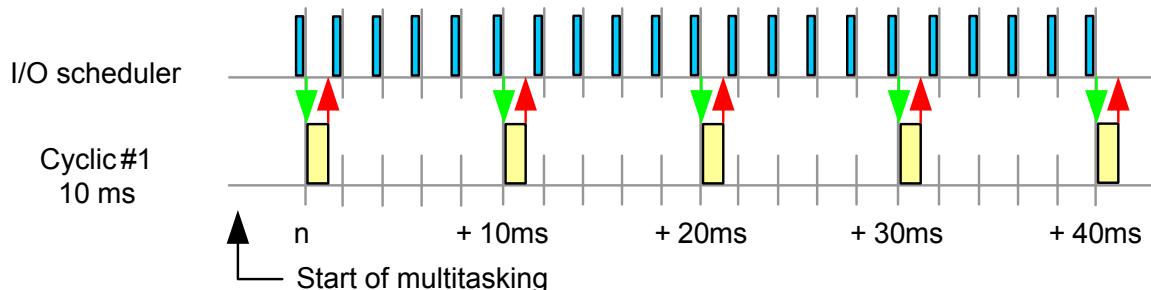


Figure 79: I/O handling for task class #1

Task class #4

The I/O scheduler starts task class #4 at the time, $n+50$ ms, $n+150$ ms and provides it with the input image for the assigned variables (green arrow).

At the end of the tasks in task class #4, the variables of the assigned outputs are written to the output image (red arrow).

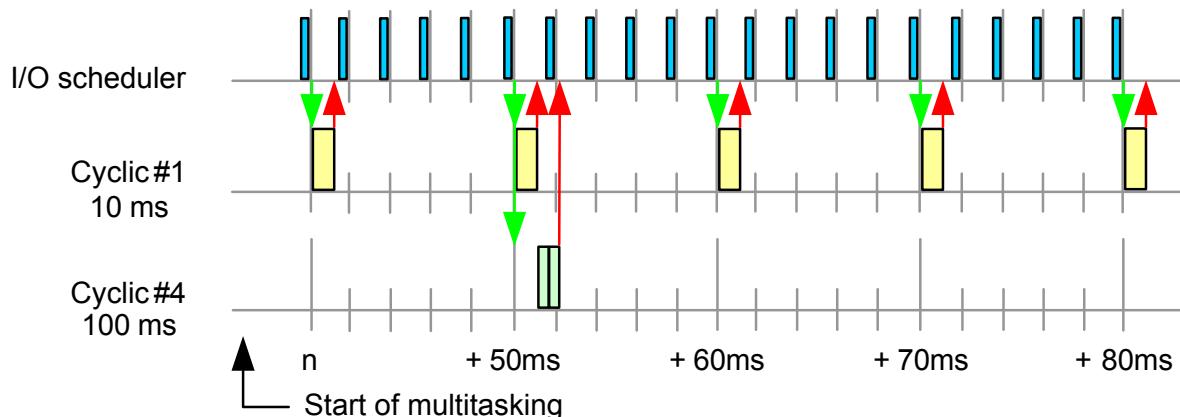


Figure 80: I/O handling for task class #4



Real-time operating system \ Target systems \ Target systems - SG4 \ I/O management - SG4

6.2.1 Startup

Booting the controller transfers the I/O configuration to the I/O modules and initializes the interfaces and fieldbus devices. This occurs even before the programs are initialized. This ensures that valid I/O data can be accessed in the initialization subroutine.

6.2.2 Mapping I/O data

The following image displays the relationship between the I/O configuration and the I/O mapping.

I/O handling

The I/O configuration is transferred to the module when the controller boots up or when a configured I/O module is recognized.

In cyclic operation, the input data is provided as a memory block and distributed by the I/O mapping to the configured process variables. In order to write the output data, the process variables are collected and transferred as a memory block to the I/O system.

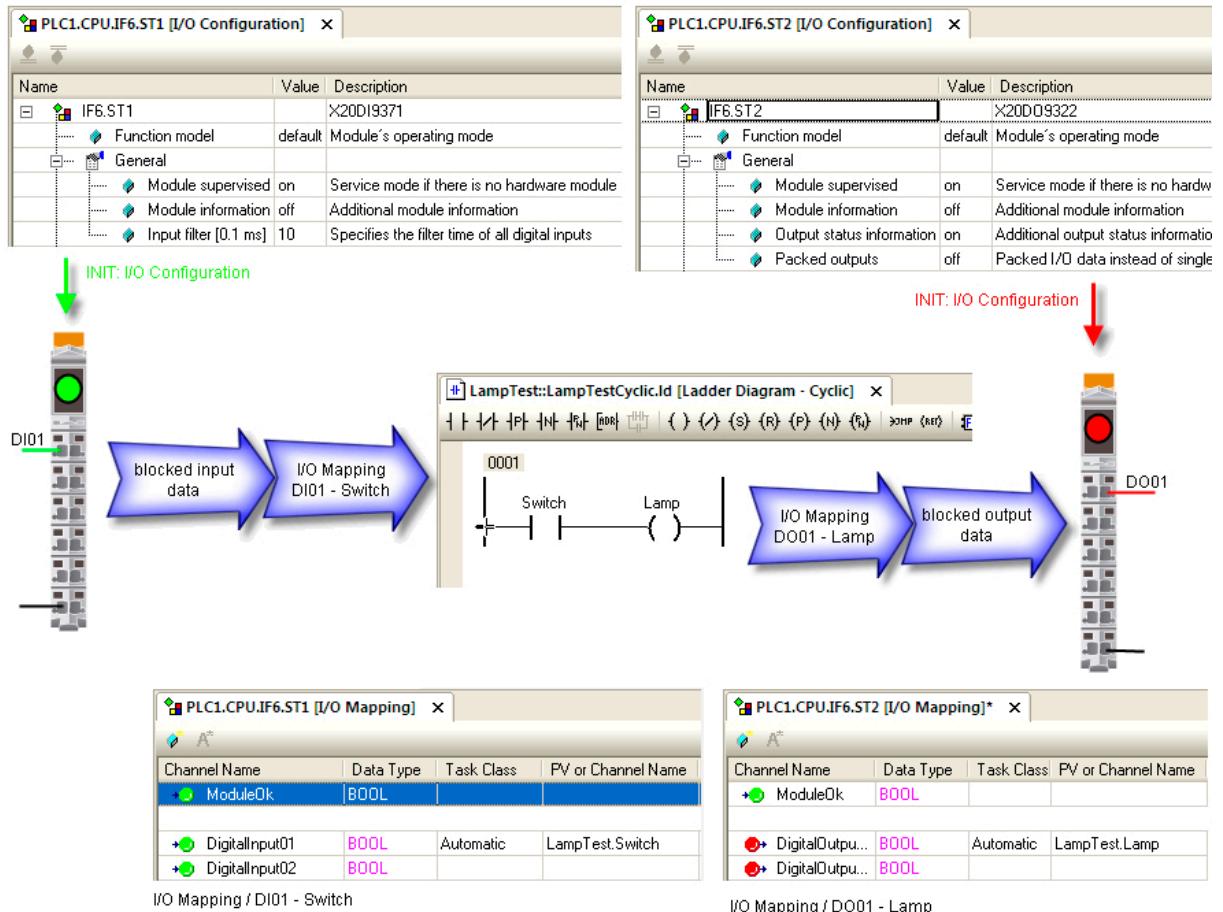


Figure 81: Basic I/O functionality

6.2.3 Settings for the timing of I/O images

There are configuration options available for the task classes for the chronological handling of I/O images. The standard case is the immediate writing of the outputs when ending the last task in the task class. This causes jitter on the outputs when the task runtime fluctuates.

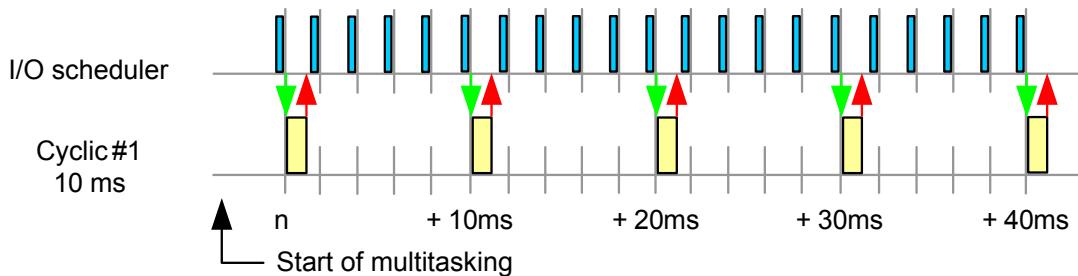


Figure 82: Outputs are transferred immediately to the I/O system after the task has ended.

Alternatively, the time when the input images are read and the output images written can be delayed via the CPU settings for the task class.

If for task class #1 the parameter "I/O output delay" is configured to the value "To end of cycle". This way output data is only transferred to the I/O scheduler at the end of the task class. The effect of this is that even during fluctuating task runtime, the outputs of a task class are written jitter-free.

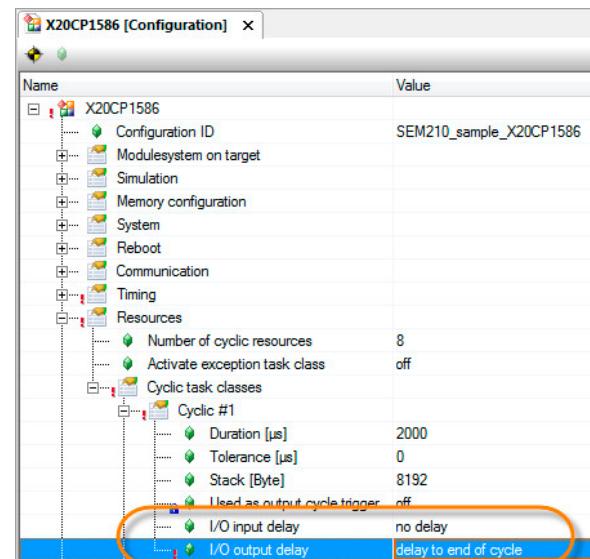


Figure 83: Configuration for the deceleration of the output data to the end of the cycle

The graphic shows the effect of the delay of the output image at the end of the cycle.

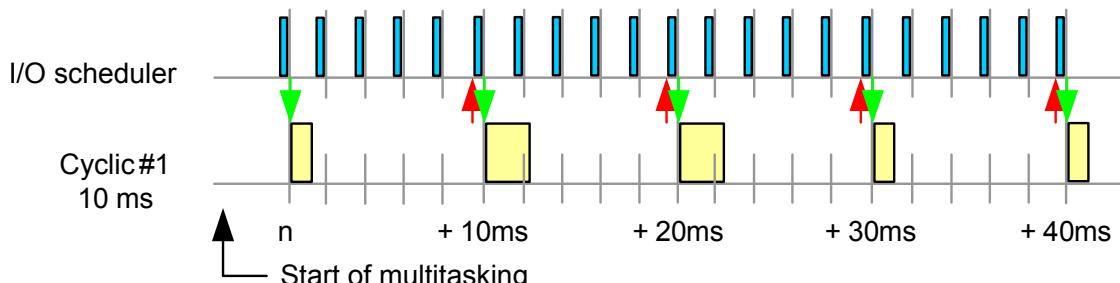


Figure 84: Jitter-free writing of output image at end of cycle



Real-time operating system \ Target systems \ Target systems - SG4 \ I/O management - SG4
 Real-time operating system \ Target systems \ Target systems - SG4 \ I/O management - SG4
 \ Method of operation

I/O handling

6.3 Error handling for I/O modules

Every configured I/O module is monitored by the I/O system. The user can configure how the system responds to error situations.

The "Module monitoring" property in the **I/O Configuration** can be used to enable or disable the monitoring of an I/O module.

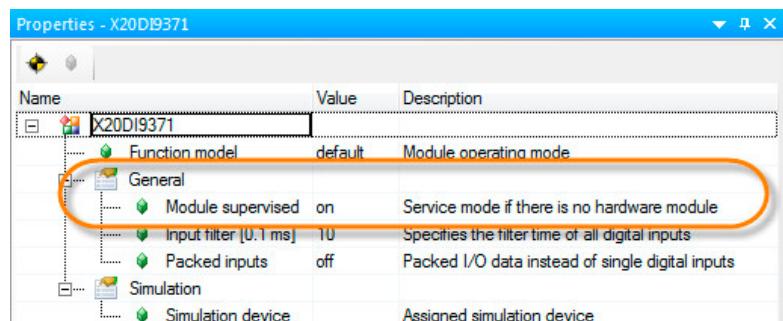


Figure 85: Configuration of module monitoring

Monitoring active

When the module monitoring by Automation Runtime is enabled, the following states will cause a restart in SERVICE mode:

- The module defined for a slot is not present (or not connected).
- The module physically connected in a slot doesn't match the module actually configured for the slot.
- The module is not addressed anymore during operation by the I/O system.

If a missing, incorrectly configured or incorrectly inserted I/O module is recognized on startup or during runtime, then this is logged in the "System" Logger file.

Logger Entries: 39 (Errors: 5, Warnings: 12, Informations: 18, Success: 4)							
Object Name	Visible	Continuous	Severity	Time	ID	Area	
Online	<input checked="" type="checkbox"/>	<input type="checkbox"/>	3	Error	2016-07-20 09:09:36.645000	30024	Module removed while running
System	<input checked="" type="checkbox"/>	<input type="checkbox"/>	4	Information	2016-07-20 09:04:18.046000	10561	NC manager info (see 'ASCII Data')
							IF6.ST3.IO ncglobal_action_ACP1

Figure 86: Logger entry "module removed while running"



The slot of the module can be identified using the ASCII data in the Logger entry. "IF6.ST3" means that the "ST3" module on the IF6 interface, which is the X2X interface in this case, was disconnected. This is the third slot on that bus. The CPU interface names can be viewed in the Physical View as well as in the data sheet of the CPU being used.

Monitoring disabled

When module monitoring is disabled, the I/O module can be monitored from the application by variable mapping on the "ModuleOK" channel. Missing or incorrectly inserted modules don't cause a reboot in SERVICE mode. Only the value of the "ModuleOk" channel becomes "FALSE". The monitoring of the modules is therefore the responsibility of the application.

Channel Name	Data Type	Task Class	PV or Channel Name	Inverse	Simulate	Description [1]
+ ModuleOk	BOOL	Automatic	modul_ok01	<input type="checkbox"/>	<input type="checkbox"/>	Module status (1 = module present)

Figure 87: Module monitoring via the application



Real-time operating system \ Target systems \ Target systems - SG4 \ I/O management - SG4
 \ Method of operation \ Error handling

Exercise: Remove bus connection or I/O module during operation

Cancel the connection to a X20 bus controller during operation or remove an I/O during runtime. The controller is restarted in SERVICE mode and the execution of the controller application is canceled. Read the Logger file and analyze the entries.

- 1) Remove bus station or I/O module during operation
- 2) Wait for reboot and startup in SERVICE mode
- 3) Read and analyze the Logger file "System"

6.4 reACTION Technology

The response times that a control system is able to achieve represent a key performance factor. High-end controllers, a fast real-time network and intelligent system architectures can push the limit – often with considerable effort – down to 100 µs or lower. The market still demands even shorter response times, however.

Through the use of reACTION Technology, the execution of small logic programs is passed on from the controller directly to an I/O module. reACTION Technology programs run on the reACTION I/O module with a very fast cycle time, which allows response times in the µs range.

In addition to the short response times made possible by the reACTION I/O module, the load on the controller and fieldbus is also reduced.

reACTION Technology programs are configured in the Logical View. When transferring the project, the reACTION Technology program is installed directly on the I/O module and executed there.

For configuration and diagnosis of reACTION Technology, data points are available in I/O mapping on the reACTION I/O modules.





Programming \ Programs \ reACTION Technology

- Technology description
- Examples for use of reACTION Technology
- reACTION diagram

7 Summary

The operating system provides an interface between the application and the hardware while ensuring consistent management of the resources on the respective target system.

Multitasking in Automation Runtime makes it possible to design an application with a modular structure. The arrangement of the application in tasks grouped into task classes enables optimal usage of resources. The timing of the application can be configured by adapting the multitasking system to the user's unique requirements.

This makes it possible to achieve an optimal configuration to maximize the performance of the application and machine while using existing resources efficiently.



Figure 88: Automation Runtime: A software platform for the entire B&R product range

The platform-independent configurable interfaces as well as client and server protocols make Automation Runtime a powerful operating system. The openness of the system makes it possible to directly integrate devices from other manufacturers, establish connections to OPC UA clients and servers and implement the required IT safety mechanisms.

The state of Automation Runtime is always transparent thanks to several diagnostic tools and software libraries. This helps with commissioning and the implementation of individual application requirements.

Offered by the Automation Academy

Offered by the Automation Academy

The Automation Academy provides targeted training courses for our customers as well as our own employees.

At the Automation Academy, you'll develop the skills you need in no time!

Our seminars make it possible for you to improve your knowledge in the field of automation engineering. Once completed, you will be in a position to implement efficient automation solutions using B&R technology. This will make it possible for you to secure a decisive competitive edge by allowing you and your company to react faster to constantly changing market demands.



Seminars



Quality and relevance are essential components of our seminars. The pace of a specific seminar is based strictly on the experience that course participants bring with them and tailored to the requirements they face. A combination of group work and self-study provides the high level of flexibility needed to maximize the learning experience. Each seminar is taught by one of our highly skilled and experienced trainers.

Training modules

Our training modules provide the basis for learning both at seminars as well as for self-study. These compact modules rely on a consistent didactic concept. Their bottom-up structure allows complex, interrelated topics to be learned efficiently and effectively. They serve as the best possible companion to our extensive help system. The training modules are available as downloads and can be ordered as printed versions.

Topic categories:

- ⇒ Control technology
- ⇒ Motion control
- ⇒ Safety technology
- ⇒ HMI
- ⇒ Process control
- ⇒ Diagnostics and service
- ⇒ POWERLINK and openSAFETY

ETA system



The ETA system provides realistic constructions for training, education and laboratory use. Two different basic mechanical constructions can be selected. The ETA light system offers a high degree of mobility, saves space and is well-suited for lab work. The ETA standard system has a sturdy mechanical structure and includes pre-wired sensors and actuators.

Find out more!

Would you like additional training? Are you interested in finding out what the B&R Automation Academy has to offer? You've come to the right place.
Detailed information can be found under the following link:
www.br-automation.com/academy



Offered by the Automation Academy

V2.2.0 ©2018/05/23 by B&R, All rights reserved.
All registered trademarks are the property of their respective owners.
We reserve the right to make technical changes.



TM213TRE.444-ENG