Automation Runtime

TM213

Prerequisites and requirements

Training modules:	TM210 – Working with Automation Studio
Software	Automation Studio 4.0
Hardware	X20CP1486

TABLE OF CONTENTS

1 INTRODUCTION

The real-time operating system – Automation Runtime – is an integral component of Automation Studio.

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 time frame. This makes it possible to achieve optimum 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 configure Automation Runtime in for your application Automation Studio.

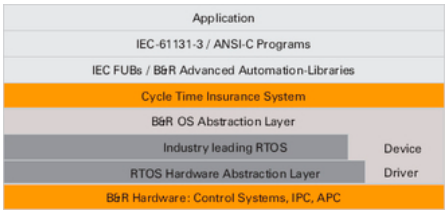
You will learn …

- 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 deterministic, hardware-independent, multitasking tool for creating applications.

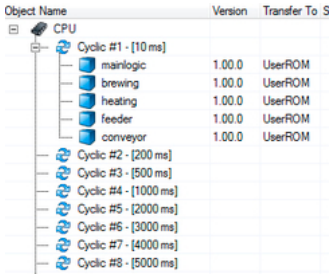
The IEC library functions in Automation Runtime make programming faster and easier while helping to avoid errors.



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.



Time basis for executing programs

2.1 Requirements and functions

Automation Runtime is fully integrated in the respective target system. It allows application programs to access I/O systems, interfaces, fieldbuses, 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
- Guarantees deterministic behavior with cyclic runtime system
- Makes it possible to configure different cycle times

- Offers 8 different task classes
- Guarantees a response to timing violations
- Configurable tolerance limits for all task classes
- Offers extensive function libraries in accordance with IEC 61131-3
- Provides access to all networks and bus systems
- Contains an integrated FTP, Web and VNC server
- Provides comprehensive system diagnostics (SDM)

Real-time operating system \ Method of operation

2.2 Target systems

Automation Runtime can run on different hardware platforms.

These hardware platforms are basically a standard PLC – like the X20 CPU, the Power Panel or the Automation PC.

SG4 target systems

This training module describes the functionality and configuration options of B&R target systems.

Further information about target systems can be found in the help system.

Real-time operating system \ Target systems

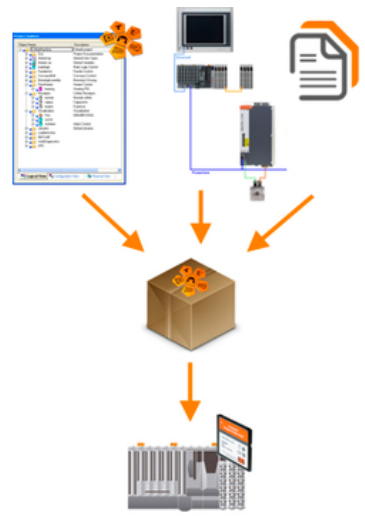
2.3 Installation and commissioning

Every target system[1] run Automation Runtime and the Automation Studio project from a CompactFlash 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.



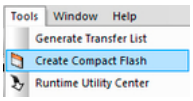
Installation and commissioning

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 can be created in Automation Studio by selecting <**Tools**> / <**Create CompactFlash**> from the menu.

After startup, the project is checked to see if it has already been built successfully. If not, it is rebuilt.

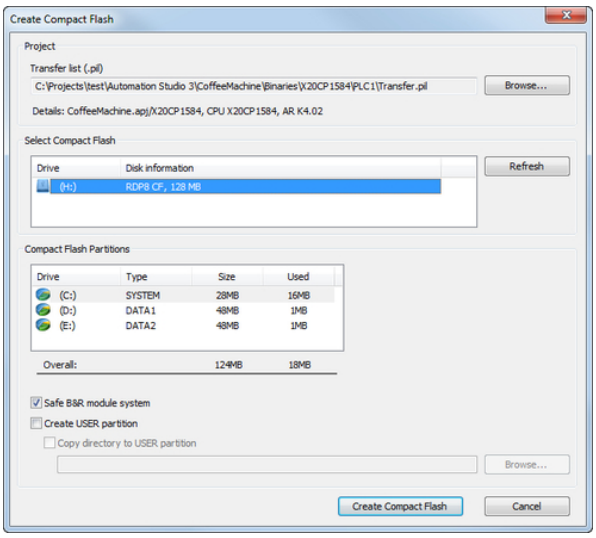


Create CompactFlash

[1] Some target systems have internal application memory installed in the device. Automation Runtime and the application program are loaded from the internal application memory. Automation Runtime is installed on these systems via the serial or Ethernet port.

In order to create the CompactFlash, you'll need a suitable CompactFlash card reader.

In the dialog box shown below you can select the desired CompactFlash card. After making any necessary user-specific settings, you can start generating the CompactFlash data.



Dialog box for creating a CompactFlash card

On the productive system, it is recommended to use the "**Safe B&R module system**".

Diagnostics and service \ Service tools \ Runtime Utility Center \ Creating a list / data medium \ CompactFlash functions \ Creating a CompactFlash

2.3.2 Installation by using the online connection

Automation Runtime can also be transferred via the online connection. To do this, the online interface must be configured correctly or the proper operating state active (4.5.1 "Startup").

Beginning with Default Runtime[2] V3.06 for SG4 systems, browsing for target systems with SNMP[3] is also supported.

Connection

A connection to the controller is established using the browse for target system function in Automation Studio. This searches the network for B&R controllers. It is also possible to temporarily modify controller connection settings in the search window.

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

[2] Default Runtime is a reduced variant of Automation Runtime that is preinstalled on all controllers. Default Runtime is responsible for the actual boot process from the CompactFlash card.

[3] The Simple Network Management Protocol is a network protocol that is used to monitor and control devices in a network (e.g. routers, servers, switches, printers, computers, etc.) from a central location. [Source: de.wikipedia.org]

Transfering the Automation Runtime

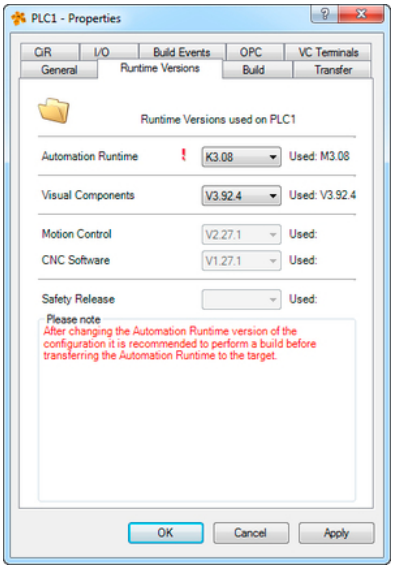
Once the connection is established, the Automation Runtime can be transferred.

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

2.4 Changing and updating Automation Runtime

If new software versions for the Automation Runtime, Visual Components, motion or CNC runtime environments are available, these software versions can be changed for the active configuration in the Automation Studio project.

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



Change the runtime versions

> Project management \ Changing runtime versions

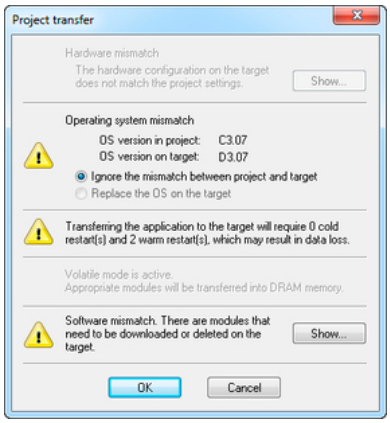
> Changing the motion control, CNC software and Visual Components versions may affect all hardware configurations since only a single instance of the libraries they use exists for all configurations in the Logical View.
>
> After the Automation Runtime version is changed for a configuration, it is necessary to recompile (rebuild) the project before it is transferred.

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.

> When using an Ethernet connection, ensure that the "**sysconf**" module is configured for "**SystemROM**" target memory (default) since UserROM is deleted once the new version of Automation Runtime has been transferred. After the UserROM is deleted, it is no longer possible to establish a connection to the controller. The target memory for software modules can be set in the software configuration.

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



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.

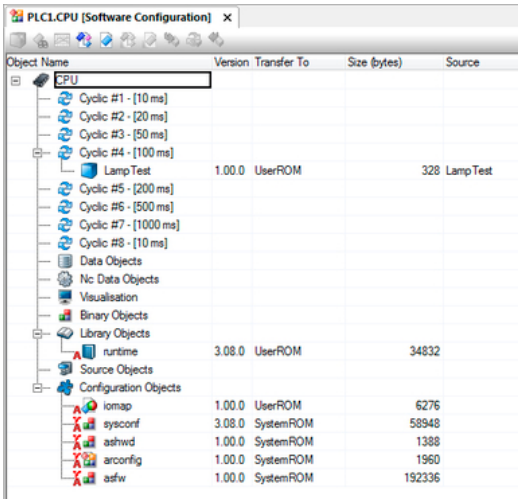
3.1 System and User ROM

During a build, an Automation Studio project generates modules with the extension .br that 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**.

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

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



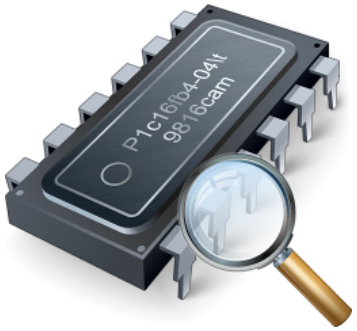
Target memory for .br modules

CompactFlash = System ROM + User ROM

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.

Target systems always have **DRAM**; **SRAM** is optional.



DRAM and SRAM

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

DRAM: DRAM is RAM memory that takes on an undefined state after startup.

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

A BR module in RAM requires the local or global variable memory area configured in Automation Studio (see 4.2.3 "Initializing the variable memory") and is also responsible for initializing this memory area.

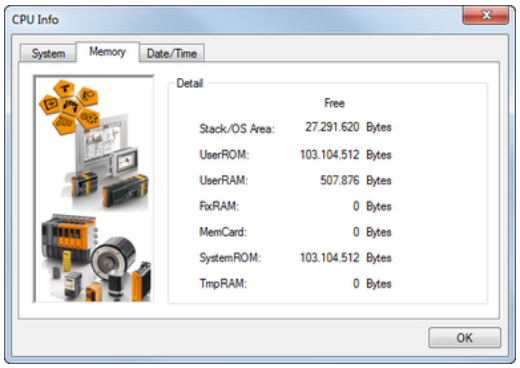
3.2.1 Tools for determining memory requirements

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

- Reading information from the CPU
- System Diagnostics Manager
- The HwGetBatteryInfo() function block in the AsHW library

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. The online info also shows the status of the backup battery as well as the current time on the controller.

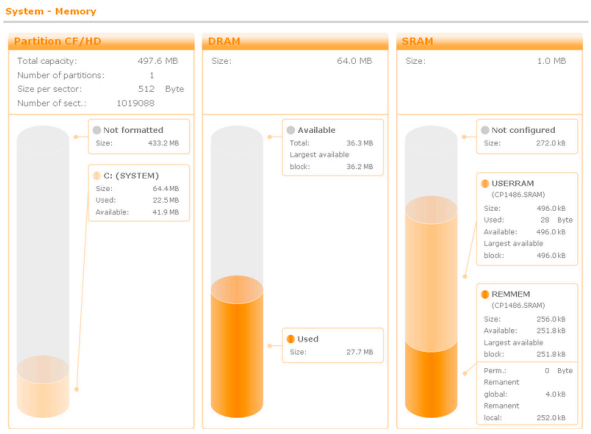


Reading information from the CPU

System Diagnostics Manager (SDM)

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

When using a proxy server, the local addresses for accessing the controller should be bypassed.



Memory allocation in SDM

Exercise: Check the free memory on the target system

CompactFlash data should be generated from the project created in "TM210 – Working with Automation Studio".

It must be possible for Automation Studio and the target system to communicate via an Ethernet connection. Determine the memory requirements of the target system using one of the methods described in section Chapter 3.2.1.

Diagnostics and service \ Diagnostic tools \ Information about the target system

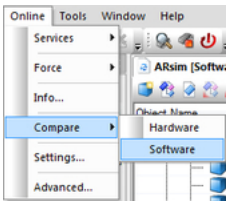
Diagnostics and service \ Diagnostics tools \ System Diagnostics Manager

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

3.2.2 Online software comparison

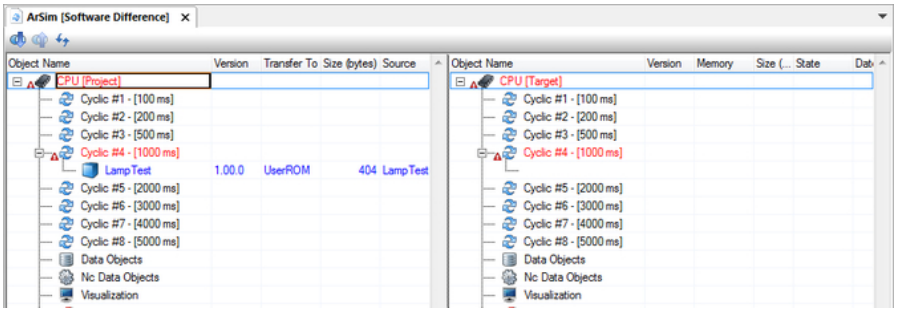
When there is an active online connection, it is possible to compare the modules configured in Automation Studio with the modules installed on the target system.

This software comparison can be launched by selecting <**Online**> / <**Compare**> / <**Software**> from the main menu.



Opening the software comparison

The software comparison window displays all of the modules in the project as well as on the controller. The "Size (bytes)" column displays lists the target memory of the BR module on the target system. The version, size, target memory and build date of modules can be compared.

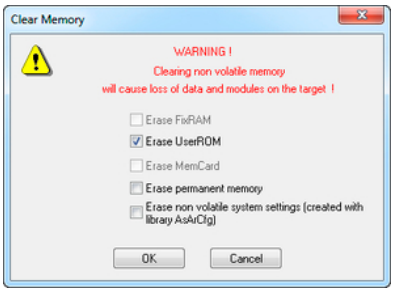


Opening the online software comparison window

Exercise: Delete the modules in User ROM

Select <**Online**> / <**Services**> / <**Clear memory**> from the main menu to delete the User ROM.

Then reactivate monitor mode in the software configuration.



Dialog box for deleting memory: User ROM selected

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.

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

This briefly interrupts the online connection.

4 RUNTIME PERFORMANCE

This section illustrates 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.

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

- Hardware check
- Hardware/firmware upgrade if necessary
- Checking BR modules
- 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 \ Method of operation \ Boot behavior

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

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

X20CP1485 in RUN mode

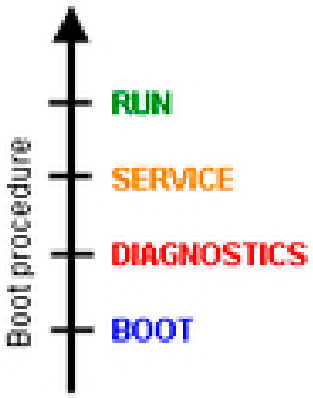
For information on troubleshooting error states, consult the training module "TM223 – Automation Studio Diagnostics".

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:

Operating 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[4] "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  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.

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

[4] Depending on the device being used, either a node selection switch is used to set its operating mode or there is a specific switch available for this. Additional information can be found in the respective user's manual.

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.

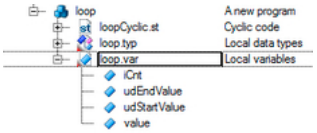
A variable's scope and properties determine its behavior during startup and runtime.

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.



Local variables from "Loop" program

Exercise: Create a program named "loop", use local variables

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

Four variables with the names „iCnt“, „value“, „udStartValue “ and "udEndValue" using the datatype UDINT will be inserted.

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

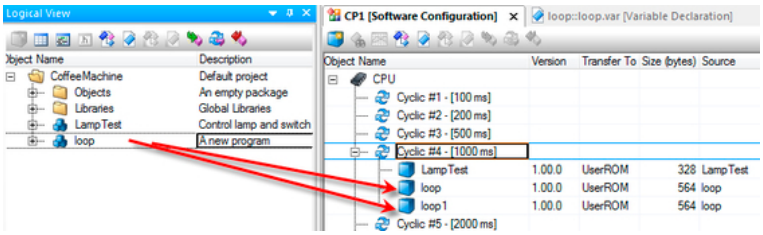
```
PROGRAM_CYCLIC
    FOR  iCnt  :=  udStartValue   TO udEndValue  DO
        value:= value + 1;
    END_FOR
END_PROGRAM
```

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.

Exercise: 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.

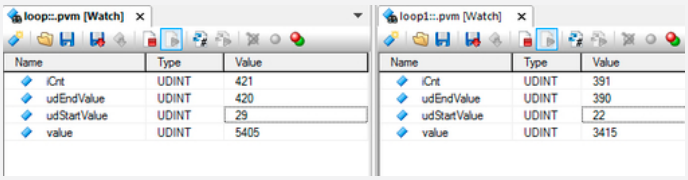


Multiple assignment of programs

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.



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

Programming \ Variables and data types \ Scope of declarations

4.2.2 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. In other words, they can be used in every program.

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

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 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 the global variable "gMain", use it in the programe "Loop"

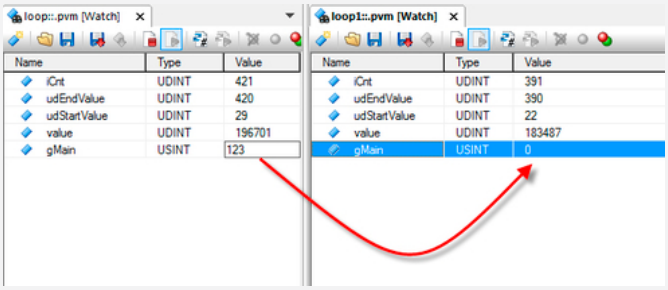
Enter a new variable in the global variable declaration "**Global.var**" and name it "**gMain**" with the data type UDINT.

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.



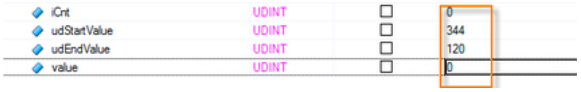
Writing global variables

Programming \ Variables and data types \ Scope of declarations

Programming \ Editors \ Configuration editors \ Variable mapping

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.



Variable initialization in "Loop.var"

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

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

Exercise: Initialization of the variable "udEndValue"

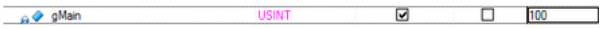
Configure the variable "**udiEndValue**" in the program "**Loop**" with an initial value of 50. Monitor the value in the variable monitor.

1) Open the variable declaration "**Loop.var**"
2) Configure the value in the column „**Value**“ of variable „**udEndValue**“

The variable „**udEndValue**“ of the programs „**Loop**“ and „**Loop1**“ will be initialized with the value 50 in the watch window. At runtime, the variable can be changed to any other value.

4.2.4 Constants

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



Declaration of constants

Exercise: Using 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**".

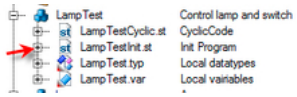
This is not permitted for constants. In order for the program to operate without errors, the variable "**gMain**" must not be defined as a constant.

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

Programming \ Variables and data types \ Variables \ Constants

4.3 Program initialization

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

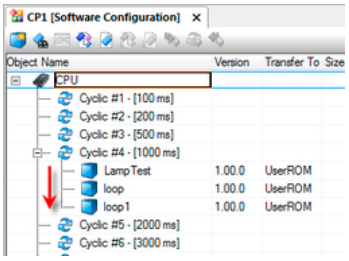


Program initialization

The initialization subroutine initializes variables and reads PLC data and machine configurations that will be executed in the cyclic subroutine.

4.3.1 Running the initialization program

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



Execution of the initialization program

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

1 Init-program of task „LampTest“
2 Init-program of task „Loop"
3 Init-program of task „Loop1“

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

Real-time operating system \ Target systems \ SG4 \ Runtime performance \ Starting initialization subroutines

4.3.2 Function calls in the initialization program

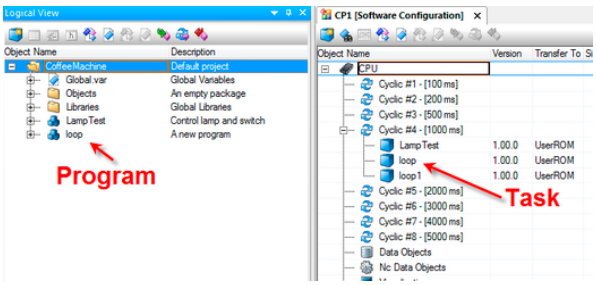
When using functions, it is important that the function return a result the first time it is called.

Functions that must be called several times 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.



Relationship between program and task

4.4.1 Multitasking operating system

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

Tasks are assigned to defined resources and executed one after the other in "**time slices**". Although execution times of tasks can vary, the moment the task starts as well as when the I/O system is accessed are defined points in time.

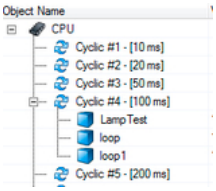
4.4.2 Task classes and cycle times

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

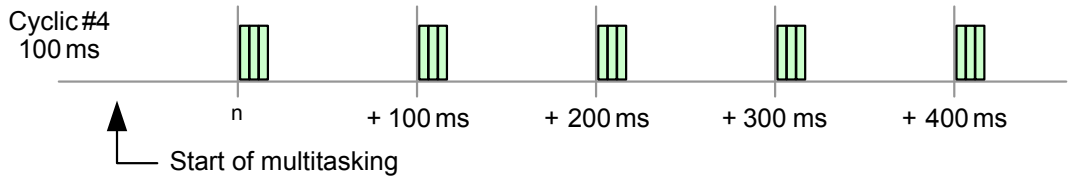
Up to eight task classes with configurable cycle times are provided to help optimize a task for its particular requirements. A task class contains tasks with the same cycle time, priority and tolerance.

Not all tasks have to run within the same time frame. Control tasks that must be executed quickly should be assigned a 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:



Cycle time



The tasks will be called in every cycle.

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

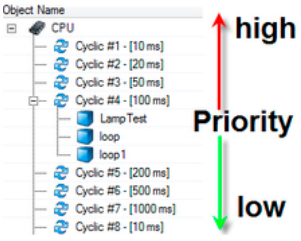
Real-time operating system \ Target systems \ SG4 \ Runtime behavior \ Start of cyclic tasks

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.

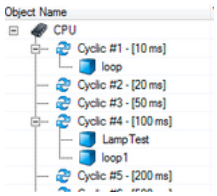
Moving a task between task classes changes its priority and cycle time.



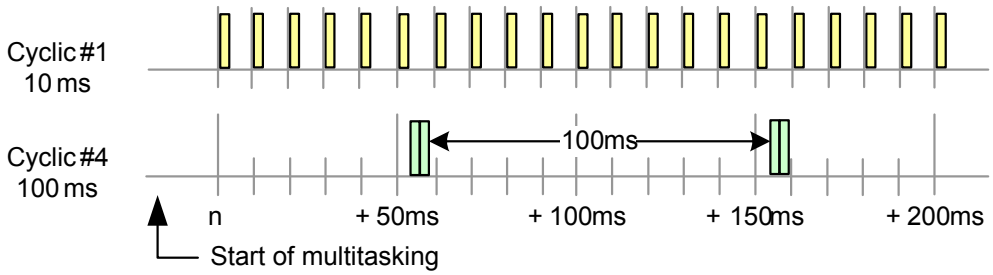
Task class priority

The execution time is not changed by moving a task. It changes the number of calls in the same time period.

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



Different task classes



Executing tasks in different taskclasses

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

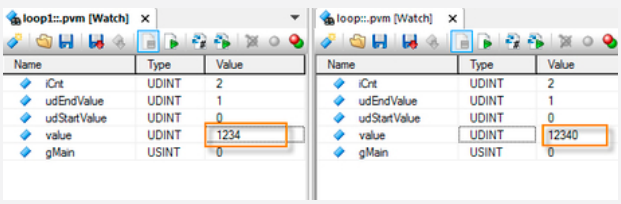
Exercise: Moving the task „Loop“ into task class#1

The "**Loop**" task in the software configuration should be moved to **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) Moving 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.



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, for example, that 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 \ Start of cyclic tasks

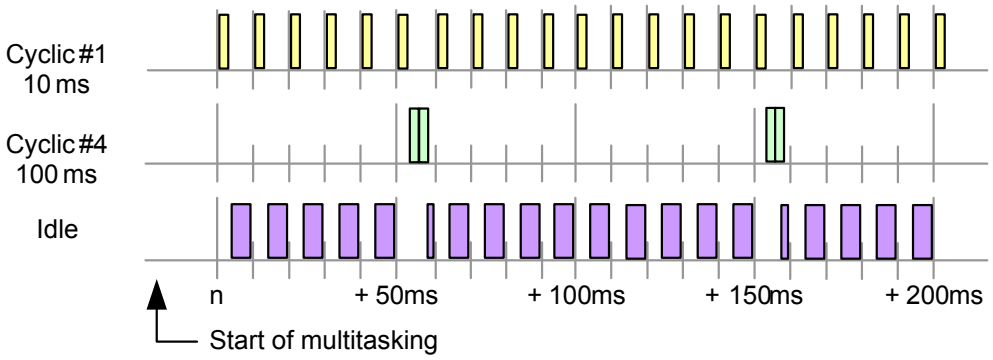
4.4.5 Idle time

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.

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

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.

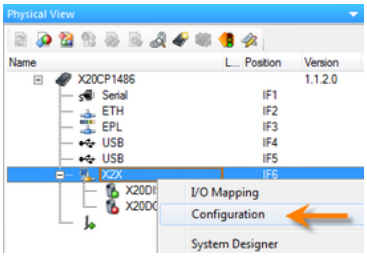
This ensures that the inputs at the beginning of the task class and the outputs at the end of the task class are transferred promptly to the mapped process variables.

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

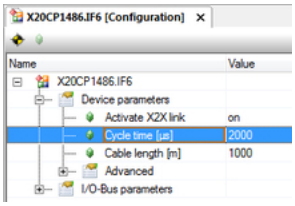
The I/O channels are mapped in the cycle of the I/O fieldbus system.

The cycle can be configured in the properties of the respective fieldbus. For example, the <**Configuration**> option can be selected from the X2X Link interface's shortcut menu.



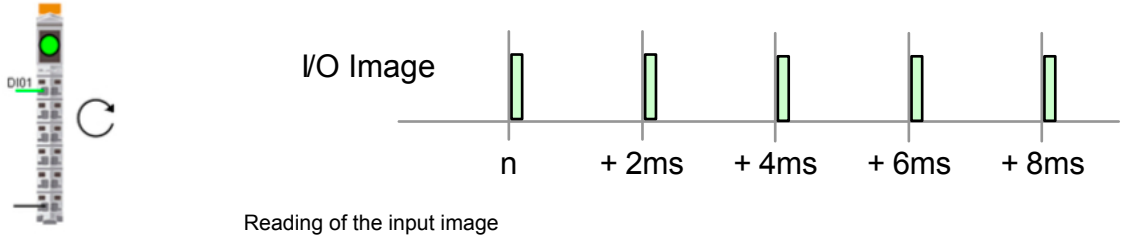
Opening the properties of the X2X Link interface

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



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 of the input image

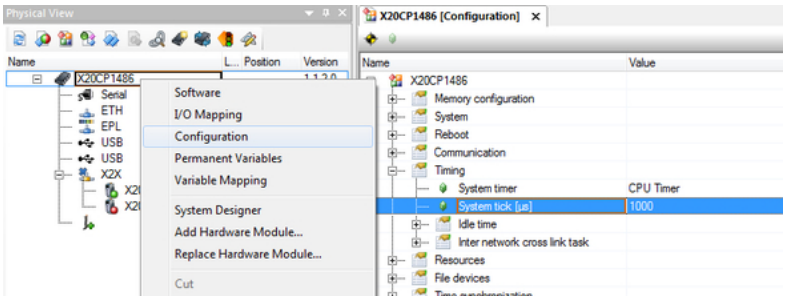
For the application, this means that the input data is not older than the configured cycle time after the start of the task class, and the output data will be written using this mapping after this time has passed at the latest.

I/O data in the task class

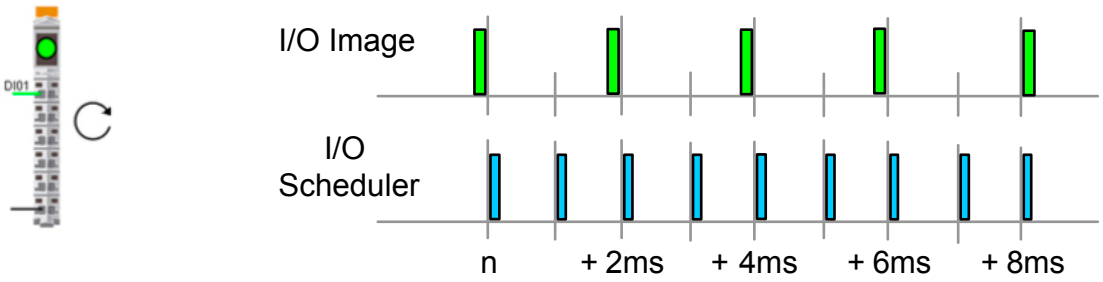
The I/O scheduler controls when the I/O data for the task classes is provided and the task classes are started.

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

This timing can be configured in the system configuration in the "Timing" group.



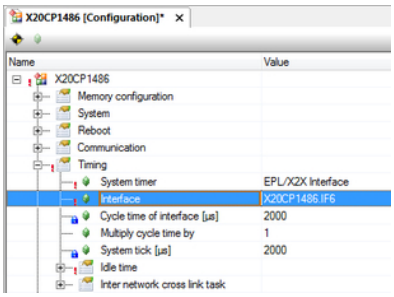
Configurable system tick



Reading of the input image

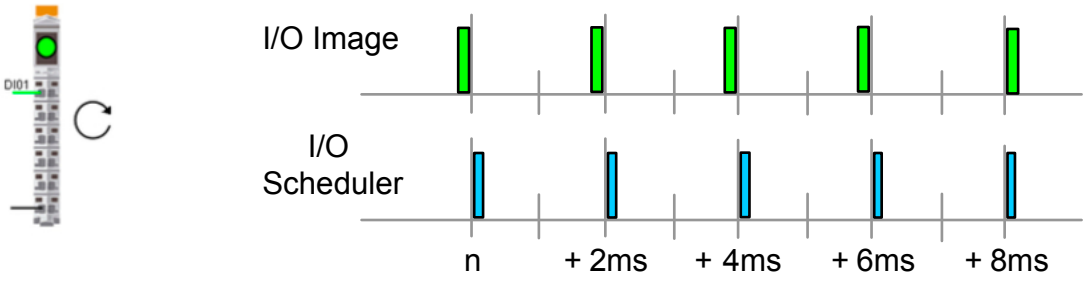
In this case, the I/O scheduler is called twice as often as the I/O mapping is updated.

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.



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 of the input image

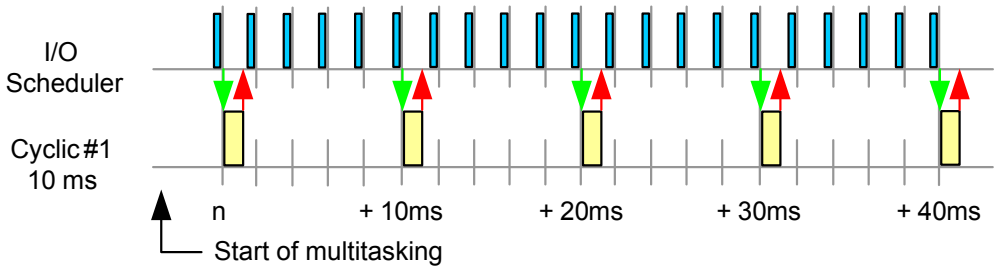
I/O management assigns I/O data from the mapping to the variables.

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).

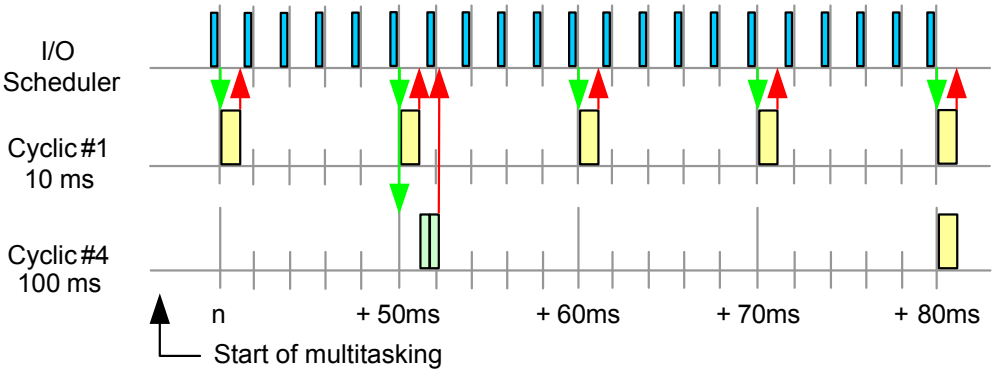


I/O handling for task class #1

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

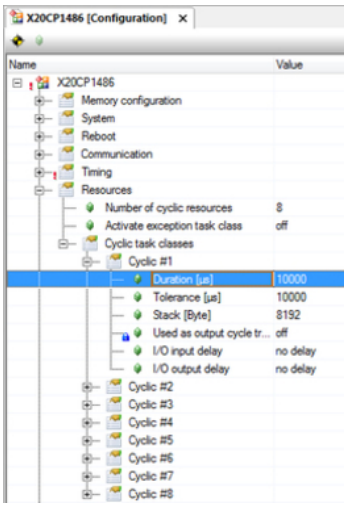
4.4.7 Cycle time and tolerance

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.

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.



Configuring the cycle time and tolerance

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.

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

Exercise: Causing a cycle time violation

In "**Loop**", which runs in the 10[ms] task class #1, change the value of the variable "**udiEndValue**" in the Watch window 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.

This exercise is composed of three different tasks.

1) Setting the "udEndValue" variable to 50000
2) Increasing the value of the "udEndValue" variable in steps
3) Increasing the value of the "udEndValue" variable until a cycle time violation occurs

Profiler results should be analyzed between each of these steps.

Step 1: Setting the "udEndValue" variable to 50000

- In the first step the value of variable „udEndValue“ should be changed to 50000.

The first step is to set the value of the "udEndValue" to 50000.

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

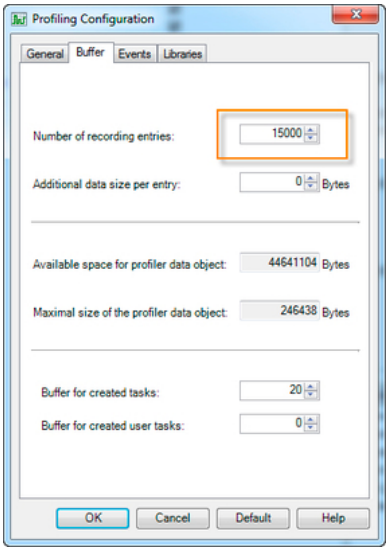
By default, Automation Runtime records all runtime behavior.

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

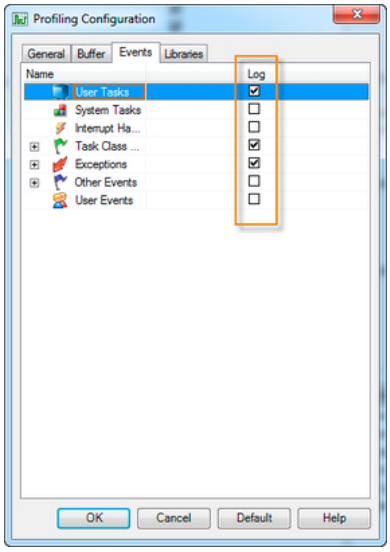
In order to edit the configuration, you must pause the current recording using the "**Stop**" icon in the Profiler toolbar.

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

Make the configurations shown in the following image:



Profiler Configuration 1

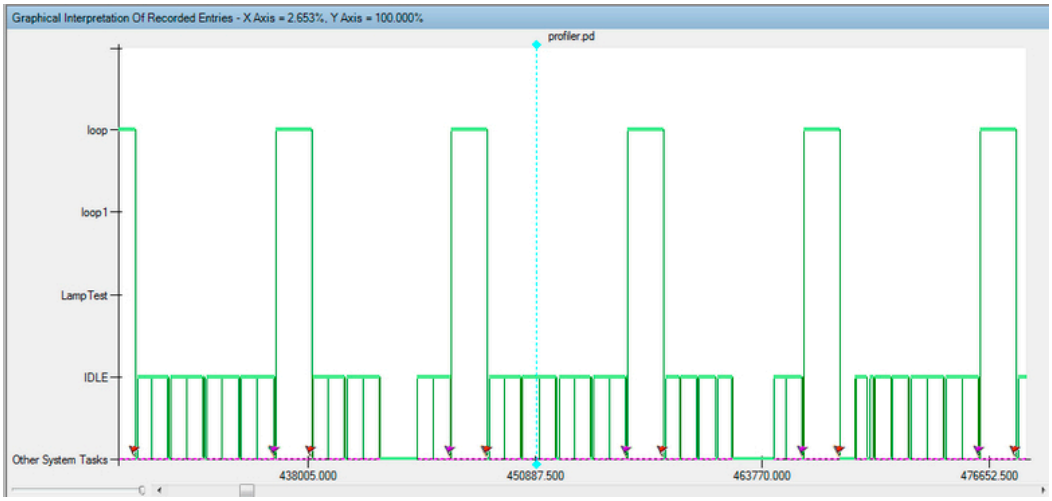


Profiler Configuration 2

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

Exercise: Checking the profiling

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

Diagnostics and service \ Diagnostics tools \ Profiler

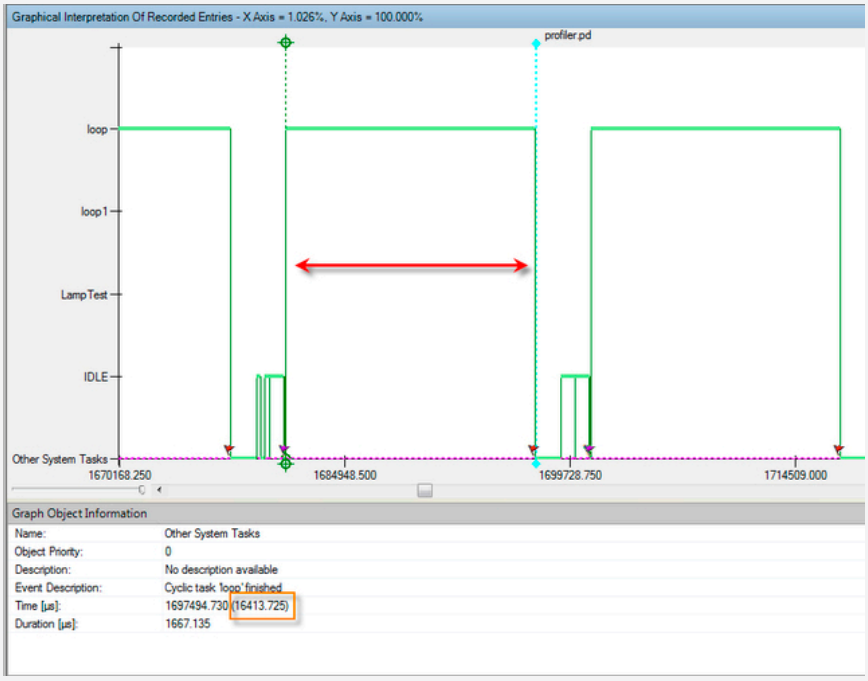
Step 2: Increasing the value of the "udEndValue" variable in steps

- In this step, restart the Profiler using the "Start" icon in the Profiler toolbar.
- The value of „**udEndValue**“ should be increased in steps of 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.



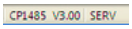
Exceeding the cycle time, effect of tolerance

Step 3: Increasing the value of the "udEndValue" variable until a cycle time violation occurs

- Increase the value of „udEndValue“ until reaching a cycle time violation

4.4.8 Cycle time violation

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



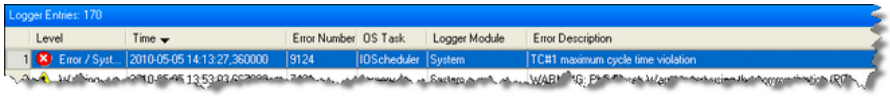
X20CP1485 in Service mode

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 window

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



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.

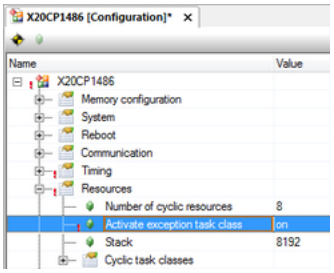
When the target system is restarted either by turning the power OFF/ON or by performing a warm restart in Automation Studio, it boots in run mode.

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.

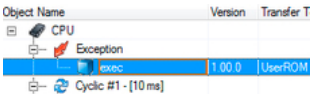
It is possible to respond to exceptions in an exception program.

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



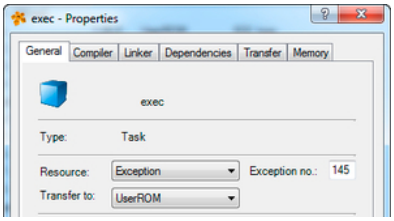
Enabling the exception task class

A program in the Logical View can be added to the exception task class in the software configuration.



Configuring the exception task class

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



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

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

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

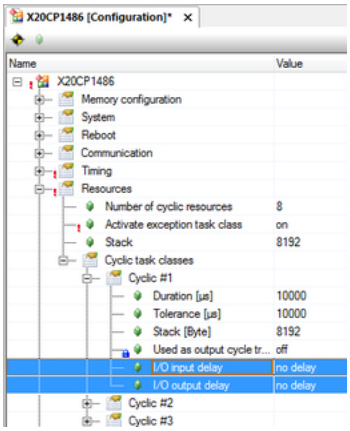
4.5 I/O management

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.

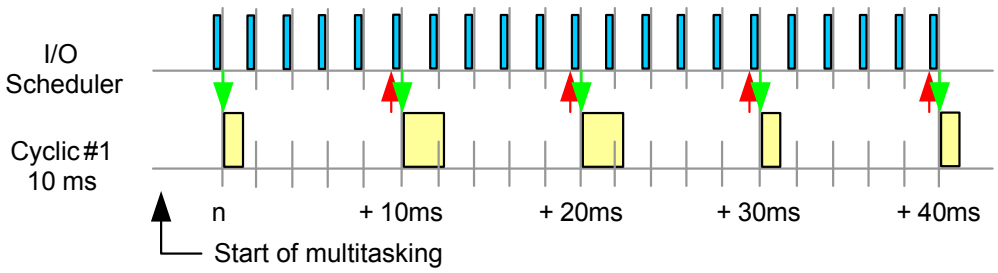
Automation Runtime I/O management is designed to meet the highest requirements regarding **response times** and **configurability**.

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.



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 \ Method of operation

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.

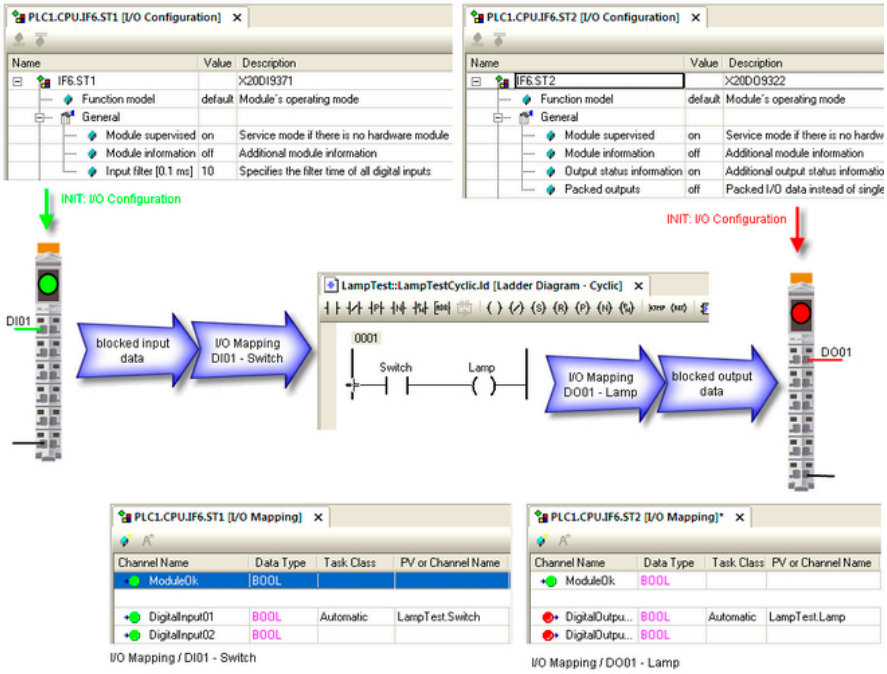
This ensures that valid I/O data is accessed in the initialization program.

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.



Basic I/O functionality

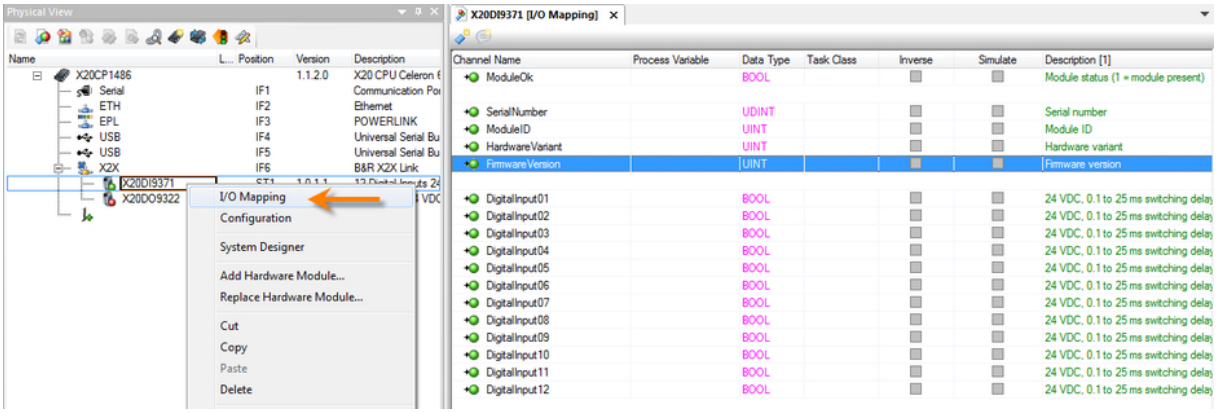
4.5.3 I/O mapping

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.

The variables can be assigned to the desired I/O module in the **Physical View** by selecting <**Open I/O mapping**> from the shortcut menu.

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



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.

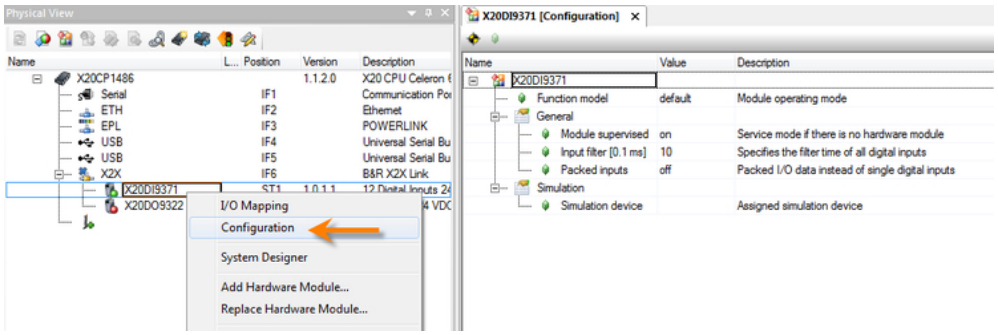
Programming \ Editors \ Configuration editors \ I/O mapping

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.

I/O channels are configured for the desired I/O module in the **Physical View** by selecting <**Open configuration**> from the shortcut menu.



I/O configuration of a digital input module

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

4.5.5 Error handling for I/O modules

A core component of the error handling is that each configured I/O module is monitored by the I/O system.

The users can configure how the system reacts in an error situation.

The monitoring of the I/O module can be enabled or disabled by changing the „**Module supervised**“ property in the **I/O Configuration** .

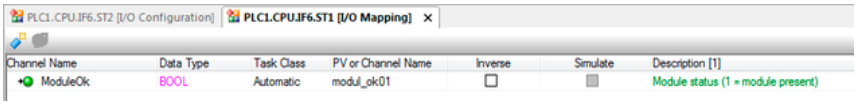
Monitoring is active

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 is inactive

If the monitoring is disabled the „**ModuleOK**“ channel can be used in the application to react on an error situation.



Monitoring the module by application

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

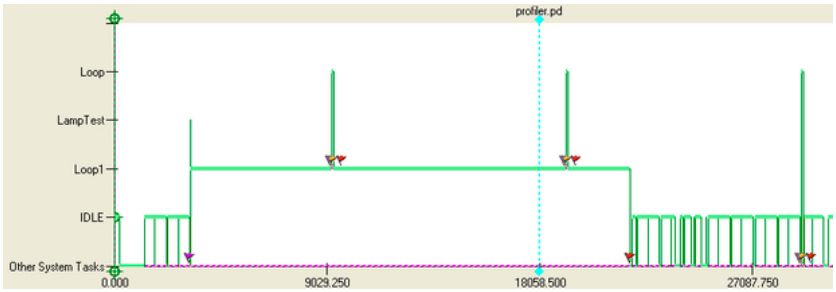
A task in a higher priority task class may interrupt a task from a lower priority task class with a longer duration.

Exercise: Interpret the task class system with 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

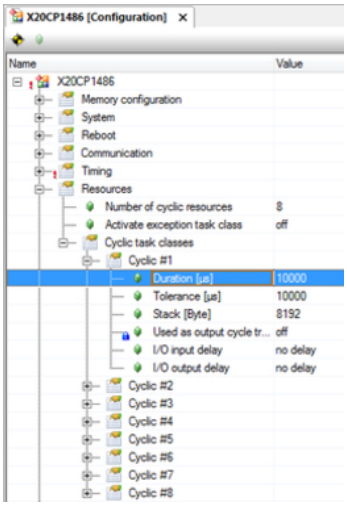
When the task "Loop1" is executed in task class #4, the input mapping is available until the task has been completely executed.

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.



Configuring the cycle time for task class #1

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

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.

Tasks that should be assigned to task class #8:

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

5.4 Transferring programs

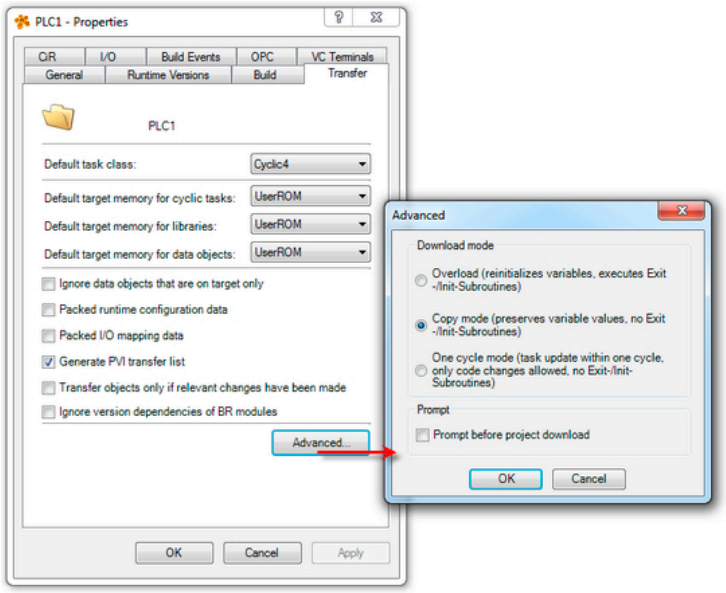
Programs are transferred to the target system after the program code or system configuration is changed.

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.



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 \ Copy mode

- Overload
- One cycle mode
- Copy mode

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

Automation Runtime starts automatically when the target system is booted and runs through all the boot phases (see 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"

Battery-buffered 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
- And much more

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.

If a system is restarted with a **cold restart**, then Retain variables are reinitialized with the value "0".

Cold restart

A cold restart is triggered by the following actions:

- Restart after exchanging the CompactFlash
- Restart after clearing the UserROM
- Performing a cold restart from Automation Studio
- Restart when the buffer battery for the retain variables is defective.

Permanent variables

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.

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.

Programming \ Variables and data types \ Variables \ Variable remanence

Programming \ Editors \ Configuration editors \ Permanent variables

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 – Working with Automation Studio
TM213 – Automation Runtime
TM220 – The Service Technician on the Job
TM223 – Automation Studio Diagnostics
TM230 – Structured Software Development
TM240 – Ladder Diagram (LD)
TM241 – Function Block Diagram (FBD)
TM242 – Sequential Function Chart (SFC)
TM246 – Structured Text (ST)
TM250 – Memory Management and Data Storage
TM400 – Introduction to Motion Control
TM410 – Working with Integrated Motion Control
TM440 – Motion Control: Basic Functions
TM441 – Motion Control: Multi-axis Functions
TM450 – ACOPOS Control Concept and Adjustment
TM460 – Initial Commissioning of Motors
TM500 – Introduction to Integrated Safety
TM510 – Working with SafeDESIGNER
TM530 – Developing Safety Applications
TM540 – Integrated Safe Motion Control
TM600 – Introduction to Visualization
TM610 – Working with Integrated Visualization
TM630 – Visualization Programming Guide
TM640 – Alarms, Trends and Diagnostics
TM670 – Advanced Visual Components
TM800 – APROL System Concept
TM811 – APROL Runtime System
TM812 – APROL Operator Management
TM813 – APROL XML Queries and Audit Trail
TM830 – APROL Project Engineering
TM890 – The Basics of LINUX
TM920 – Diagnostics and Service for End Users

TM1010 – B&R CNC system (ARNC0)
TM261 – Closed Loop Control with LOOPCONR
TM480 – The Basics of Hydraulics
TM481 – Valve-based Hydraulic Drives
TM482 – Hydraulic Servo Pump Drives

www.br-automation.com

TM213TRE.40-ENG / V2.0.0.4