Automation Runtime

TM213

Perfection in Automation
www.br-automation.com

Requirements

Training modules:	TM210 – The Basics of Automation Studio TM211 – Automation Studio Onlinecommunikation
Software	Automation Studio 3.0.90
Hardware	X20CP1485

TABLE OF CONTENTS

1 INTRODUCTION

An integral component of Automation Studio is the real-time operating system Automation Runtime.

Automation Runtime makes up the software kernel which allows applications to run on a target system.

To meet demands, Automation Runtime includes a modular structure and the ability to quickly execute the application repeatedly within a precise timeframe. This makes it possible to achieve maximum quantity, quality and precision during runtime.

Automation Runtime target systems

This training module provides a general overview of Automation Runtime and its features.

1.1 Training module objectives

Selected examples that illustrate typical applications will help you learn how to model and configure Automation Runtime in Automation Studio for your application.

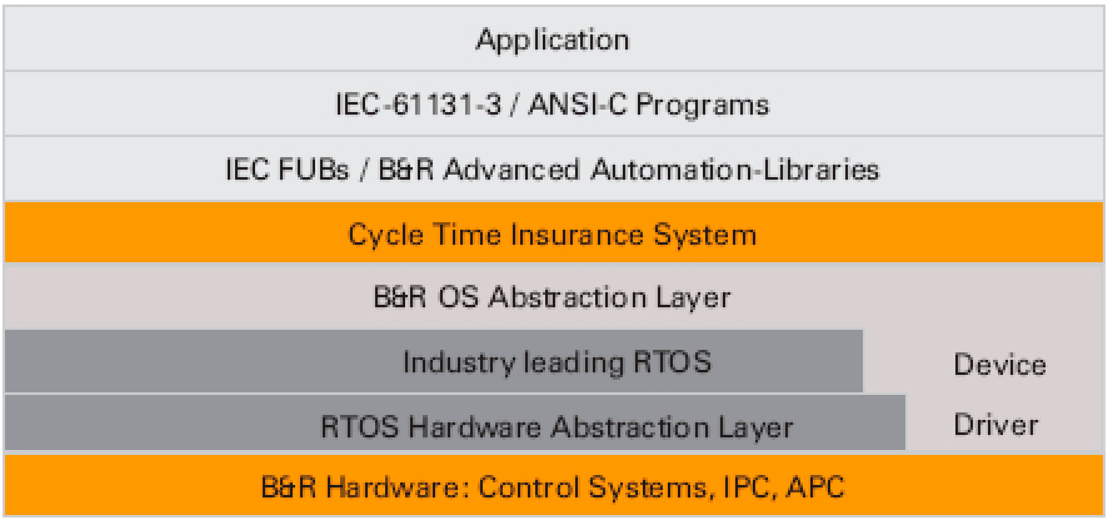
You will learn …

- How to commission and configure the target system
- How to work with Automation Runtime
- How to use variables and memory types correctly
- How Automation Runtime operates during runtime for your application

2 REAL-TIME OPERATING SYSTEM

Automation Runtime provides the user with a hardware-independent multitasking tool for creating applications.

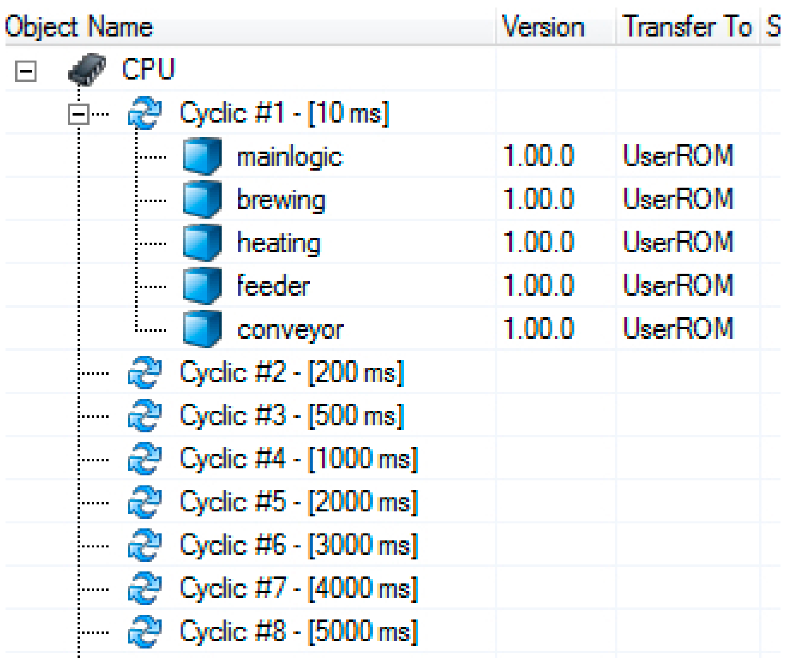
The IEC functions in Automation Runtime make programming faster and easier while helping to avoid errors.


Application
IEC-61131-3 / ANSI-C Programs
IEC FUBs / B&R Advanced Automation-Libraries
Cycle Time Insurance System
B&R OS Abstraction Layer
Industry leading RTOS
RTOS Hardware Abstraction Layer
Device Driver
B&R Hardware: Control Systems, IPC, APC

Operating system architecture

The operating system handles two main tasks: managing hardware and software resources and providing a uniform method of accessing hardware without requiring users to deal with every detail.

A program (also referred to as a task) can be assigned a specific execution time, or task class, in which it is executed cyclically.



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]			

Time basis for executing programs

Real-time operating system

? Real-time operating system

2.1 Requirements and functions

Automation Runtime is fully integrated in the respective target system. It allows application programs to access I/O systems, fieldbus systems and other devices such as interfaces, networks and storage devices.

Demands on Automation Runtime

Automation Runtime provides a number of important functions:

- Runs on all B&R target systems
- Makes the application hardware-independent
- Cyclic runtime system guarantees deterministic behavior
- Allows configuration of priorities, time classes, and jitter tolerance
- Up to eight different time classes with any number of programs
- Guaranteed response to timing violations
- Tolerance can be configured in all task classes
- Extensive function library in accordance with IEC 61131-3

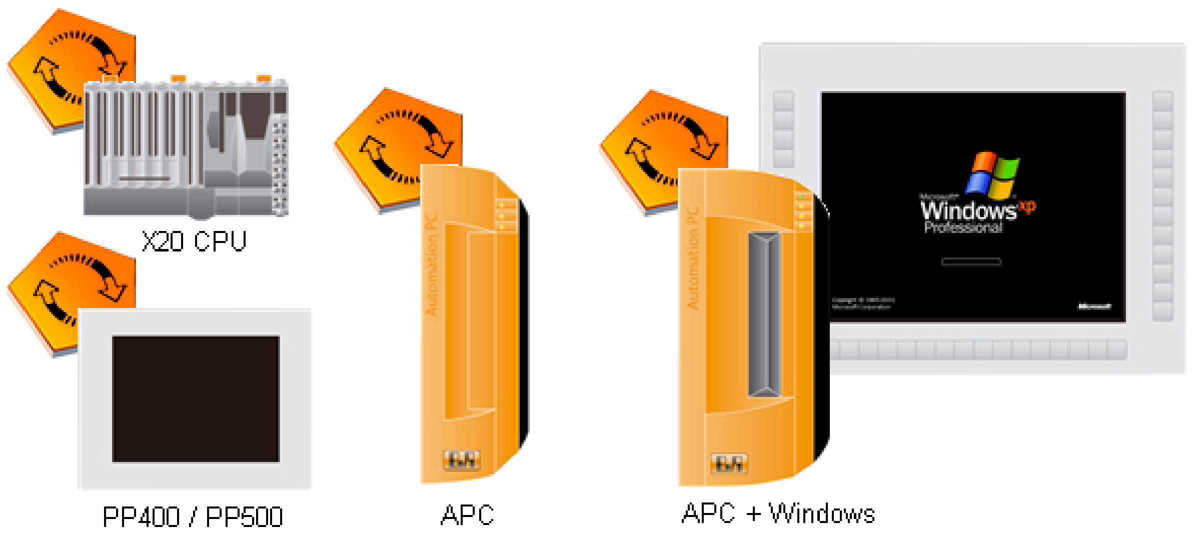
- Access to all networks and bus systems via function calls or the Automation Studio configuration
- Integrated FTP, web and VNC servers

Real-time operating system / Method of operation

2.2 Target systems

Automation Runtime can run on different hardware platforms.

Main examples include a PLC – such as an X20 CPU or a PowerPanel – or a PC-based PLC such as an Automation PC.



SG4 target systems

This training module describes the functionality of SG4 target systems - hardware platforms with Intel-compatible processors.

Further information about SG4 and functionally similar SG3 / SGC target systems can be found in the help system documentation.

Real-time operating system / Target systems

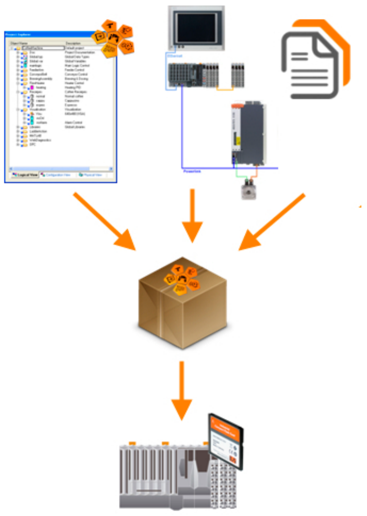
2.3 Installation and commissioning

All SG4 target systems boot Automation Runtime and the Automation Studio project from a Compact-Flash card.

A bootable CompactFlash card is created using Automation Studio.

During this process, the CompactFlash card is partitioned according to the requirements of the application.

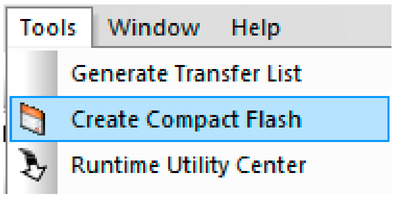
Automation Runtime and the Automation Studio project are installed on the CompactFlash card.


Installation and commissioning

The Windows-based Automation Runtime Windows is the exception on PC-based target systems. In this case, Automation Runtime is installed the using a special setup tool that is found on the Automation Studio DVD.

2.3.1 Creating a functioning CompactFlash card

A CompactFlash card can be created in Automation Studio by selecting <**Tools**> / <**Create Compact Flash**> .

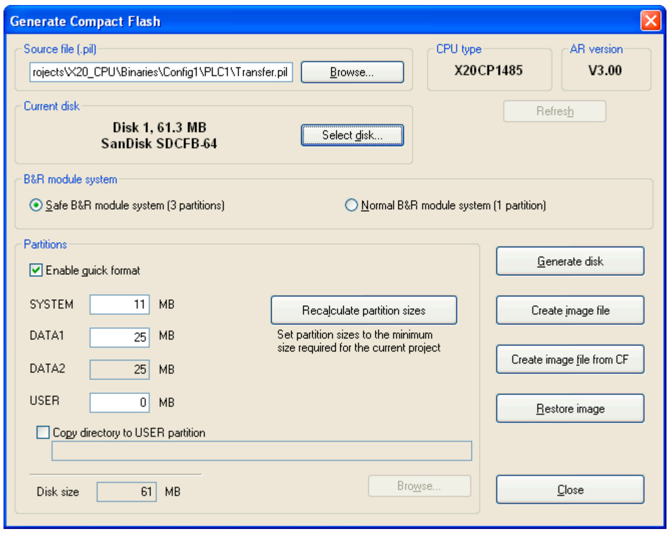


Create CompactFlash

After startup, the project is checked to see if it has already been compiled successfully. If it has not, it is compiled (Build).

In order to create the CompactFlash card, you will need a free PCMCIA slot with a PCMCIA-CF adapter or a USB CompactFlash card reader.

After selecting the CompactFlash card in the following dialog box, you can make user-specific settings and then begin creating the CompactFlash card.



Source file (.pil): rojects\X20_CPU\Binaries\Config1\PLC1\Transfer.pil

CPU type: X20CP1485

AR version: V3.00

Current disk: Disk 1, 61.3 MB SanDisk SDCFB-64

B&R module system: [✓ Safe B&R module system (3 partitions)] Normal B&R module system (1 partition)

Partitions:

[✓] Enable quick format

SYSTEM	11	MB
DATA1	25	MB
DATA2	25	MB
USER	0	MB

Set partition sizes to the minimum size required for the current project

☐ Copy directory to USER partition

Disk size 61 MB

Dialog box for creating a CompactFlash card

On the production system, it is recommended to use the "**Safe B&R module system**" with three or more partitions.

The "**Normal B&R module system**" with one partition can be used during development.

Diagnostics and Service \ Service Tool \ Runtime Utility Center \ Creating a list / data medium \ Creating Compact Flash

2.3.2 Installation by using the online connection

Automation Runtime can also be transfered by using the online connection. The prerequisite is that the online interface has been configured correctly. (4.5.1 "Startup").

Beginning with Default Runtime V3.06 for SG4 systems searching the targets in the network with SNMP works also in BOOT mode.

Connection

A connection to the controller works easiest by using the target system search of Automation Studio. Here, the network will be searched to find B&R controllers. It is possible to change the connection settings inside search dialogue temporary.

Programming \ Build & Transfer \ Establishing a connection to the target system \ Ethernet connection \ Browse for targets

Transfering the Automation Runtime

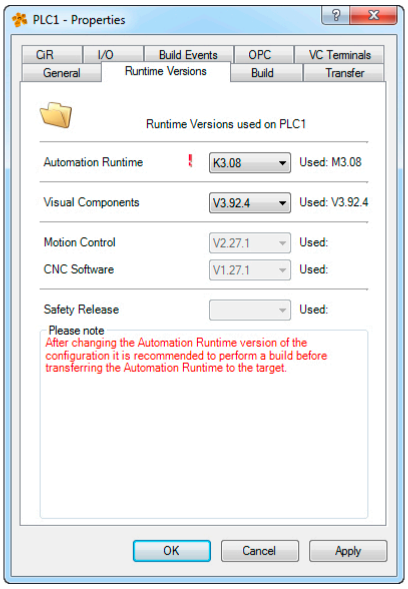
Once the connection is established, the Automation Runtime can be transferred.

Programming \ Build & Transfer \ Online services \ Transfer Automation Runtime \ Transfer to SGx target systems \ Installation via online connection

2.3.3 Changing and upgrading Automation Runtime

If there is a new version of Automation Runtime, Visual Components, Motion or CNC available, it is possible to change the version of Automation Runtime in the Automation Studio project for the active configuration.

Select the version under <**Project**> / <**Change Runtime Versions…**> .



Runtime Versions used on PLC1

Automation Runtime	K3.08	Used: M3.08
Visual Components	V3.92.4	Used: V3.92.4
Motion Control	V2.27.1	Used:
CNC Software	V1.27.1	Used:
Safety Release		Used:

Please note
After changing the Automation Runtime version of the configuration it is recommended to perform a build before transferring the Automation Runtime to the target.

Change the runtime versions

Project Management \ Change the Runtime versions

Changing the Motion, CNC Software and Visual Components versions can affect the hardware configuration since there is only one library for all configurations.

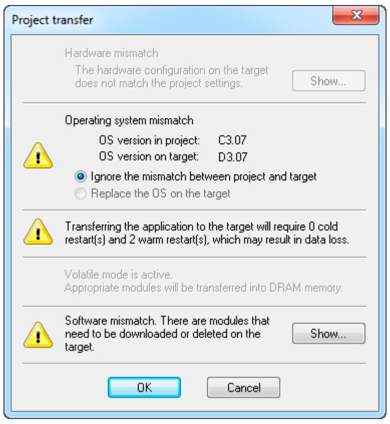
After the Automation Runtime version is changed for this configuration, it is necessary to recompile (rebuild) before transferring the project.

Upgrade using Automation Studio

If Automation Runtime is already installed on the target system, the project can be transferred "**online**" together with the new version of Automation Runtime.

With an Ethernet connection, ensure that the "**sysconf**" module is configured for the target memory "**SystemROM**" (default) since the UserROM is deleted after the new version of Automation Runtime is transferred.

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


Project transfer

Hardware mismatch
The hardware configuration on the target does not match the project settings.

Operating system mismatch
OS version in project: C3.07
OS version on target: D3.07
[✓ Ignore the mismatch between project and target]
Replace the OS on the target

Transferring the application to the target will require 0 cold restart(s) and 2 warm restart(s), which may result in data loss.

Volatile mode is active.
Appropriate modules will be transferred into DRAM memory.

Software mismatch. There are modules that need to be downloaded or deleted on the target.

Version conflict detection

Upgrade 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. This can be transferred using a CD or USB flash drive.

The update requires a PC with an online connection to the target system.

Diagnostics and Service \ Service Tool \ Runtime Utility Center \ Creating a list / data medium \ Cd creation

3 MEMORY MANAGEMENT

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

Parts of these are used exclusively by Automation Runtime during runtime, and the rest is available for the application.

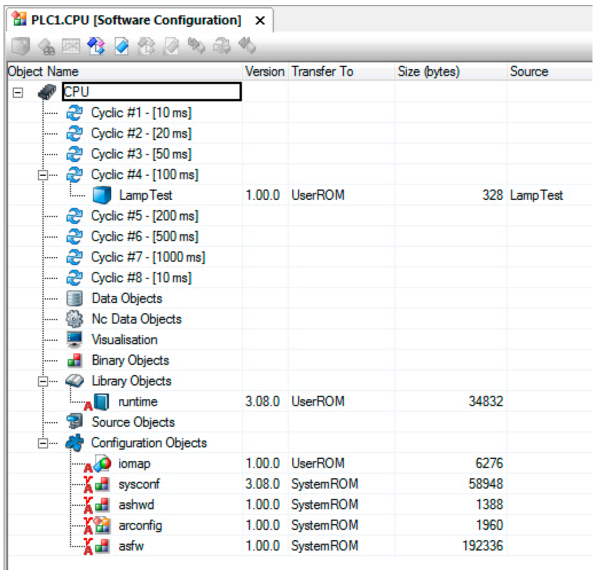
3.1 System and User ROM

During a build, an Automation Studio project generates modules with the extension *.br, which can be executed by Automation Runtime.

In the software configuration, each module is automatically assigned target memory – which is divided into **User ROM** and **SystemROM**.

UserROM is memory on the CompactFlash where all the BR modules for the Automation Studio project are stored.

SystemROM is memory on the CompactFlash where Automation Runtime and the system modules for the project are stored.


PLC1.CPU [Software Configuration]

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				
iomap	1.00.0	UserROM	6276	
sysconf	3.08.0	SystemROM	58948	
ashwd	1.00.0	SystemROM	1388	
arconfig	1.00.0	SystemROM	1960	
asfw	1.00.0	SystemROM	192336	

Target memory for .br modules

CompactFlash = SystemROM + UserROM

3.2 DRAM and SRAM

RAM is fast read/write memory where data for executing Automation Runtime and the Automation Studio application can be loaded and executed.

SG4 target systems always have **DRAM**; **SRAM** is optional.

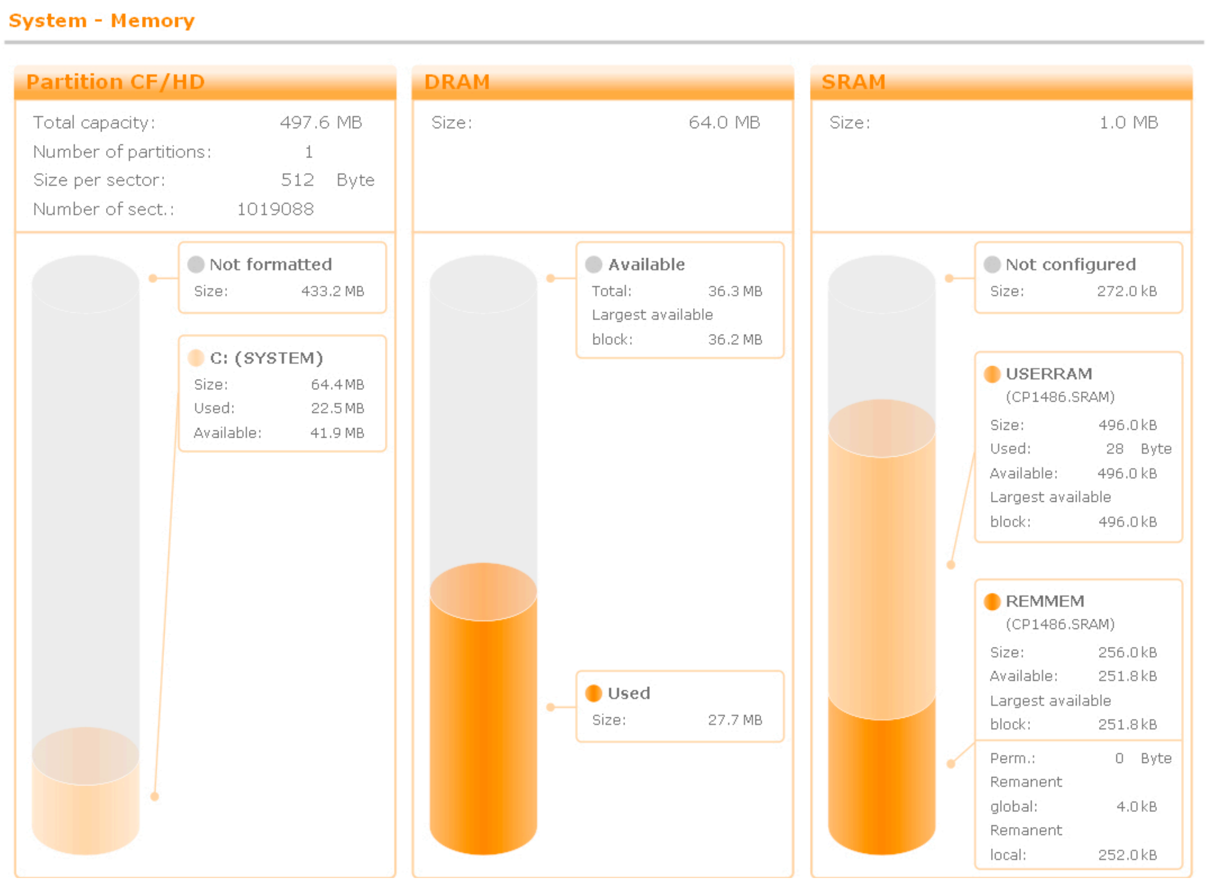
DRAM

SRAM: Unlike DRAM, SRAM (static RAM) is buffered by a battery. This allows nonvolatile data storage (functional battery required).

DRAM: DRAM is RAM memory that has no defined status when the system is turned on. It is typically > 10x faster than SRAM.

When booted, Automation Runtime loads all necessary BR modules to DRAM, allowing them to be accessed quickly.

A BR module requires the local or global variable memory area in RAM that are configured in Automation Studio (see 4.3), and is also responsible for initializing this memory area.


System - Memory

Partition CF/HD
Total capacity: 497.6 MB
Number of partitions: 1
Size per sector: 512 Byte
Number of sect.: 1019088

Not formatted
Size: 433.2 MB

C: (SYSTEM)
Size: 64.4 MB
Used: 22.5 MB
Available: 41.9 MB

DRAM
Size: 64.0 MB

Available
Total: 36.3 MB
Largest available block: 36.2 MB

Used
Size: 27.7 MB

SRAM
Size: 1.0 MB

Not configured
Size: 272.0 kB

USERRAM (CP1486.SRAM)
Size: 496.0 kB
Used: 28 Byte
Available: 496.0 kB
Largest available block: 496.0 kB

REMMEM (CP1486.SRAM)
Size: 256.0 kB
Available: 251.8 kB
Largest available block: 251.8 kB

Perm.: 0 Byte
Remanent global: 4.0 kB
Remanent local: 252.0 kB

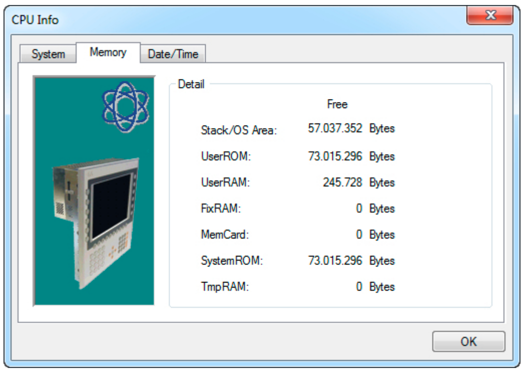
Determining memory requirements using the System Diagnostics Manager

3.2.1 Tools for determining memory requirements

The amount of free memory can be determined in the following ways:

- Reading CPU information
- Using the System Diagnostics Manager
- Using the BrSystem library during runtime

Selecting <**Online**> / <**Info**> from the menu shows general information about the amount of available memory. This option is not available for all target systems.



	Free	
Stack/OS Area:	57.037.352	Bytes
UserROM:	73.015.296	Bytes
UserRAM:	245.728	Bytes
FixRAM:	0	Bytes
MemCard:	0	Bytes
SystemROM:	73.015.296	Bytes
TmpRAM:	0	Bytes

Reading CPU information

This view also shows memory types for SG3 / SGC target systems as 0 bytes. Not available for SG4.

The System Diagnostics Manager (SDM) is an integral component of Automation Runtime V3.0 and higher. Selecting <**Tools**> / <**SystemDiagnostics**> rom the menu opens a browser window and shows the SDM diagnostics pages.

If the PC's default browser uses a proxy server, the IP addresses of the target system must be bypassed on the proxy server for SDM access.

Checking the amount of memory available on the target system

Create a CompactFlash card for one of the module systems in the project created in TM210.

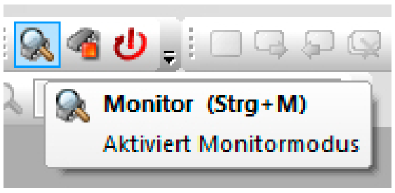
The target must be connected via an ethernet connection with Automation Studio With one of the under Chapter 3.2.1 listed possibilities the memory consumption can be estimated.

Diagnostics and service / Diagnostics tools / System Diagnostics Manager

3.2.2 Automation Studio monitor

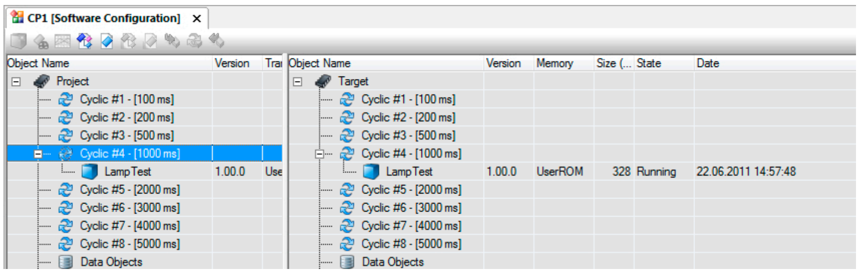
When there is an active online connection, it is possible to compare the modules configured in Automation Studio with the modules installed on the target system.

With the software configuration open, Monitor Mode can be turned on and off using the monitor icon in the toolbar.



Activating the monitor

Modules are displayed in the workspace for comparison. The "Memory" column displays the target memory area of the BR module on the target system.



CP1 [Software Configuration]

Object Name	Version	Trai
Project		
Cyclic #1 - [100 ms]		
Cyclic #2 - [200 ms]		
Cyclic #3 - [500 ms]		
Cyclic #4 - [1000 ms]		
LampTest	1.00.0	Use
Cyclic #5 - [2000 ms]		
Cyclic #6 - [3000 ms]		
Cyclic #7 - [4000 ms]		
Cyclic #8 - [5000 ms]		
Data Objects		

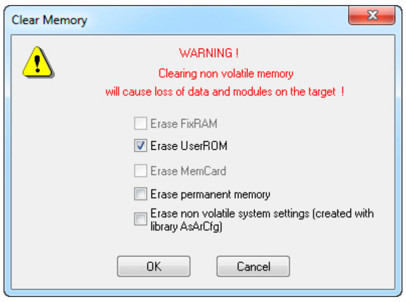
Object Name	Version	Memory	Size (...	State	Date
Target					
Cyclic #1 - [100 ms]					
Cyclic #2 - [200 ms]					
Cyclic #3 - [500 ms]					
Cyclic #4 - [1000 ms]					
LampTest	1.00.0	UserROM	328	Running	22.06.2011 14:57:48
Cyclic #5 - [2000 ms]					
Cyclic #6 - [3000 ms]					
Cyclic #7 - [4000 ms]					
Cyclic #8 - [5000 ms]					
Data Objects					

Monitor mode activated for the software configuration

Task: Delete the modules from the UserROM

Select <**Online**> / <**Services**> / <**Clear Memory**> from the menu to clear the UserROM.

Then reactivate monitor mode in the software configuration.



Clear Memory

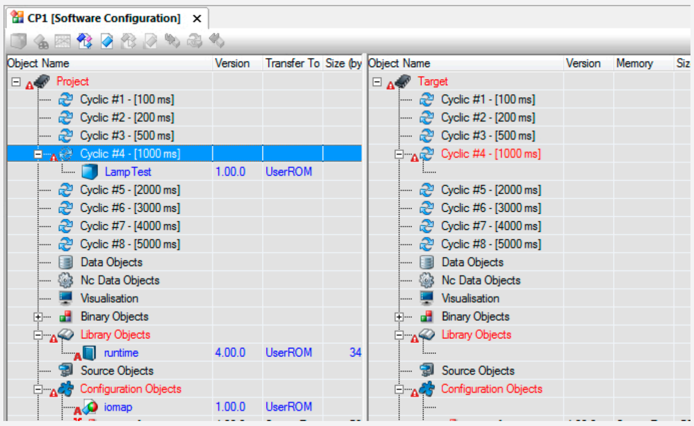
WARNING !
Clearing non volatile memory
will cause loss of data and modules on the target !

- [ ] Erase FixRAM
- [x] Erase UserROM
- [ ] Erase MemCard
- [ ] Erase permanent memory
- [ ] Erase non volatile system settings (created with library AsArCfg)

Clearing the UserROM

Only the modules that have **SystemROM** defined as their target memory are displayed.

Since the system settings such as the Ethernet configuration are stored in the "**sysconf.br**" module with **SystemROM** set as the target memory, the online connection is not lost when **UserROM** is cleared.



CP1 [Software Configuration]

Object Name	Version	Transfer To	Size (by
Project			
Cyclic #1 - [100 ms]			
Cyclic #2 - [200 ms]			
Cyclic #3 - [500 ms]			
Cyclic #4 - [1000 ms]			
LampTest	1.00.0	UserROM	
Cyclic #5 - [2000 ms]			
Cyclic #6 - [3000 ms]			
Cyclic #7 - [4000 ms]			
Cyclic #8 - [5000 ms]			
Data Objects			
Nc Data Objects			
Visualisation			
Binary Objects			
Library Objects			
runtime	4.00.0	UserROM	34
Source Objects			
Configuration Objects			
iomap	1.00.0	UserROM	

Object Name	Version	Memory	Siz
Target			
Cyclic #1 - [100 ms]			
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]			
Data Objects			
Nc Data Objects			
Visualisation			
Binary Objects			
Library Objects			
Source Objects			
Configuration Objects			

Monitor mode for the software configuration after clearing the UserROM

When UserROM is cleared, the system automatically restarts and changes to diagnostic mode.

This briefly interrupts the online connection.

4 RUNTIME PERFORMANCE

This chapter uses small programs to illustrate the runtime behavior of the Automation Studio application with respect to Automation Runtime.

Several of the diagnostics tools available in Automation Studio will be used to describe how this works.

4.1 Starting Automation Runtime

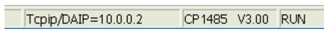
Automation Runtime boots when the target system is switched on. This involves the following tasks:

- Hardware check
- Hardware/firmware upgrade if necessary
- BR module check (safe BR module system)
- BR modules copied from ROM to DRAM
- SRAM variables copied to DRAM
- Variable memory set to initialization value
- Execution of program initialization
- Activation of cyclic programs

Real-time operating system / Functionality / Boot behavior

Real-time operating system / Functionality / Module / Data security

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



Tcpip/DAIP=10.0.0.2 CP1485 V3.00 RUN

X20CP1485 in RUN mode

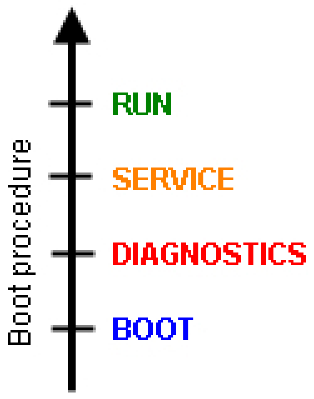
For information on diagnosing error states, consult the training module "TM223 Automation Studio Diagnostics".

Automation Runtime boot phases

The boot process can be divided into the following four phases.

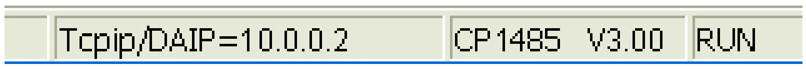
Once one phase has completed successfully, checking the next phase begins.

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



Boot phases

Some events cause the target system to terminate one of the phases and wait in the respective system status for diagnostics purposes. These are described in the following table:

SYSTEM STATUS	Conditions that lead to this system status
BOOT	• If no CompactFlash card is inserted • If there is no operating system on the CF • Node switch 00
DIAGNOSTICS	• Clearing memory • Fatal system error • Node switch FF
SERVICE	• Division by zero • Page fault • Cycle time violation • CPU halted by Automation Studio • Other errors
RUN	• No error  Tcpip/DAIP=10.0.0.2 CP1485 V3.00 RUN RUN mode

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

Automation Runtime boot phases

Before the controller is in RUN mode, additional steps may occur . These states are also shown in the status bar of Automation Studio. However, these phases are usually very short.

System state	Description
STARTUP	Operating system-related initializations are performed.
FIRMWARE	During this phase, firmware updates are performed.
INIT	Here the Init subroutines of the application are executed.

If the phases before the RUN state (STARTUP, FIRMWARE, INIT) take longer, for example the X20 system, indicates that by a green flashing R / E LED.

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

4.2 Global and local variables

Variables are symbolic programming elements whose size and structure are determined by its data type. When the project is built, variables are assigned a position in memory.

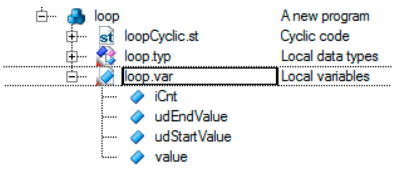
A variable's scope and properties determine its behavior during startup and runtime.

Programming / Variables and data types

4.2.1 Local variables

Local variables are defined with a local scope for a specific program, and can't be used in other programs.

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



```
loop                  A new program
  loopCyclic.st       Cyclic code
  loop.typ            Local data types
  loop.var            Local variables
    iCnt
    udEndValue
    udStartValue
    value
```

Local variables from "Loop" program

Create a "Loop" program using local variables

In the logical view of the project, create a new Structured Text program with the following local variables and name it "Loop".

Add three variables named "i", "udiCounter" and "udiEndValue" with the data type UDINT.

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

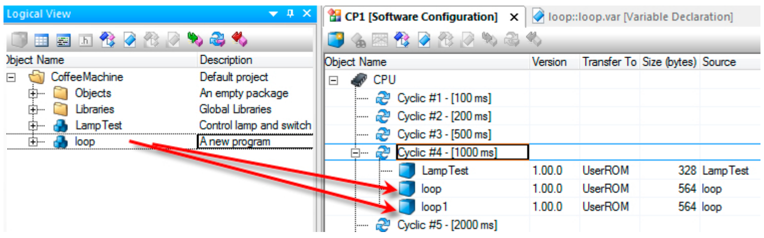
```
PROGRAM_CYCLIC

FOR i := 0 TO udiEndValue DO

udiCounter := udiCounter + 1;

END_FOR

END_PROGRAM
```

1) Add a new program and name it "Loop"
2) Select "Structured Text" as the programming language
3) Open the file "Loop.var" and insert the variables
4) After saving the file "Loop.var" the variables in "LoopCyclic.st" can be used in the task program.

The program is automatically added to the software configuration when it is inserted.

Multiple assignment of a program in the software configuration

After completing the last task, drag and drop the same program a second time from the logical view into the software configuration.


Logical View
CP1 [Software Configuration]
loop::loop.var [Variable Declaration]

Object Name	Description
CoffeeMachine	Default project
Objects	An empty package
Libraries	Global Libraries
LampTest	Control lamp and switch
loop	A new program

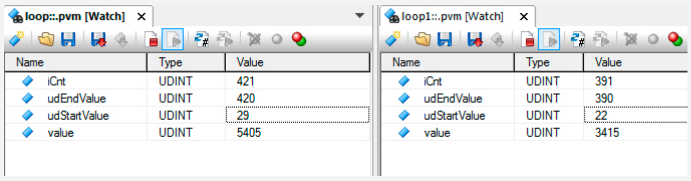
Object Name	Version	Transfer To	Size (bytes)	Source
CPU				
Cyclic #1 - [100 ms]				
Cyclic #2 - [200 ms]				
Cyclic #3 - [500 ms]				
Cyclic #4 - [1000 ms]				
LampTest	1.00.0	UserROM	328	LampTest
loop	1.00.0	UserROM	564	loop
loop1	1.00.0	UserROM	564	loop
Cyclic #5 - [2000 ms]				

Multiple assignment of programs

1) Open the software configuration
2) Switch to the logical view in the project explorer
3) Drag and drop the "Loop" program into the software configuration

Once all the variables are inserted in the variable monitor for "Loop" and "Loop1" a separate variable value is displayed in each task that is only valid for that task.

Changing a local variable in one task only affects that task – the local variable values for the other task don't change.



Name	Type	Value
iCnt	UDINT	421
udEndValue	UDINT	420
udStartValue	UDINT	29
value	UDINT	5405

Name	Type	Value
iCnt	UDINT	391
udEndValue	UDINT	390
udStartValue	UDINT	22
value	UDINT	3415

Variable monitor for both tasks

Variables that are contained 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.

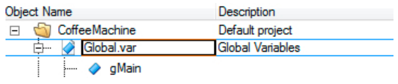
warning 6424: Variable <variable name> is declared but not used in the current configuration

Programming / Variables and data types / Scope of declarations

4.2.2 Global / Package global variables

Global variables at the highest level of the Automation Studio project are visible throughout the project. They can therefore be used in any program at any level of a package.

A global variable is managed at the top level in the file Global.var.



Object Name	Description
CoffeeMachine	Default project
Global.var	Global Variables
gMain	

Global variables

Package global variables declared within a package are only visible within the respective package and in all subordinate packages.

Global variables should only be used when multiple programs need to exchange data.

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

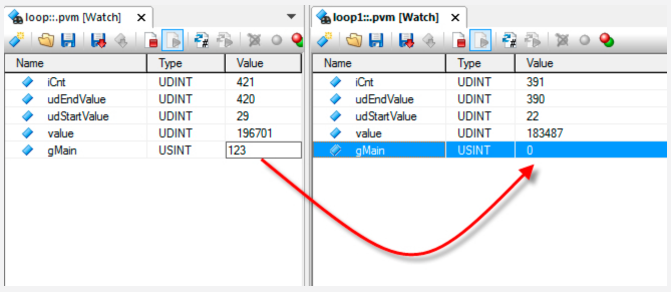
Enter a new variable in the global variable declaration "**Global.var**" and name it "**gMain**" with the data type UDINT.

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

gMain := gMain + 1;

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

If the variable **gMain** is inserted in the variable monitor for the tasks "**Loop**" and "**Loop1**", writing it with a value in one of the tasks also immediately changes its value in the other task.



loop::.pvm [Watch]

Name	Type	Value
iCnt	UDINT	421
udEndValue	UDINT	420
udStartValue	UDINT	29
value	UDINT	196701
gMain	USINT	123

loop1::.pvm [Watch]

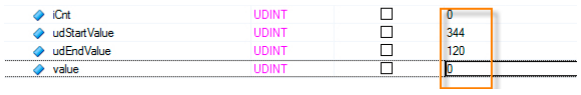
Name	Type	Value
iCnt	UDINT	391
udEndValue	UDINT	390
udStartValue	UDINT	22
value	UDINT	183487
gMain	USINT	0

Writing global variables

Programming / Variables and data types / Scope of declarations

4.2.3 Initializing the variable memory

By default, variables are written with "0" during initialization. However, a different initialization value can be specified in the variable declaration.



iCnt	UDINT	☐	0
udStartValue	UDINT	☐	344
udEndValue	UDINT	☐	120
value	UDINT	☐	0

Variable initialization in "Loop.var"

This initialization is the equivalent of the following lines of code in the initialization program "**LoopInit.st**":

PROGRAM _INIT

```
PROGRAM _INIT
udiEndValue := 50;
END_PROGRAM
```

Initialize the variable "udiEndValue"

Configure the variable "**udiEndValue**" in the program "**Loop**" with an initial value of 50. Monitor the value in the variable monitor.

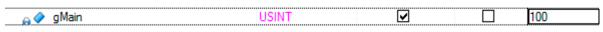
1) Open the variable declaration "**Loop.var**"

2) Set the value in the "**Value**" column for the variable "**udiEndValue**"

In the variable monitor for the "**Loop**" and "**Loop1**" tasks, the variable "**udiEndValue**" is initialized with the value 50. During runtime the value can be changed to any other value.

4.2.4 Constants

Constants are variables whose values never change while the program is running.



gMain	USINT	☑	☐	100

Declaration of constants

Global variable "gMain" as a constant

Configure the global variable "**gMain**" as a constant with a value of 50.

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

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

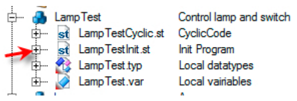
This is not permitted for constants. In order for the program to operate without errors, the variable "**gMain**" must not be defined as a constant.

LoopCyclic.st (Ln: 19, Col: 8) : error 1138: No write access to variable 'gMain' allowed.

Programming / Variables and data types / Variables / Constants

4.3 Program initialization

Each program can contain an initialization program (first scan).



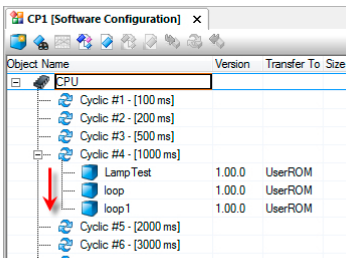
LampTest	Control lamp and switch
LampTestCyclic.st	CyclicCode
LampTestInit.st	Init Program
LampTest.typ	Local datatypes
LampTest.var	Local vairiables

Program initialization

An initialization program initializes variables and pointers and reads PLC data and machine configurations that are to be executed in the cyclic program.

4.3.1 Running the initialization program

Before the first cyclic program is started, all initialization programs are executed **once** in the order specified in the software configuration.



CP1 [Software Configuration]

Object Name	Version	Transfer To	Size
CPU			
Cyclic #1 - [100 ms]			
Cyclic #2 - [200 ms]			
Cyclic #3 - [500 ms]			
Cyclic #4 - [1000 ms]			
LampTest	1.00.0	UserROM	
loop	1.00.0	UserROM	
loop1	1.00.0	UserROM	
Cyclic #5 - [2000 ms]			
Cyclic #6 - [3000 ms]			

Execution of the initialization program

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

1. Initialization program for the "LampTest" task
2. Initialization program for the "Loop" task
3. Initialization program for the "Loop1" task

There is no cycle time monitoring for the initialization program, so no cycle time violation is triggered by long initializations.

Real-time operating system / Target systems / SG4 / Runtime behavior / Starting initialization SPs

4.3.2 Function calls in the initialization program

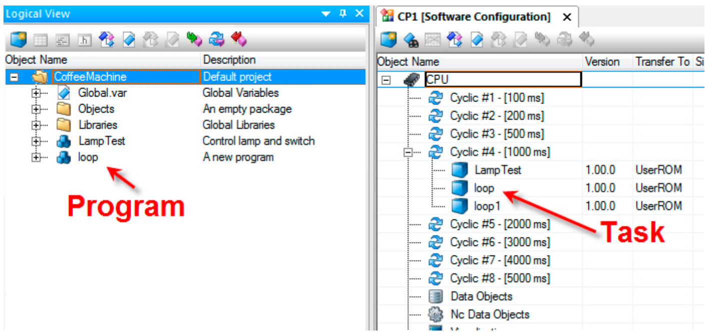
When using functions, ensure that the function call returns a result.

Functions that require multiple cycles must be called in the cyclic section of the program.

4.4 Cyclic programs

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

Programs in the software configuration are called tasks.



Object Name	Description
CoffeeMachine	Default project
Global.var	Global Variables
Objects	An empty package
Libraries	Global Libraries
LampTest	Control lamp and switch
loop	A new program

Object Name	Version	Transfer To
CPU		
Cyclic #1 - [100 ms]		
Cyclic #2 - [200 ms]		
Cyclic #3 - [500 ms]		
Cyclic #4 - [1000 ms]		
LampTest	1.00.0	UserROM
loop	1.00.0	UserROM
loop1	1.00.0	UserROM
Cyclic #5 - [2000 ms]		
Cyclic #6 - [3000 ms]		
Cyclic #7 - [4000 ms]		
Cyclic #8 - [5000 ms]		
Data Objects		
Nc Data Objects		

Relationship between program and task

4.4.1 Multitasking operating system

The Automation Runtime is a **deterministic realtime multitasking** operating system.

Each task is assigned **time slices** which give them ordered access to required resources such as I/O channels.

4.4.2 Task class cycle time

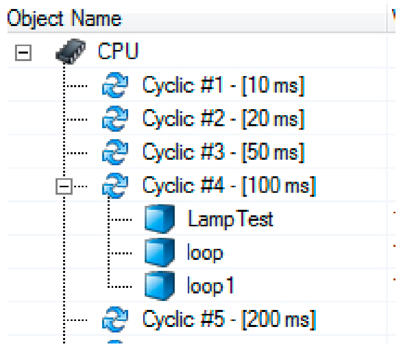
A task is executed cyclically in the time defined by its task class, also known as its cycle time.

Up to 8 task classes (SG4) with defined, configurable cycle times are provided to help optimize a task for its particular requirements. A task class contains tasks with the same cycle time, priority and tolerance.

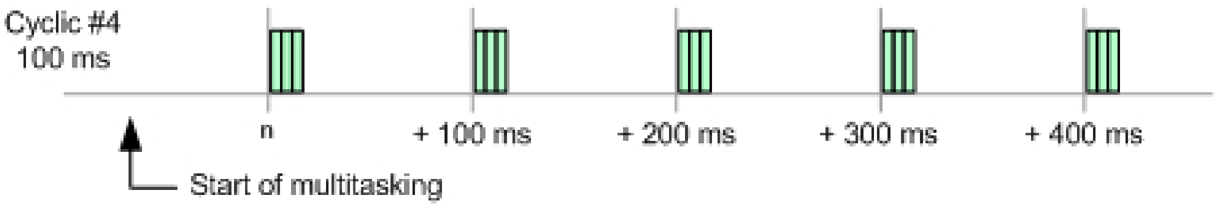
Real-time operating system / Target systems / SG4 / Runtime behavior / Starting cyclic tasks

Not all tasks have to run within the same timeframe. Control tasks that must be executed quickly should be assigned a shorter task class with a lower number. Slower processes should be assigned task classes with higher numbers.

This example contains three tasks in task class #4 with a cycle time of 100 milliseconds. Ignoring the time it takes to execute each of these tasks, the program cycle would look like this:


Object Name
CPU
Cyclic #1 - [10 ms]
Cyclic #2 - [20 ms]
Cyclic #3 - [50 ms]
Cyclic #4 - [100 ms]
LampTest
loop
loop1
Cyclic #5 - [200 ms]

Cycle time



The tasks will be called in every cycle.

This means that the tasks in this task class are executed every 100 milliseconds.

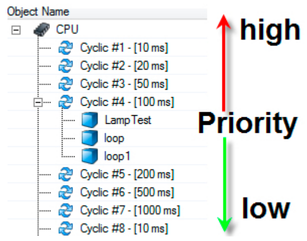
4.4.3 Cycle time and priority

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

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

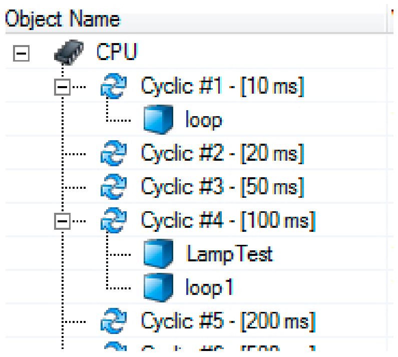
Moving a task between task classes changes its priority and cycle time.

Moving a task does not change its execution time. It does change the number of times it is called in a given period of time.


Object Name
CPU
Cyclic #1 - [10 ms]
Cyclic #2 - [20 ms]
Cyclic #3 - [50 ms]
Cyclic #4 - [100 ms]
LampTest
loop
loop1
Cyclic #5 - [200 ms]
Cyclic #6 - [500 ms]
Cyclic #7 - [1000 ms]
Cyclic #8 - [10 ms]

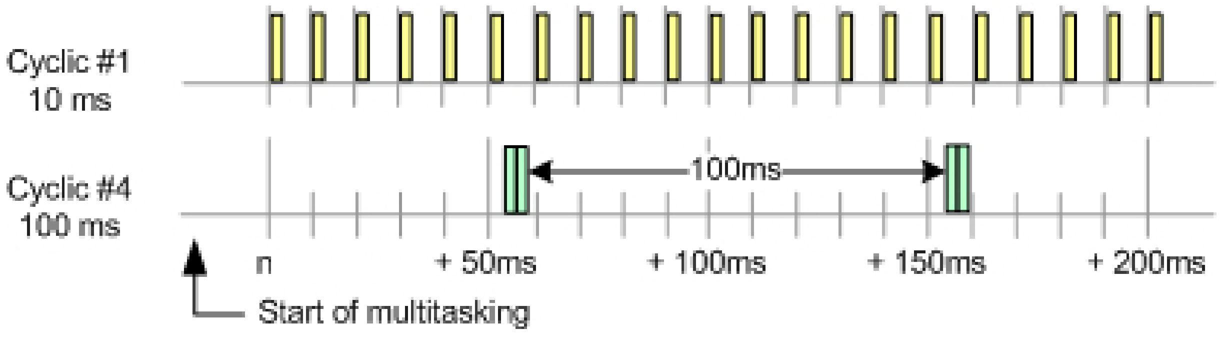
Task class priority

For example, if the "**Loop**" task is moved from tack class #4 to task class #1, it will be executed every 10 milliseconds.



Object Name
CPU
Cyclic #1 - [10 ms]
loop
Cyclic #2 - [20 ms]
Cyclic #3 - [50 ms]
Cyclic #4 - [100 ms]
LampTest
loop1
Cyclic #5 - [200 ms]

Different task classes



Executing tasks in different taskclasses

The tasks "**LampTest**" and "**Loop1**" in task class #4 are still executed every 100 milliseconds.

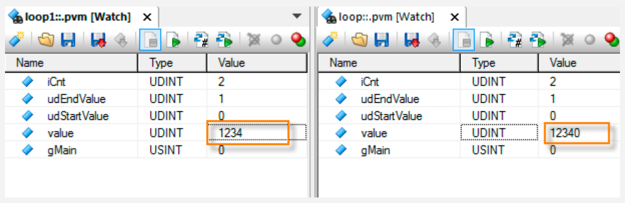
Move the "Loop" task to task class #1

In the software configuration, move the "**Loop**" task to **10[ms] task class #1**.

Observe the changes to the cycle times of "**Loop**" and "**Loop1**", which reference the same program.

1) Open the software configuration.
2) Drag and drop the "**Loop**" task from task class #4 to task class #1.
3) Open the variable monitor for both tasks.
4) Monitor the "**udiCounter**" variable.

The "Loop" program is executed 10 times as frequently as "Loop1". As a result, the value of the "udiCounter" is increased more often.



loop1::.pvm [Watch]

Name	Type	Value
iCnt	UDINT	2
udEndValue	UDINT	1
udStartValue	UDINT	0
value	UDINT	1234
gMain	USINT	0

loop::.pvm [Watch]

Name	Type	Value
iCnt	UDINT	2
udEndValue	UDINT	1
udStartValue	UDINT	0
value	UDINT	12340
gMain	USINT	0

Cycle times in different task classes

4.4.4 Starting task classes

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

The start time is always task class cycle / 2.

This means that for example, the 100 ms task class will be started after 50 ms. Distributing the start time of all task classes can shorten execution times and keep output jitter to a minimum despite a high load on the processor.

Real-time operating system / Target systems / SG4 / Runtime behavior / Starting cyclic tasks

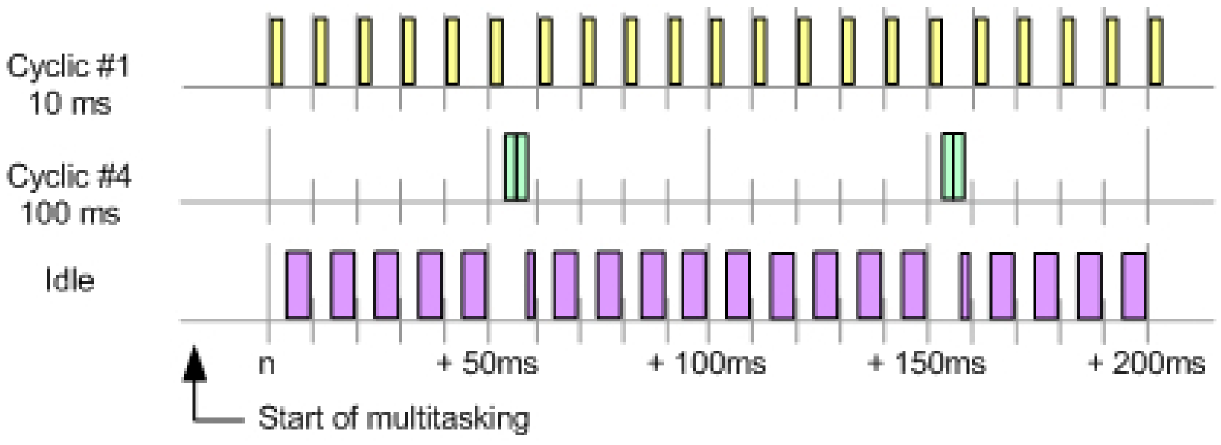
4.4.5 Idle time

During operation, Automation Runtime performs many other **cyclic** system tasks in addition to the tasks described above. These provide the user convenience and security (e.g. module verification, online communication).

The speed at which these tasks are executed can vary depending on the performance of the CPU being used.

The time when no Automation Runtime or user tasks are being executed is referred to as **idle time**. Automation Runtime makes this idle time available to other processes for the following tasks:

- Online communication
- Visual Components application
- File access



Idle time

The time that remains after subtracting the runtime of the idle time processes is the idle time.

If the idle time is not sufficient to run a Visual Components application or provide a stable online connection, the amount of idle time can be increased by assigning programs to a higher task class or by adjusting the task class cycle time.

The **Profiler** can be used to determine the idle time.

Project Management / Configuration View / System configuration / System configuration SG4 / CPU / Timing

4.4.6 I/O scheduler

The I/O scheduler starts the task class at a precise time and provides consistent I/O mapping for the execution of all the tasks in a task class.

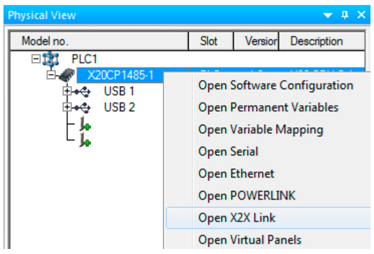
This ensures that the inputs at the beginning of the task class and the outputs at the end of the task class are transferred promptly to the mapped process variables.

Each task class uses a separate I/O mapping. The input states don't change during the task's runtime, and the output states are not written until the tasks in a task class are finished.

Cycle for I/O data mapping

The I/O channels are mapped in the cycle of the I/O fieldbus system.

This cycle can be configured in the fieldbus properties (e.g. X2X Link) by right clicking on the CPU node and selecting <**Open X2X Link**> from the shortcut menu.



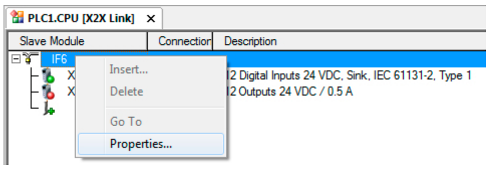
Physical View

Model no.	Slot	Version	Description
PLC1			
X20CP1485-1			
USB 1			
USB 2			

Open Software Configuration
Open Permanent Variables
Open Variable Mapping
Open Serial
Open Ethernet
Open POWERLINK
Open X2X Link
Open Virtual Panels

Opening X2X

The properties of the X2X interface can be opened by selecting **Properties** from the shortcut menu.



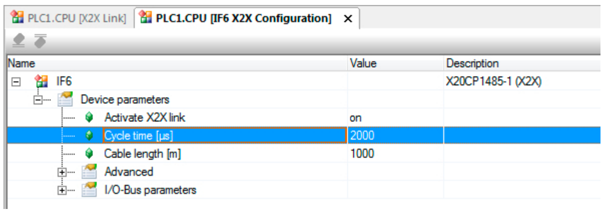
PLC1.CPU [X2X Link]

Slave Module	Connection	Description
IF6		
X		2 Digital Inputs 24 VDC, Sink, IEC 61131-2, Type 1
X		2 Outputs 24 VDC / 0.5 A

Insert...
Delete
Go To
Properties...

X2X Link Editor

The cycle time configured in the properties is the basis for copying the I/O data to the I/O mapping.

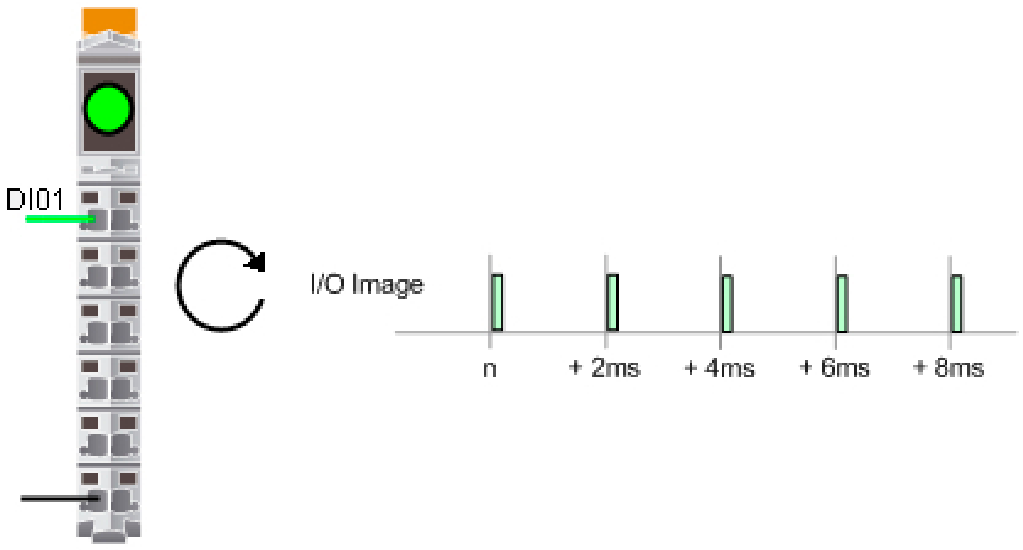


PLC1.CPU [X2X Link] | PLC1.CPU [IF6 X2X Configuration]

Name	Value	Description
IF6		X20CP1485-1 (X2X)
Device parameters		
Activate X2X link	on	
Cycle time [µs]	2000	
Cable length [m]	1000	
Advanced		
I/O-Bus parameters		

X2X cycle time setting

This means that the fieldbus device (X2X Link in this case) copies the inputs to the I/O mapping and the outputs from the I/O mapping within the configured cycle time.



Reading input mapping

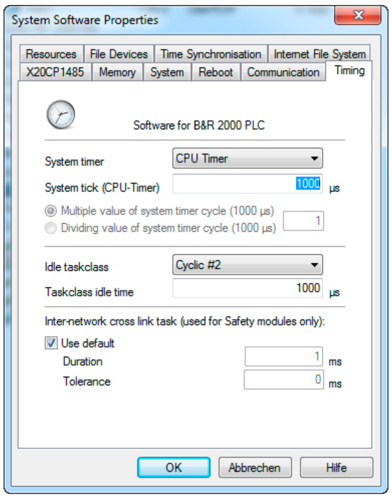
For the application, this means that the input data is not older than the configured cycle time after the start of the task class, and the output data will be written using this mapping after this time has passed at the latest.

I/O data in the task class

The I/O scheduler controls when the I/O data 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 µs.

This timing can be configured in the system software properties.


System Software Properties

Resources | File Devices | Time Synchronisation | Internet File System | X20CP1485 | Memory | System | Reboot | Communication | Timing

Software for B&R 2000 PLC

System timer: CPU Timer

System tick (CPU-Timer): 1000 µs

(●) Multiple value of system timer cycle (1000 µs): 1

( ) Dividing value of system timer cycle (1000 µs)

Idle taskclass: Cyclic #2

Taskclass idle time: 1000 µs

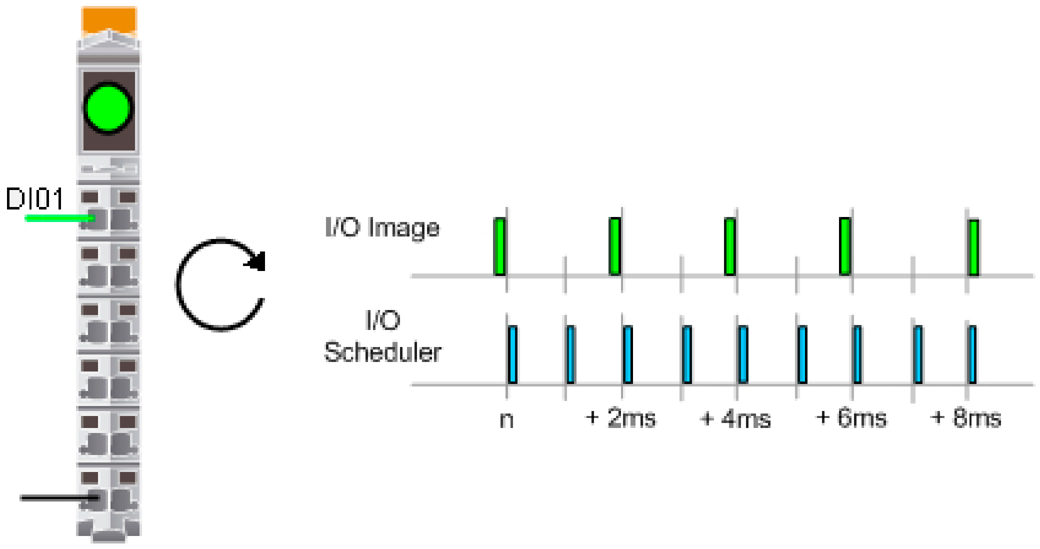
Inter-network cross link task (used for Safety modules only):

[✓] Use default

Duration: 1 ms

Tolerance: 0 ms

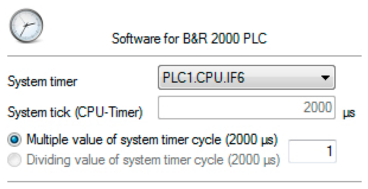
Configurable system tick



Reading input mapping

In this case, the I/O scheduler is called twice as often as the I/O mapping is updated.

The I/O scheduler runs at a whole-number ratio synchronous to the I/O cycle of the fieldbus. The timer and the ratio can be set as needed.


Software for B&R 2000 PLC

System timer: PLC1.CPU.IF6

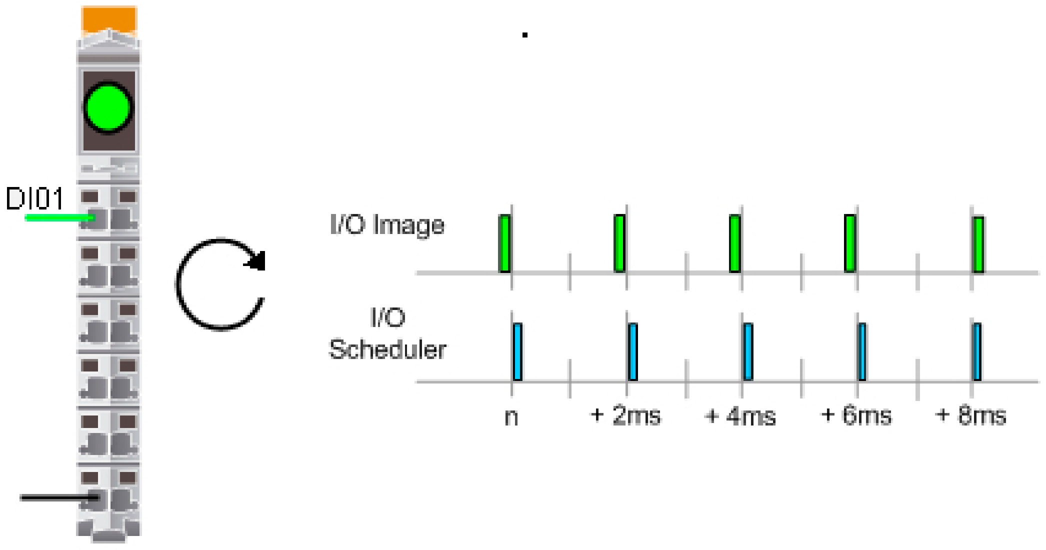
System tick (CPU-Timer): 2000 µs

(●) Multiple value of system timer cycle (2000 µs): 1

( ) Dividing value of system timer cycle (2000 µs)

Synchronizing the system tick

Setting the system timer to the fieldbus timer (X2X or POWERLINK) with a 1:1 ratio will cause the I/O Scheduler to be called with the configured I/O cycle time.



Reading input mapping

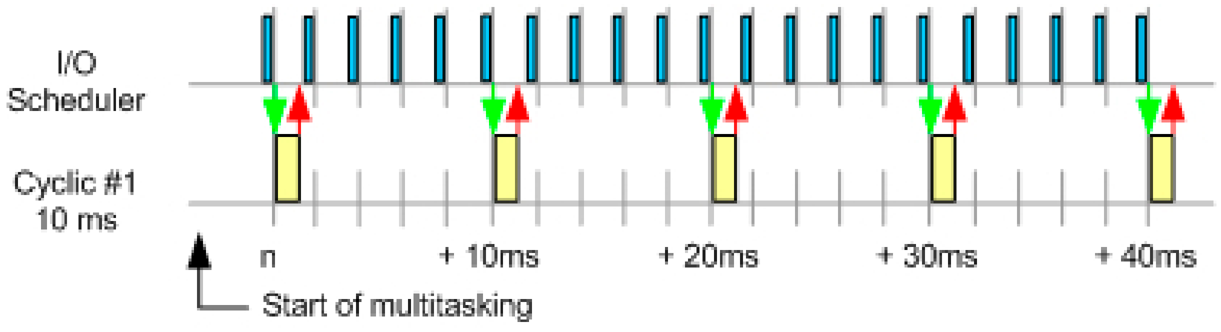
With a POWERLINK fieldbus, an I/O cycle time 400 µs can be achieved. Together with an APC and a suitable X20 CPU, an I/O cycle time of 200 µs can be achieved.

I/O management (see 4.5) assigns I/O data from the mapping to the variables.

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

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

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

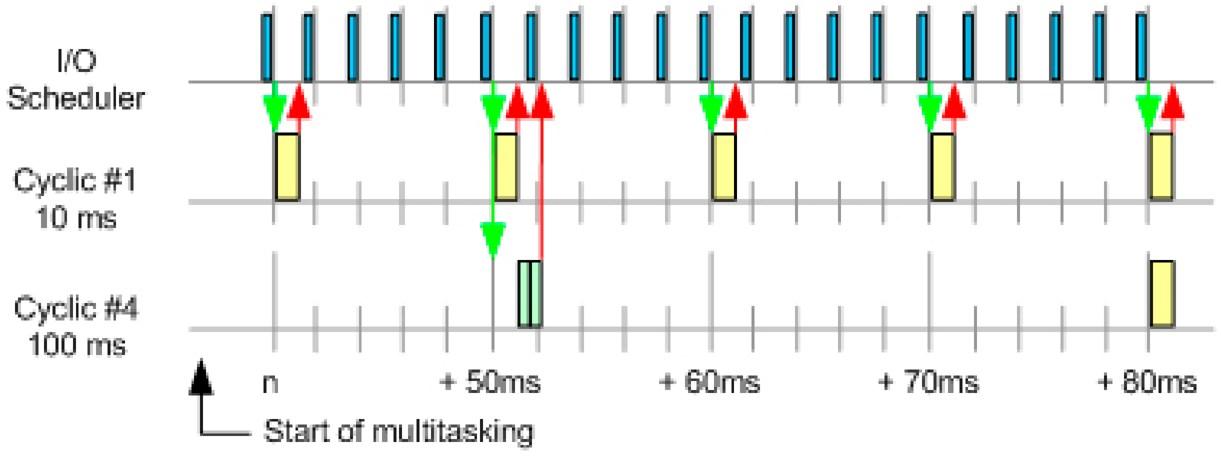


I/O handling for task class #1

Task class #4

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

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



I/O handling for task class #4

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

4.4.7 Cycle time and tolerance

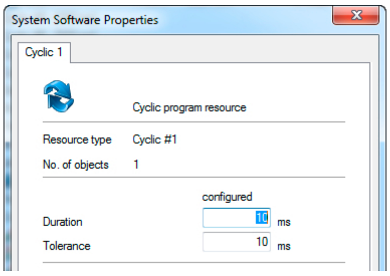
Each task class has a **predefined** cycle time. All of the tasks in a task class must be executed within this cycle time.

The cycle time of a task class is defined when a project is created and can be changed by the user later according to the 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 prevents the system from booting in service mode when the cycle time is exceeded.

If the configured runtime for a task class is exceeded, the task class doesn't start again until the beginning of its next cycle. This can lead to timing problems in the application.

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



Resource type	Cyclic #1
No. of objects	1
Duration	10 ms
Tolerance	10 ms

Configuring the cycle time and tolerance

Project Management / Configuration View / Systemkonfiguration / Systemkonfiguration SG4 / Cyclic / Cyclic n

Cause a cycle time violation

In "**Loop**", which runs in the 10[ms] task class #1, change the value of the variable "**udiEndValue**" in the variable monitor to cause a cycle time violation.

Use the **Profiler** to monitor the runtime behavior of the task and the available idle time. After causing the cycle time violation, use the Automation Studio **Logger** to examine the cause.

Step 1

- In the first step, set the value of the variable "udiEndValue" to 50,000.

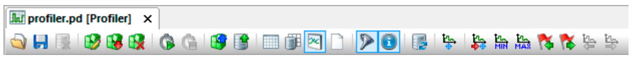
The current execution time can be determined using the Profiler. This process is described in the next step.

Profiler for determining execution times

Open the Profiler from the software configuration by selecting <**Open**> / <**Profiler**> rom the shortcut menu.

While performing this task, it is recommended to have the **variable monitor** for the "**Loop**" task, the **software configuration** and the **Profiler** open at all times.

The task can be changed by selecting the corresponding workbook.



Changing the Profiler view

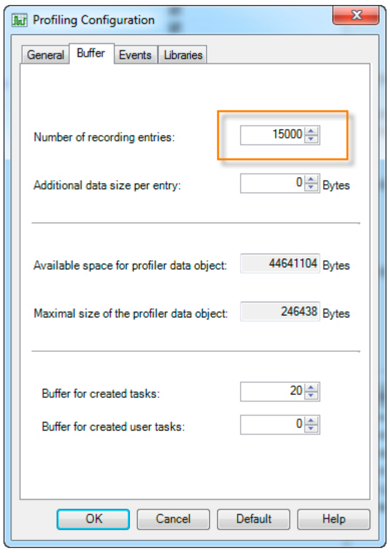
By default, Automation Runtime records all runtime behavior.

For this task, only the execution times of the user tasks, the task classes and the exceptions are recorded.

In order to edit the configuration, you must pause the current recording using the "**Stop**" icon in the Profiler toolbar.

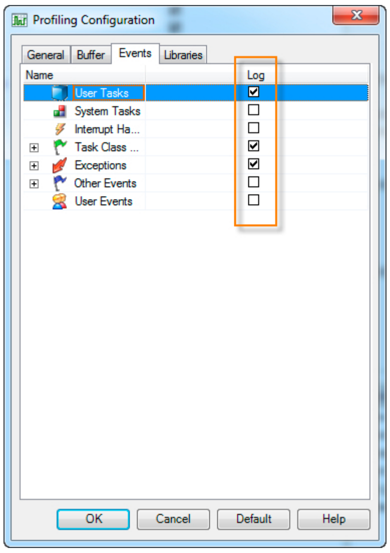
To open the dialog box for making configurations, click the „**Configuration**“ icon in the Profiler toolbar.

Make the configurations shown in the following image:



Number of recording entries:	15000	
Additional data size per entry:	0	Bytes
Available space for profiler data object:	44641104	Bytes
Maximal size of the profiler data object:	246438	Bytes
Buffer for created tasks:	20	
Buffer for created user tasks:	0	

Profiler Configuration 1



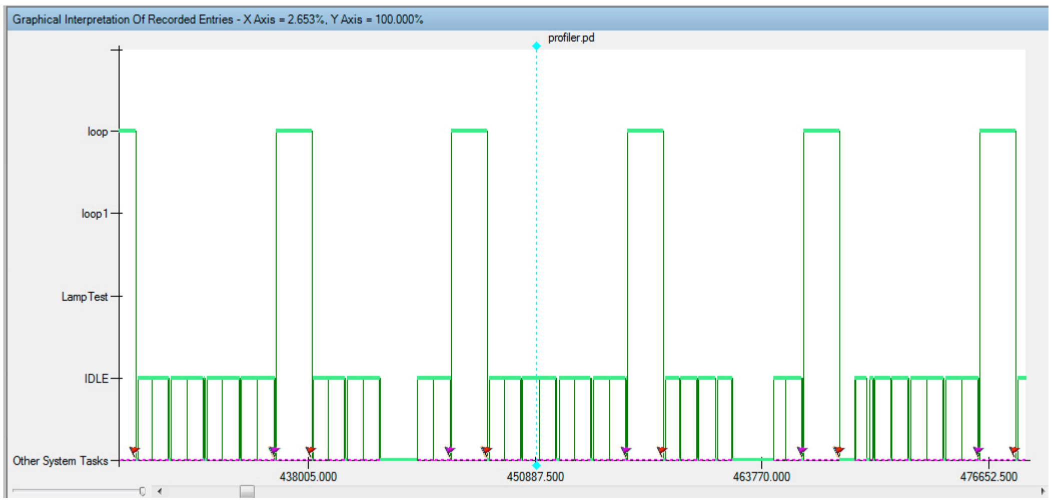
Name	Log
User Tasks	☑
System Tasks	☐
Interrupt Ha...	☐
Task Class ...	☑
Exceptions	☑
Other Events	☐
User Events	☐

Profiler Configuration 2

Transfer your changes to the target system using the "Install" icon in the Profiler toolbar „**Install**“ icon in the Profiler toolbar. Recording Recording begins again immediately with the new configuration.

Pause the recording again using the „**Stop**“ icon and then click the „**Upload Data**“ icon in the Profiler toolbar to load the recording into Automation Studio, where it can be displayed by clicking on the „**Graphic**“ icon.

Depending on the starting point, the recording should look something like this:



Result of the first Profiler measurement

Check the Profiler recording

Evaluate the results of the Profiler recording using the Automation Studio Help system documentation or the training manual TM223 Automation Studio Diagnostics as a guide.

Diagnostics and service / Diagnostics tools / Profiler

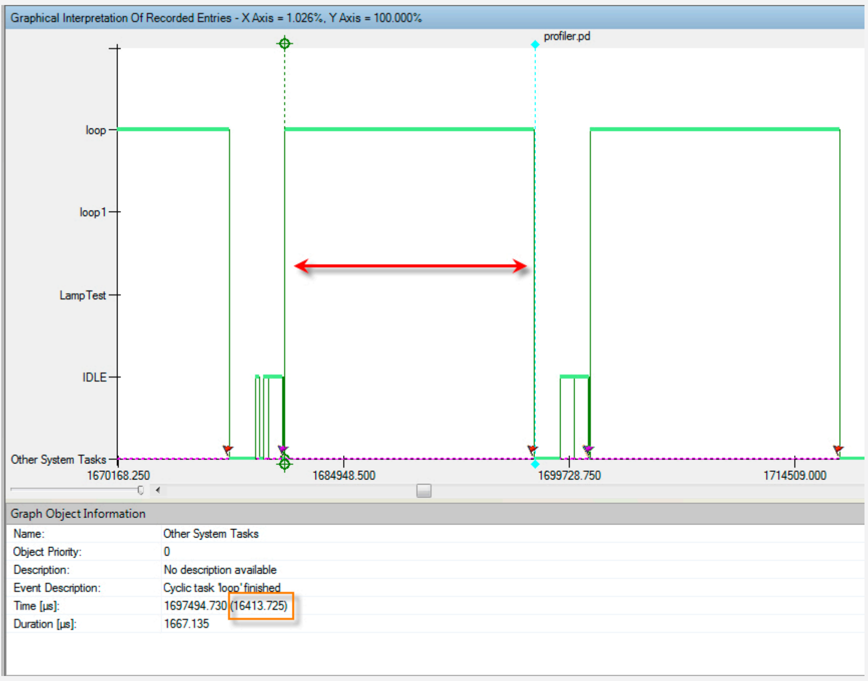
Step 2:

- In this step, start the Profiler using the "Start" icon in the Profiler toolbar.
- Increase the value of the variable "**udiEndValue**" in steps (e.g. 10000).
- Use the Profiler to monitor the execution time of the "**Loop**" task.

To determine the exact execution time, you can set a reference cursor at the beginning of "**Loop**" using the icon in the Profiler toolbar and then set the cursor at the end of "**Loop**" to see the time.

The "**Loop**" task operates in a 10 millisecond task class. According to this image, a cycle time violation must have occurred since the execution time is already 16.5 milliseconds.

This is where the tolerance takes effect, which is defined in the properties of task class #1 as 10 milliseconds.



Graph Object Information

Name:	Other System Tasks
Object Priority:	0
Description:	No description available
Event Description:	Cyclic task 'loop' finished
Time [µs]:	1697494.730 (16413.725)
Duration [µs]:	1667.135

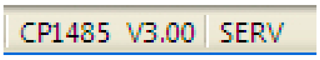
Exceeding the cycle time and tolerance

Step 3:

- Increase the value of the variable "udiEndValue" until a cycle time violation occurs.

4.4.8 Cycle time violation

When Automation Runtime detects a cycle time violation, the target system boots in service mode.



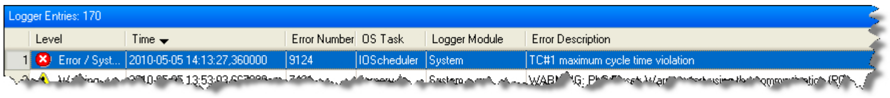
X20CP1485 in Service mode

In this example, the cycle time violation is the result of changing values in the variable monitor. When a cycle time violation occurs, all values in the variable monitor are "frozen" and displayed as 0 after restarting in service mode.

To analyze why the system rebooted in service mode, use the Automation Studio **Logger**.

Diagnostics and service / Diagnostic tools / Logger

Open the Logger from the **Software Configuration** by selecting <**Open**> / <**Logger**> from the shortcut menu.



Logger Entries: 170

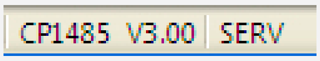
	Level	Time	Error Number	OS Task	Logger Module	Error Description
1	Error / Syst...	2010-05-05 14:13:27,360000	9124	IOScheduler	System	TC#1 maximum cycle time violation

Analyzing the cause of the error in Logger

The cause of the error status is listed as a 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 manual TM223 – Automation Studio Diagnostics.

When a cycle time violation occurs, the controller boots in service mode.



CP1485 V3.00 SERV

X20CP1485 in Service mode

The Logger can be used to determine the cause of the error.

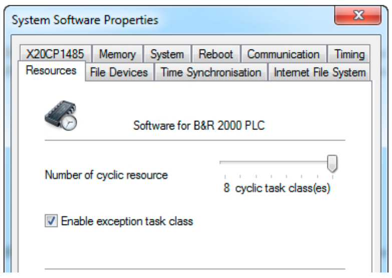
When the target system is restarted either by turning the power OFF/ON or by performing a warm restart in Automation Studio, it boots in run mode.

4.4.9 Responding to a cycle time violation in the application program

A production system should not switch to service mode when the cycle time is rarely exceeded.

It is possible to respond to exceptions in an exception program.

The exception task class can be enabled in the properties of the software configuration under the **Resources** tab.



System Software Properties

X20CP1485 | Memory | System | Reboot | Communication | Timing

Resources | File Devices | Time Synchronisation | Internet File System

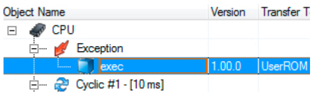
Software for B&R 2000 PLC

Number of cyclic resource: 8 cyclic task class(es)

[✓] Enable exception task class

Enabling the exception task class

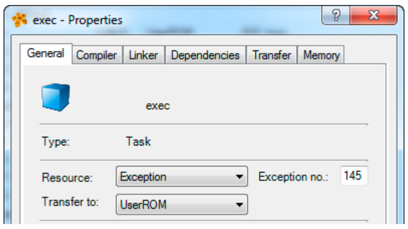
A program in the logical view can be added to the exception task class in the software configuration.



Object Name	Version	Transfer T
CPU		
Exception		
exec	1.00.0	UserROM
Cyclic #1 - [10 ms]		

Configuring the exception task class

An exception number in the properties of the exception task defines which exception triggers this task to be called.


exec - Properties

General | Compiler | Linker | Dependencies | Transfer | Memory

exec

Type: Task

Resource: Exception Exception no.: 145

Transfer to: UserROM

Configuring the exception task

If a cycle time violation (Exception no. 145) occurs during runtime, this task is called (which contains the response from the application).

Consult the Help system documentation for possible exceptions

Real-time operating system / Target systems / SG4 / Runtime behavior / Exceptions

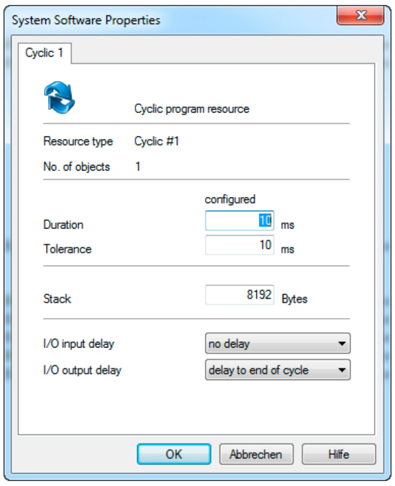
4.5 I/O management

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

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

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

Task class #1 can be configured for jitter-free responses from outputs at the end of a task class.


System Software Properties

Cyclic 1

Cyclic program resource

Resource type: Cyclic #1

No. of objects: 1

configured

Duration: 10 ms

Tolerance: 10 ms

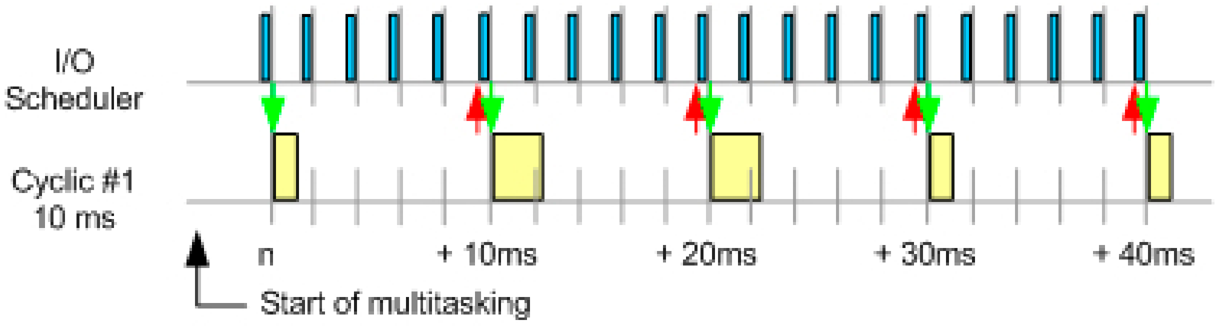
Stack: 8192 Bytes

I/O input delay: no delay

I/O output delay: delay to end of cycle

OK | Abbrechen | Hilfe

Configuration for executing output data



Jitter-free writing of the output map

The I/O manager is controlled by the I/O configuration and the I/O mapping.

Real-time operating system / Target systems / SG4 / I/O Management Real-time operating system / Target systems / SG4 / I/O Management / Functionality / Shoveling output data

4.5.1 Startup

When the controller is booted, the **active I/O configuration** is transferred to the I/O modules, and the interfaces and fieldbus devices are initialized.

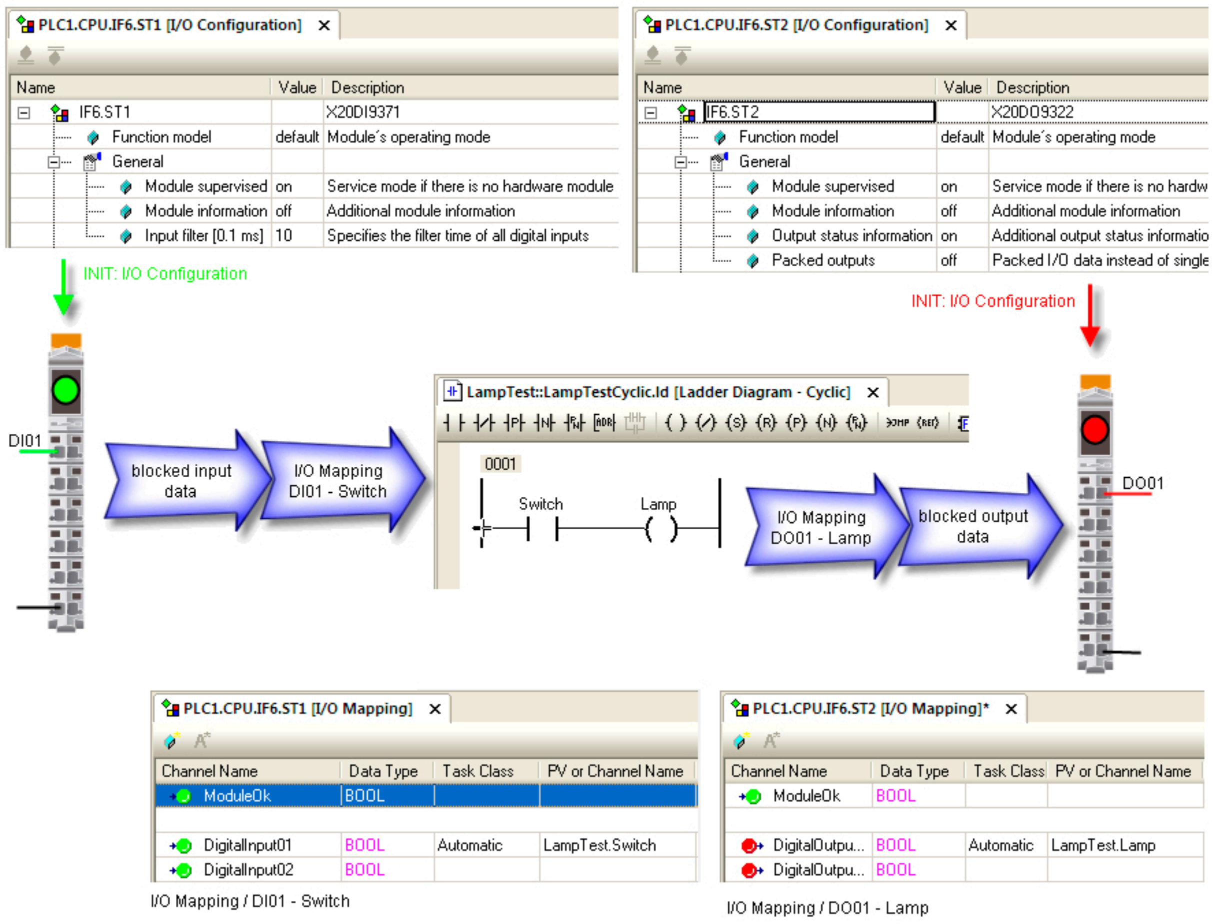
This ensures that valid I/O data is accessed in the initialization program.

4.5.2 Mapping I/O data

Blocked input data from the I/O terminals is stored in memory. The **I/O Mapping** is used to map the data to variables.

Output data is written in reverse order via the blocked output data.

The following image illustrates the relationship between the I/O configuration, which is transferred to the module during start up, and the I/O mapping, which assigns the I/O data to the corresponding variables.


PLC1.CPU.IF6.ST1 [I/O Configuration]
PLC1.CPU.IF6.ST2 [I/O Configuration]
INIT: I/O Configuration
INIT: I/O Configuration
DI01
blocked input data
I/O Mapping DI01 - Switch
LampTest::LampTestCyclic.ld [Ladder Diagram - Cyclic]
0001
Switch
Lamp
I/O Mapping DO01 - Lamp
blocked output data
DO01
PLC1.CPU.IF6.ST1 [I/O Mapping]
PLC1.CPU.IF6.ST2 [I/O Mapping]*
I/O Mapping / DI01 - Switch
I/O Mapping / DO01 - Lamp

Name	Value	Description
IF6.ST1		X20DI9371
Function model	default	Module´s operating mode
General		
Module supervised	on	Service mode if there is no hardware module
Module information	off	Additional module information
Input filter [0.1 ms]	10	Specifies the filter time of all digital inputs

Name	Value	Description
IF6.ST2		X20DO9322
Function model	default	Module´s operating mode
General		
Module supervised	on	Service mode if there is no hardw
Module information	off	Additional module information
Output status information	on	Additional output status informatio
Packed outputs	off	Packed I/O data instead of single

Channel Name	Data Type	Task Class	PV or Channel Name
ModuleOk	BOOL		
DigitalInput01	BOOL	Automatic	LampTest.Switch
DigitalInput02	BOOL		

Channel Name	Data Type	Task Class	PV or Channel Name
ModuleOk	BOOL		
DigitalOutpu...	BOOL	Automatic	LampTest.Lamp
DigitalOutpu...	BOOL		

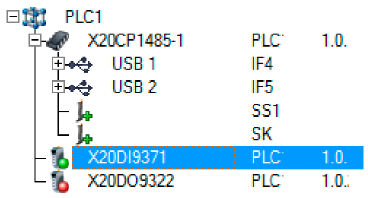
Basic I/O functionality

4.5.3 I/O mapping

The **I/O mapping** defines which I/O data is assigned to a variable.

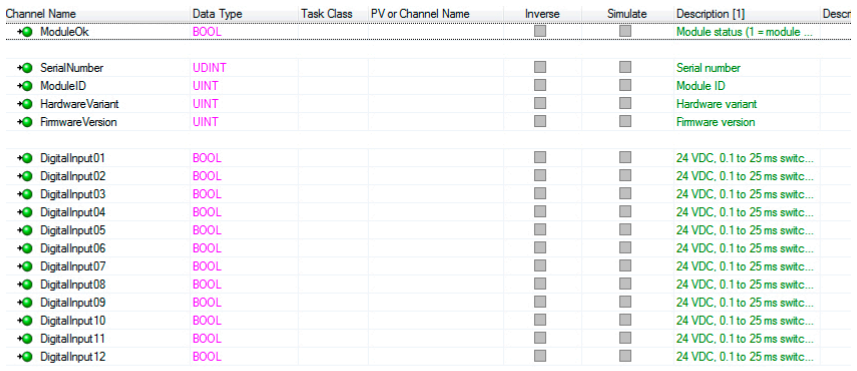
Each variable in a .var file that is used in a program can be assigned to a channel of an I/O module, regardless of its scope.

Variables can be assigned to the desired I/O module in the **physical view** or by selecting <**Open I/O Mapping**> from the shortcut menu.



Open I/O Mapping

A variable is assigned to the respective channel in the I/O mapping editor for the selected module.



Channel Name	Data Type	Task Class	PV or Channel Name	Inverse	Simulate	Description [1]	Descr
ModuleOk	BOOL			☐	☐	Module status (1 = module ...	
SerialNumber	UDINT			☐	☐	Serial number	
ModuleID	UINT			☐	☐	Module ID	
HardwareVariant	UINT			☐	☐	Hardware variant	
FirmwareVersion	UINT			☐	☐	Firmware version	
DigitalInput01	BOOL			☐	☐	24 VDC, 0.1 to 25 ms switc...	
DigitalInput02	BOOL			☐	☐	24 VDC, 0.1 to 25 ms switc...	
DigitalInput03	BOOL			☐	☐	24 VDC, 0.1 to 25 ms switc...	
DigitalInput04	BOOL			☐	☐	24 VDC, 0.1 to 25 ms switc...	
DigitalInput05	BOOL			☐	☐	24 VDC, 0.1 to 25 ms switc...	
DigitalInput06	BOOL			☐	☐	24 VDC, 0.1 to 25 ms switc...	
DigitalInput07	BOOL			☐	☐	24 VDC, 0.1 to 25 ms switc...	
DigitalInput08	BOOL			☐	☐	24 VDC, 0.1 to 25 ms switc...	
DigitalInput09	BOOL			☐	☐	24 VDC, 0.1 to 25 ms switc...	
DigitalInput10	BOOL			☐	☐	24 VDC, 0.1 to 25 ms switc...	
DigitalInput11	BOOL			☐	☐	24 VDC, 0.1 to 25 ms switc...	
DigitalInput12	BOOL			☐	☐	24 VDC, 0.1 to 25 ms switc...	

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 mapping should be transferred.

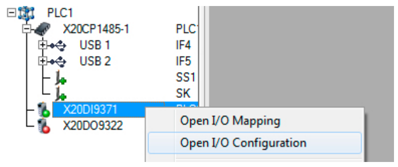
Selecting the task class as Automatic means that Automation Studio automatically determines the fastest task class where the variable is being used when the project is built.

4.5.4 I/O configuration

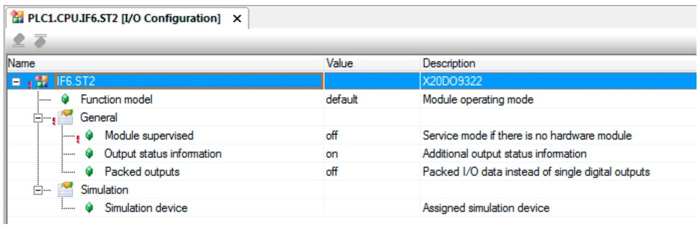
Module-specific properties can be configured in the **I/O configuration** without requiring any extra programming.

The ever-increasing functionality of remote B&R I/O modules results in more and more possibilities and operating modes in which these modules can be used.

The configuration can be opened in **physical view** by selecting <**Open I/O Configuration**> from the shortcut menu.



Opening the I/O configuration



PLC1.CPU.IF6.ST2 [I/O Configuration]

Name	Value	Description
IF6.ST2		X20DO9322
Function model	default	Module operating mode
General		
Module supervised	off	Service mode if there is no hardware module
Output status information	on	Additional output status information
Packed outputs	off	Packed I/O data instead of single digital outputs
Simulation		
Simulation device		Assigned simulation device

Function model for an I/O card

4.5.5 Error handling for I/O modules

A core component of handling I/O module errors lies in the fact that each configured I/O module is monitored by the I/O system.

The user can determine how the system responds to error situations.

Monitoring the I/O module can be activated (default) or deactivated in the **I/O configuration** using the "**Module supervised**" property.

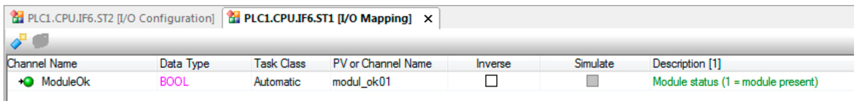
Monitoring active

When the module is actively being monitored by Automation Runtime, the following states cause a restart in service mode:

- The module defined to a slot is not present (not inserted).
- The module physically inserted in the slot doesn't match the module actually configured for the slot.
- The module cannot be addressed by the I/O system (e.g. module defective).

Monitoring inactive

When monitoring is deactivated, mapping a variable to the "**ModuleOK**" channel makes it possible to react to any errors from the application.



PLC1.CPU.IF6.ST2 [I/O Configuration] | PLC1.CPU.IF6.ST1 [I/O Mapping]

Channel Name	Data Type	Task Class	PV or Channel Name	Inverse	Simulate	Description [1]
ModuleOk	BOOL	Automatic	modul_ok01	☐	▨	Module status (1 = module present)

Module monitoring via the application

Real-time operating system / Target systems / SG4 / I/O Management / Functionality / Error handling

5 INTERACTION IN A MULTITASKING SYSTEM

This chapter describes how Automation Runtime behaves when running and editing one or more programs.

5.1 Task interrupts another task

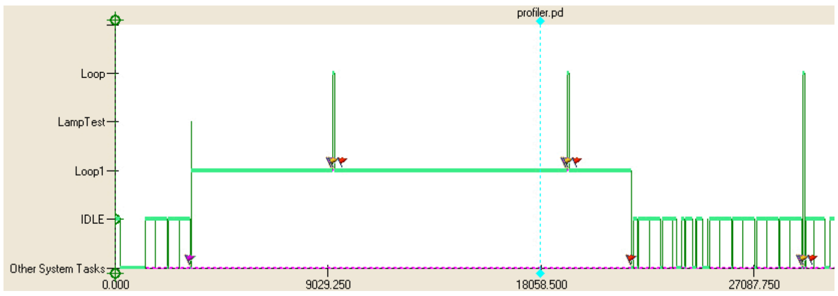
A task in a higher priority task class may interrupt a task from a lower priority task class with a longer duration.

Interpret the task classes in a system using the Profiler

Based on the tasks "**Loop**" and "**Loop1**", use the variable monitor to change the final value of the variable "**udiEndValue**" so that the task "**Loop1**" is interrupted exactly **twice** by the task "**Loop**".

Set the starting value for the variable "**udiEndValue**" in the "**Loop**" task to 2000.

The Profiler measurement looks like this:



Interrupting tasks

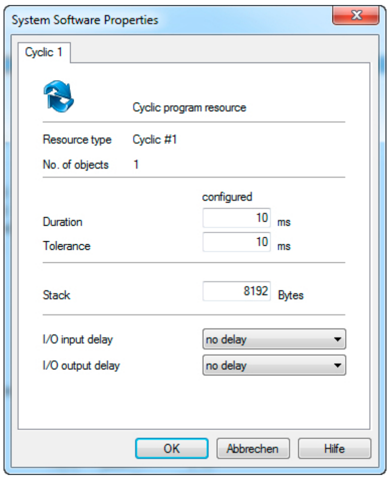
When the task "Loop1" is executed in task class #4, the input mapping is available until the task has been completely executed.

5.2 Synchronizing I/O cycle with task class

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

However, this is not sufficient for processing high-speed I/O data such as via POWERLINK or X2X.

As described in 4.4.6 "I/O scheduler"data collection can be synchronized with the fieldbus I/O cycle. In order for the data to also be processed in the I/O collection cycle, the task class must be called more quickly and more often.


System Software Properties

Cyclic 1

Cyclic program resource

Resource type	Cyclic #1	
No. of objects	1	
	configured	
Duration	10	ms
Tolerance	10	ms
Stack	8192	Bytes
I/O input delay	no delay	
I/O output delay	no delay	

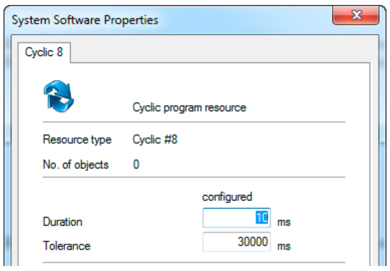
Configuring the cycle time for task class #1

Project management / Physical view / System configuration / SG4 system configuration / Cyclic / Cyclic n

5.3 Task class with high tolerance

Tasks that are to be processed as quickly as possible but with a low priority should be assigned to task class #8.

This task class has a default cycle time of 10 milliseconds and a tolerance of 30 seconds.


System Software Properties

Cyclic 8

Cyclic program resource

Resource type	Cyclic #8	
No. of objects	0	
	configured	
Duration	10	ms
Tolerance	30000	ms

Configuring task class #8

Tasks that should be assigned to task class #8:

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

5.4 Transferring programs

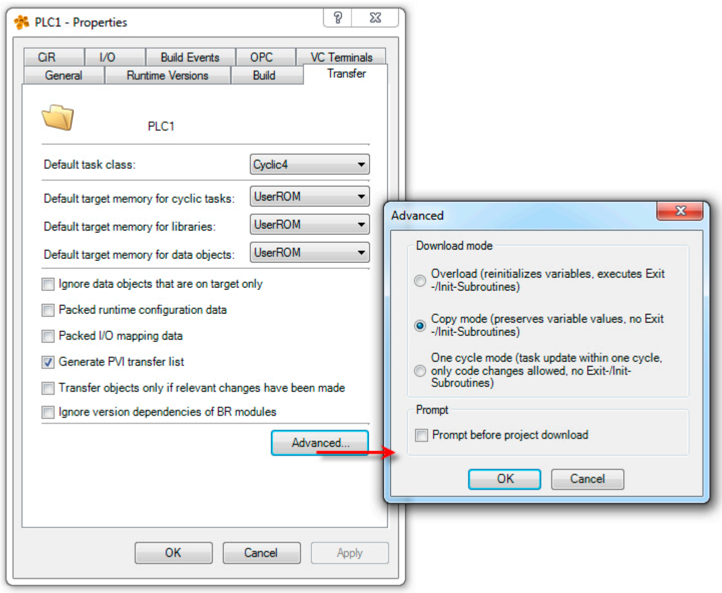
Programs are transferred to the target system after the program code or system configuration is changed.

5.4.1 Download modes

With respect to transferring program changes, there are two different situations that must be considered:

- Transfer during development – No effect on the production system
- Transfer during runtime – Effects on the production system

Depending on the situation, the user must select the appropriate download mode in the advanced transfer properties for the respective configuration in the configuration view.



PLC1

Default task class: Cyclic4

Default target memory for cyclic tasks: UserROM

Default target memory for libraries: UserROM

Default target memory for data objects: UserROM

- [ ] Ignore data objects that are on target only
- [ ] Packed runtime configuration data
- [ ] Packed I/O mapping data
- [✓] Generate PVI transfer list
- [ ] Transfer objects only if relevant changes have been made
- [ ] Ignore version dependencies of BR modules

Download mode

- ○ Overload (reinitializes variables, executes Exit -/Init-Subroutines)
- ● Copy mode (preserves variable values, no Exit -/Init-Subroutines)
- ○ One cycle mode (task update within one cycle, only code changes allowed, no Exit-/Init-Subroutines)

Prompt

- [ ] Prompt before project download

Setting the transfer mode in the configuration view

Transfer during development – Overload mode

When frequent changes are made to the program code during development, overload mode should be used. This is the default transfer mode for a new configuration.

Overload mode is not suited for a running production system.

Transfer during runtime – One cycle mode

This transfer mode is suited for making changes to a production system. In this mode, only one program is exchanged in one cycle. This guarantees the fastest possible changeover times for the production system.

Transfer during runtime – Copy mode

This transfer mode is suited for making changes to a production system. The programs are transferred over multiple cycles.

Real-time operating system / Target systems / SG4 / Download / Overload

Real-time operating system / Target systems / SG4 / Download / OneCycleMode

Real-time operating system / Target systems / SG4 / Download / CopyMode

5.4.2 The exit program

A task's exit program is called when uninstalling (deleting) the task.

If resources (memory, interfaces) were used in the initialization or cyclic program, then these resources must be freed up correctly.

5.5 System behavior of retain variables

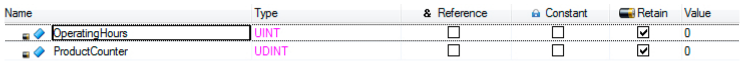
Beim Hochlauf des Zielsystems bootet das Automation Runtime und durchläuft alle Bootphasen (siehe 4.1 "Starting Automation Runtime").

In order for the values of process variables to be retained after a restart, they must be stored in battery-buffered remanent memory and automatically reloaded at startup.

5.5.1 Retain variables

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

- Target system with SRAM and battery buffering
- Configuring variables as "retain"



Name	Type	Reference	Constant	Retain	Value
OperatingHours	UINT	☐	☐	☑	0
ProductCounter	UDINT	☐	☐	☑	0

Battery-buffered variables

Programming / Variables and data types / Variables / Remanent variables

Different target systems have varying amounts of SRAM available.

Information about the size and availability of SRAM can be found in the respective hardware documentation.

Examples of data to be stored in SRAM:

- Operating time counters
- Number of product errors
- Product IDs
- Type counters
- etc.

The CompactFlash card should be used to store data that is only read or written once when the program is started (e.g. configuration data) or when a change is made (e.g. to a recipe).

A restart is triggered by the following actions:

- Power on
- Changing a system configuration and transferring it to the target system
- Performing a warm restart from Automation Studio (same as power on)

Retain variables keep their values after a **warm restart** that is triggered by a **power on** or from Automation Studio.

The following actions cause even retain variables to be initialized with "0" when the system boots = **cold restart**.

A cold restart is triggered by the following actions:

- • Restart after replacing the Compact Flash
- • Restart after clearing the UserROM
- • Performing a cold restart from Automation Studio
- • Restart when the buffer battery for the retain variables is defective.

Battery

In order for retain variables to be stored permanently, they must be inserted and managed in the permanent variable editor.

Examples of data that should be stored permanently in battery-buffered SRAM:

- Operating time counters

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

Programming / Editors / Configuration editors / Permanent variables

Permanent memory is never formatted or written by the system. It is 100% the responsibility of the user to manage the variable values when the application is started.

Permanent variables without a correct basis can result in varying and inconsistent program behavior.

This is especially important to remember when replacing the CPU or battery.

6 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 makes it possible to design an application with a modular structure. The arrangement of the application in tasks grouped into task classes enables optimum utilization of the resources.

Automation Runtime target systems

The timing of the application can be configured by adjusting the multitasking system to the user's unique requirements.

Tasks that should be executed quickly and frequently run in a faster task class, while other important tasks that are less time critical can run in a slower task class.

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

TRAINING MODULES

TM210 – The Basics of Automation Studio
TM211 – Automation Studio Online Communication
TM213 – Automation Runtime
TM220 – The Service Technician on the Job
TM223 – Automation Studio Diagnostics
TM230 – Structured Software Generation
TM240 – Ladder Diagram (LD)
TM241 – Function Block Diagram (FBD)
TM242 – Sequential Function Chart (SFC)
TM246 – Structured Text (ST)
TM250 – Memory Management and Data Storage
TM261 – Closed Loop Control with LOOPCONR
TM400 – The Basics of Drive Technology
TM410 – ASiM Basis
TM440 – ASiM Basic Functions
TM441 – ASiM Multi-Axis Functions
TM450 – ACOPOS Control Concept and Adjustment
TM460 – Starting up Motors
TM500 – Basics of Integrated Safety Technology
TM510 – ASiST SafeDESIGNER
TM540 – ASiST SafeMC
TM600 – The Basics of Visualization
TM630 – Visualization Programming Guide
TM640 – ASiV Alarms, Trends and Diagnostics
TM670 – Visual Components Advanced
TM700 – Automation Net PVI
TM710 – PVI Communication
TM711 – PVI DLL Programming
TM712 – PVIServices
TM810 – APROL Setup, Configuration and Recovery
TM811 – APROL Runtime System
TM812 – APROL Operator Management
TM813 – APROL XML Queries and Audit Trail
TM830 – APROL Project Engineering
TM890 – The Basics of LINUX