TM213

Automation Runtime

PERFECTION IN AUTOMATION
www.br-automation.com

Prerequisites and requirements	
Training modules	TM210 – Working with Automation Studio
Software	Automation Studio 4.1 Automation Runtime V4.08
Hardware	X20CP1586

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 that allows applications to run on a B&R target system.

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

Figure 1: Automation Runtime target systems

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

1.1 Learning objectives

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

- You will learn what demands are placed on a real-time operating system in the field of automation.
- You will learn about the features and functionalities available in Automation Runtime.
- You will learn how a uniform runtime system can benefit integrated automation solutions.
- You will learn how memory management, variable scope and RETAIN variables are handled in Automation Runtime.
- You will learn about the startup and runtime behavior of B&R controllers.
- You will learn how Automation Runtime I/O management works.
- You will learn about the interrelationships involved in multitasking.
- You will learn about Automation Runtime's configuration options.

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.

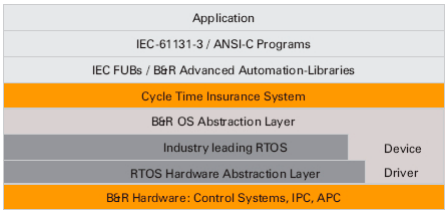

Figure 2: The operating system architecture

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

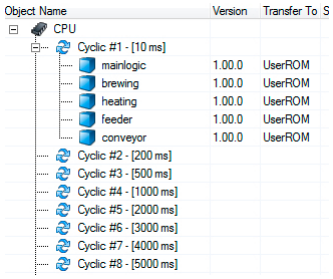

Figure 3: Timing framework for executing programs

Real-time operating system

2.1 Requirements and functions

Automation Runtime is fully integrated into B&R target systems. It allows application programs to access I/O systems, interfaces, field-buses, networks and storage devices.

Figure 4: Features of Automation Runtime

Automation Runtime provides a number of important features:

- Runs on all B&R target systems
- Makes the application hardware-independent
- Guarantees deterministic behavior with a cyclic runtime system
- Allows the configuration of different cycle times
- Offers eight different task classes
- Guarantees a response to timing violations
- Provides configurable tolerance limits for all task classes
- Offers extensive function libraries in accordance with IEC 61131-3
- Provides access to all networks and bus systems
- Contains an integrated web, FTP, and VNC server
- Provides comprehensive system diagnostics (SDM)

Real-time operating system \ Method of operation

2.2 Target systems

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

Figure 5: SG4 target systems

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

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

Real-time operating system \ Target systems

2.3 Installation and startup

Every target system[1] boots Automation Runtime and the Automation Studio project from a Flash memory.

A bootable CompactFlash card is set up the first time 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 then installed on the CompactFlash card.

Devices with an internal application memory are either updated via an Ethernet connection or the USB remote install mechanism.

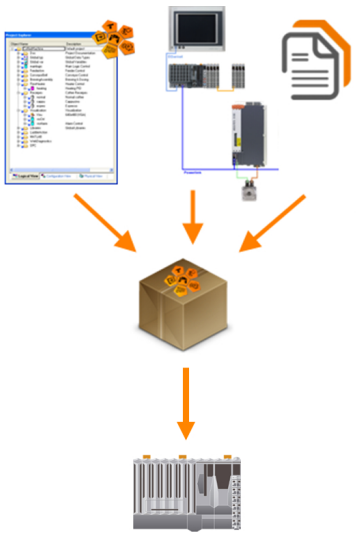

Figure 6: Installation and startup

[1] Some target systems have application memory installed internally in the device. In these cases, Automation Runtime and the application program are loaded from this internal application memory. Automation Runtime is installed on these systems via the serial interface, Ethernet interface or the USB remote install mechanism.

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

2.3.1 Generating an operational CompactFlash

Data for the CompactFlash card is generated in Automation Studio by selecting **<Tools> / <Create CompactFlash>** from the main menu.

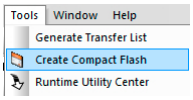

Figure 7: Generating CompactFlash data

After startup, the project is checked to see if it has already been built successfully without errors. If not, it is rebuilt.

A suitable CompactFlash adapter is needed to set up the CompactFlash card.

The desired CompactFlash card can then be selected from the window shown here. Once any necessary user-specific settings have been made, the process for generating the CompactFlash data can begin.

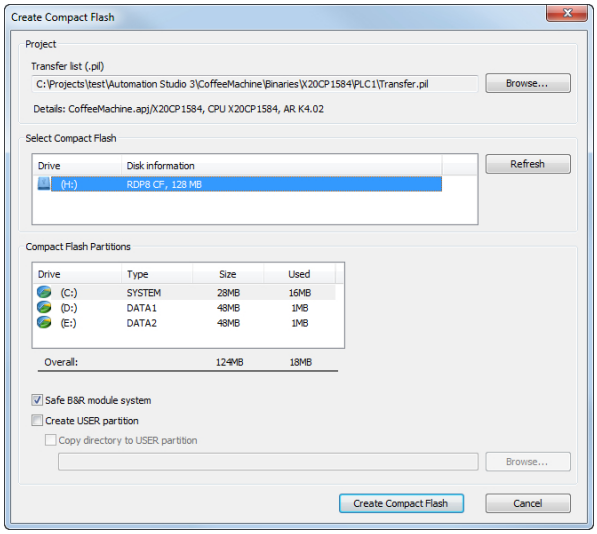

Figure 8: The dialog box for creating a CompactFlash

It is recommended to use the **Safe B&R module system** option on live production systems.

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

2.3.2 Installing over an online connection

It is also possible to transfer Automation Runtime over an online connection. In order to do this, the online interface must be configured correctly and be in the right mode. (4.4.1 "Startup" on page 41).

Browsing for target systems with SNMP[2] is also supported for SG4 systems beginning with Default Automation Runtime V3.06[3].

Many target systems have a reset button. Using the reset button, the system can be restarted and the operating mode can be changed. In the user's manual for your target system, you can find out how this can be used to put the control system into BOOT mode.

Establishing a connection

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

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

Transferring Automation Runtime

Automation Runtime can be transferred once the connection has been established.

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

2.3.3 Installation via USB remote install

Automation Runtime and application software can be transferred to the target system with the help of a USB stick.

In order to do that, the function USB remote install must have been enabled in the system settings of the controller.

First the Automation Studio project is compiled and a transfer list created.

The remote install structure is copied to a USB stick using the Runtime Utility Center, which is automatically installed when Automation Studio is installed.

The USB stick is inserted in the target system.

The versions of Automation Runtime and the application software are checked and updated if need be when the controller is next started up.

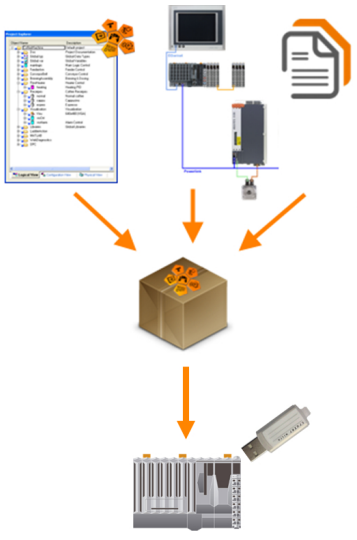

Figure 9: Automation Runtime and application are updated via USB remote install

[2] The Simple Network Management Protocol is a network protocol used to monitor and control devices on a network from a central location (e.g. routers, servers, switches, printers, computers, etc.).

[3] Default Automation Runtime is a reduced variant of Automation Runtime that comes preinstalled on all controllers. It is responsible for handling the actual boot process from the CompactFlash card, for example.

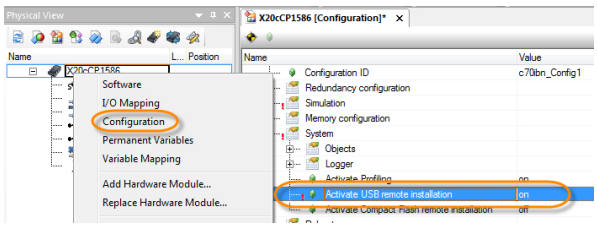

Figure 10: Enabling the USB remote install function in system properties

Enabling USB remote install

Enabling this function in the system configuration or checking this setting is required so that control systems[4] that have default AR onboard (such as the Power Panel C70) will continue to support this function after the first update via USB remote install. The configuration is opened in Physical View via the controller's shortcut menu. The required configuration entry is located in the "System" section.

Real-time operating system \ Target systems \ SG4 \ AR remote install \ Remote install sequence

- Target system with Default AR and CF
- Target system without Default AR and CF

Generating the transfer list

First, a transfer list is generated in Automation Studio. This happens automatically when compiling the project. This process can also be triggered by selecting <Tools > / <Generate transfer list>. The transfer list lists all required data for the installation of the control system.

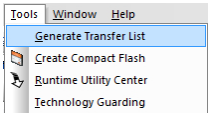

Figure 11: Generating a transfer list

Creating a USB remote install structure

The service tool Runtime Utility Center is now started by selecting <Tools> / <Runtime Utility Center> in Automation Studio. "Create a remote install structure" must be selected from the selection dialog box.

The USB stick can be selected in the configuration dialog box. It is also possible to decide if Automation Runtime should be updated along with the application. A version check prevents the target system from being updated unintentionally.

The remote install structure is copied to the USB stick by pressing the "Start" button. The system can now be updated using the newly set-up USB stick.

4 This applies to control systems using Default Automation Runtime.

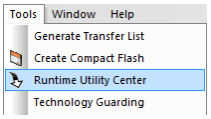

Figure 12: Start Runtime Utility Center

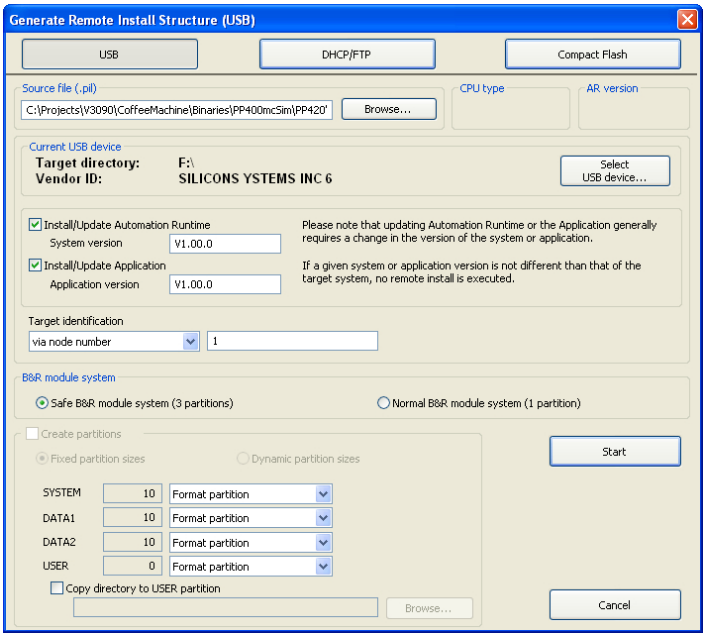

Figure 13: Runtime Utility Center install dialog

Many target systems have a reset button. Using the reset button, the system can be restarted and the operating mode can be changed. In the user's manual for your target system, you can find out how this can be used to put the control system into BOOT mode.

Installing Automation Runtime via the online connection and searching for the target system via SNMP is also possible in BOOT mode[5].

Diagnostics and service \ Service tool \ Runtime Utility Center \ Creating a list / data medium \ Remote install structure creation

5 Beginning with Default Automation Runtime V3.06, searching for the target system via SNMP is also supported for SG4 systems in BOOT mode.

2.4 Modifying and updating Automation Runtime

If new software versions are available for the Automation Runtime, Visual Components, motion or CNC runtime environments, then it is possible to change these software versions in the Automation Studio project for the configuration that is active.

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

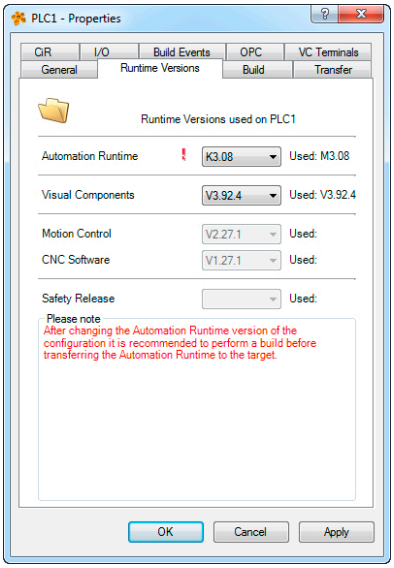

Figure 14: Changing runtime versions

Project management \ Changing runtime versions

Changing 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, the project must be rebuilt before it is transferred.

Upgrading from 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 to be used in **SystemROM** target memory (default) since UserROM is deleted once the new version of Automation Runtime has been transferred. After 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.

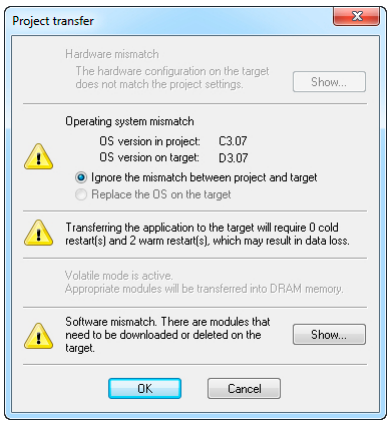

Figure 15: Version conflict detection

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. It can then 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 \ Generating an installation package

3 Memory management

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

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

3.1 Logical grouping of the Flash memory

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

In the software configuration, each module is automatically assigned to target memory – which is divided into **User ROM** and **System ROM**. This is purely a logical grouping because the data is stored on the same data storage medium.

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[6] where Automation Runtime and the system modules for the project are stored.

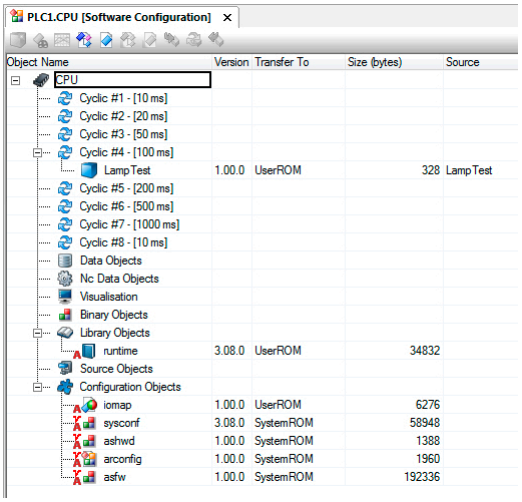

Figure 16: Target memory for .br modules

Figure 17: CompactFlash = System ROM + User ROM

6 The logical structure in User ROM and System ROM also applies for systems with integrated application memory.

3.2 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.

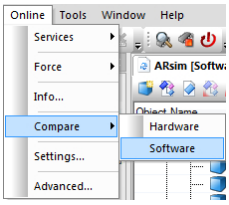

Figure 18: Opening the software comparison

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

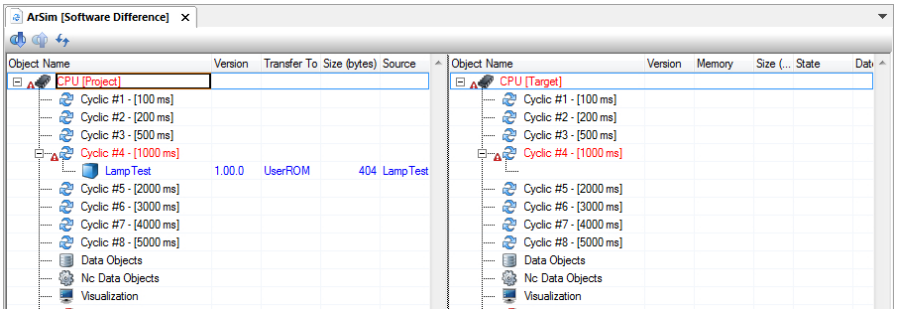

Figure 19: Opening the online software comparison window

Exercise: Delete the modules in User ROM

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

Then check which objects are still on the controller using the online software comparison.

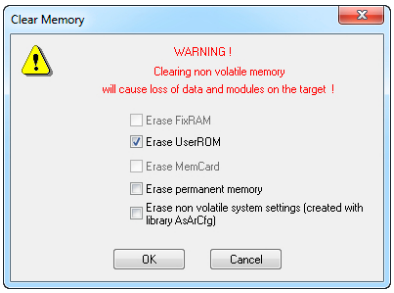

Figure 20: Window for clearing memory: User ROM selected

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

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

When User ROM is cleared, the system automatically restarts and switches to diagnostic mode.

This briefly interrupts the online connection.

3.3 Used RAM

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 a **DRAM** , but an **SRAM** or **FRAM** is optional.

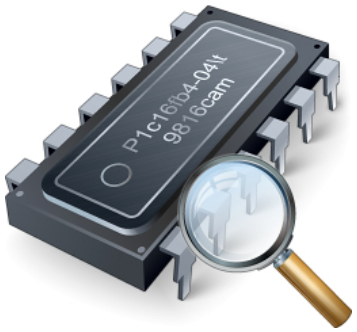

Figure 21: DRAM, SRAM and FRAM

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 3.5.3 "Initializing variable memory" on page 22) and is also responsible for initializing this memory area.

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

FRAM: Newer controller models are produced without a battery. The FRAM saves data on a nonvolatile electronic memory type based on crystals with ferroelectrical properties, as opposed to SRAM and DRAM. This allows data to be retained even after a power failure. No battery is required for this.

3.4 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 MEMxInfo() function block in the BRSystem 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.

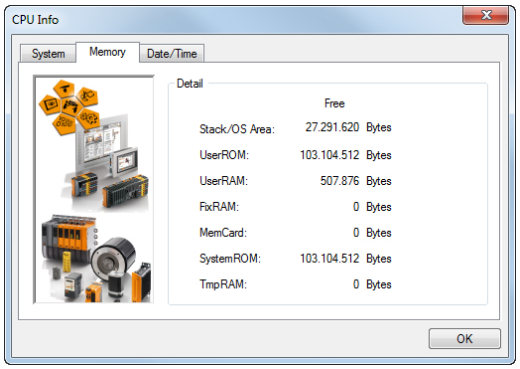

Figure 22: Reading information from the CPU

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.

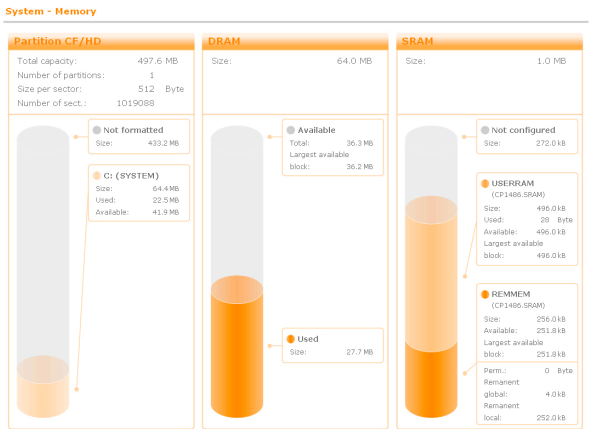

Figure 23: Memory allocation in SDM

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

The project, which was created while working with TM210 – Working with Automation Studio, must be transferred to the target system.

The target system must be connected to Automation Studio via an Ethernet connection. Determine the memory allocation on the target system using one of the methods described previously.

It should be determined what memory types exist on the used target system and how they are used.

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

3.5 Global and local variables

Variables are symbolic programming elements whose data type determines their size and structure. When the project is built, variables are assigned to a specific location in memory.

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

Programming \ Variables and data types

3.5.1 Local variables

Local variables are defined as having a local scope within a specific program and cannot be used in other programs.

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

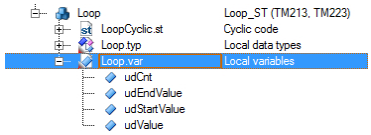

Figure 24: Local variables in the "Loop" program

Exercise: Create a "Loop" program using local variables

In the project's Logical View, create a new Structured Text program called "Loop" with the following local variables.

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

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

```
PROGRAM _CYCLIC
    FOR udCnt := udStartValue  TO udEndValue DO
        udValue:= udValue + 1;
    END_FOR
END_PROGRAM
```

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

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

Exercise: Multiple assignment of a program in the software configuration

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

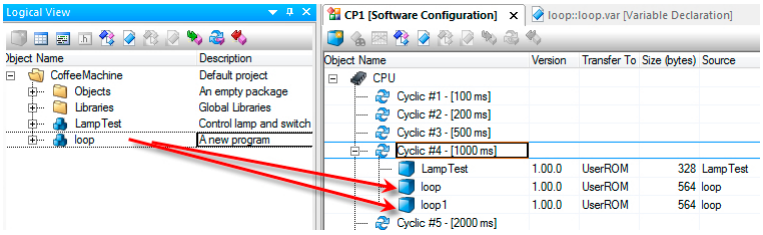

Figure 25: Multiple assignment of programs

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

The Watch window for both instances of the "Loop" program can be opened by selecting it from the shortcut menu in the software configuration or in the Logical View. A selection dialog box appears when opening the Watch window via the Logical View with regard to the variables you would like to open the display for.

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

Once all variables have been added to the Watch window for the "Loop" and "Loop1" tasks, a separate variable value is displayed in each task that only applies to that task.

Changing a local variable in one task only affects that task – the local variable values for the other task do not change.

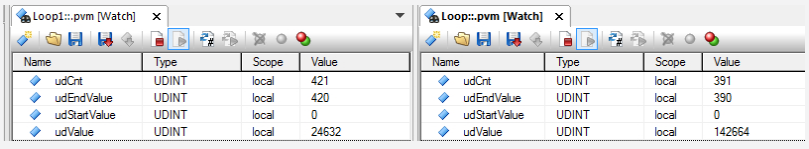

Figure 26: Watch window for both tasks

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

A corresponding warning is output during the build process.

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

Programming \ Variables and data types \ Scope of declarations

3.5.2 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. Because of this, they can be used in every program.

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

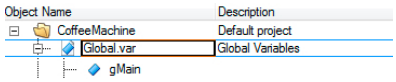

Figure 27: Global variables

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

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

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

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

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

gMain := gMain + 1;

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

If the **"gMain"** variable is added to the Watch window for the **"Loop"** and **"Loop1"** tasks, writing a value to it in one of the tasks will also immediately change its value in the other task.

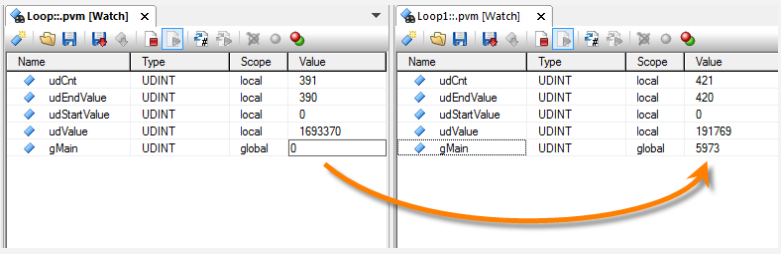

Figure 28: Writing to global variables

Programming \ Variables and data types \ Scope of declarations

Programming \ Editors \ Configuration editors \ Variable mapping

3.5.3 Initializing variable memory

By default, "0" is written to variables during initialization. A different initialization value can be specified in the variable declaration, however.

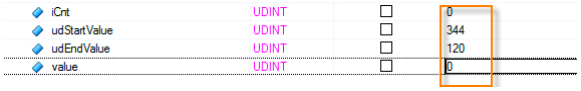

Figure 29: Initializing variables in "Loop.var"

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

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

Exercise: Initialize the "udEndValue" variable

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

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

In the Watch window for the **"Loop"** and **"Loop1"** tasks, initialize the **"udEndValue"** variable with the value 50. During runtime, this value can be changed to any other value.

3.5.4 Constants

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

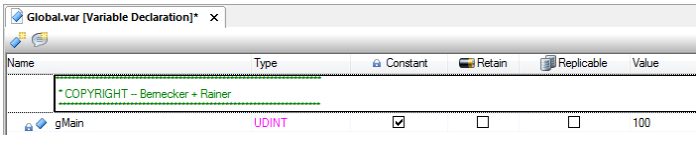

Figure 30: Declaring constants

Exercise: Global variable "gMain" as a constant

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

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

An error message will appear during the build process since **"gMain"** is being written to.

This is not permitted for constants! In order for the program to operate without errors, the **"gMain"** variable is not permitted to 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

3.5.5 System behavior of Retain variables

When the target system is booted, Automation Runtime is started automatically and begins running through all of the boot phases (4.1 "Starting Automation Runtime" on page 26).

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

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

- Target system with battery-backed SRAM or FRAM
- Variables configured as "Retain"

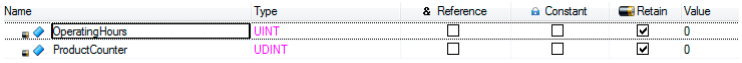

Figure 31: Battery-backed variables

Different target systems have varying amounts of SRAM or FRAM available.

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

Examples of the use of Retain variables:

- Operating time counters
- Number of product errors
- Product IDs
- Type counters
- Condition and status of the machine at the time of a power failure
- and much more...

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) should be stored on a Flash memory.

Warm restart

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

- Power on after power failure
- 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** 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:

- Restarting after exchanging the CompactFlash card
- Restarting after clearing UserROM
- Performing a cold restart from Automation Studio
- Restarting when the battery backing the Retain variables in the SRAM is defective

Permanent variables

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

Examples of data to be stored permanently:

- Operating time counters
- All data that isn't written to a Flash memory but should also be received when deleting Retain variables

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

Permanent memory is never formatted or written to by the system. It is the full responsibility of the user to manage the values of variables while the application is starting.

Permanent variables without a proper foundation can result in varying and inconsistent program behavior.

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

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

Programming \ Variables and data types \ Variables \ Variable remanence

Programming \ Editors \ Configuration editors \ Permanent variables

Automation software \ Technology Guarding

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

4 Runtime performance

This section explains the runtime behavior of the Automation Studio application with respect to Automation Runtime.

Several of the diagnostic 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.

The following tasks are carried out during this procedure:

- Hardware check
- Hardware/firmware upgrade, if necessary
- Check of BR modules
- BR modules copied from ROM to DRAM
- Retain variables copied to DRAM
- Variable memory set to initialization value
- Execution of initialization subroutine
- Activation of cyclic programs

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.

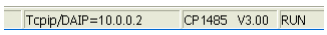

Figure 32: X20CP1485 in RUN mode

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

Automation Runtime operating states

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

Once one phase has completed successfully, a check of the next phase takes place.

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

Figure 33: Boot phases

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

Operating state	Conditions that lead to this system state
BOOT	• If no CompactFlash is inserted • If there is no operating system on the CF card • Node switch "00"[7]
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  Figure 34: RUN mode

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

Automation Runtime boot phases

While the controller is booting, it may run through a series of intermediate steps before reaching the RUN state. These states are also indicated in the Automation Studio status bar. Nevertheless, these phases are usually very short.

System state	Description
STARTUP	Initialization procedures relevant to the operating system are performed during this phase.
FIRMWARE	Firmware is updated during this phase.
INIT	The application's initialization subroutines are executed during this phase.

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

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

4.2 Program initialization

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

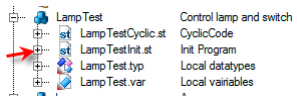

Figure 35: Program initialization

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

4.2.1 Running the initialization subroutine

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

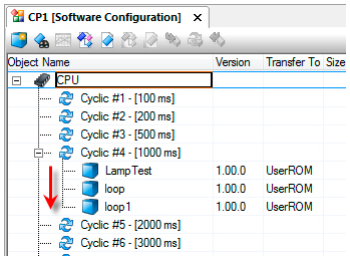

Figure 36: Executing initialization subroutines

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

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

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

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

4.2.2 Function calls in an initialization subroutine

When using functions, it is important that the function returns 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.

4.3 Cyclic program sequence

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

Programs in the software configuration are called tasks. The controller only executes programs that are assigned in the software configuration.

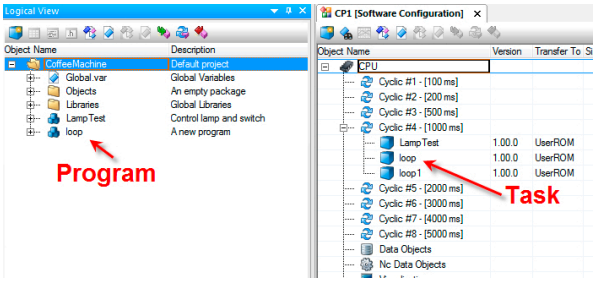

Figure 37: Relationship between programs and tasks

4.3.1 Multitasking operating system

Automation Runtime is a **deterministic**, **real-time multitasking** operating system.

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

4.3.2 Task classes and cycle times

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

Up to eight task classes with a configurable cycle time 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 need 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 sequence would look like this:

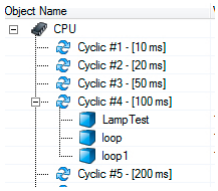

Figure 38: Cycle time

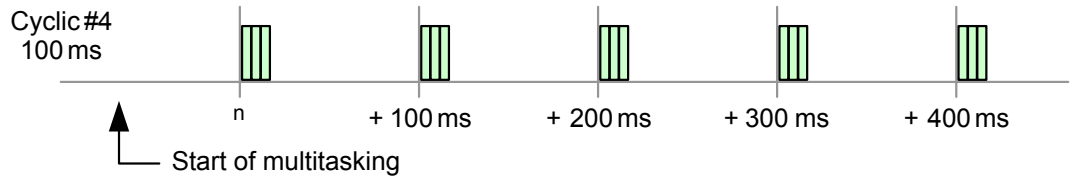

Figure 39: Tasks are called again during each cycle.

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

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

4.3.3 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.

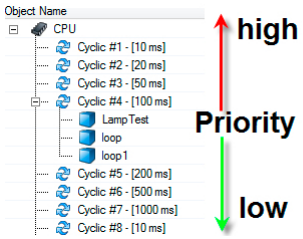

Figure 40: Task class priority

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.

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

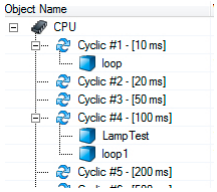

Figure 41: Different task classes

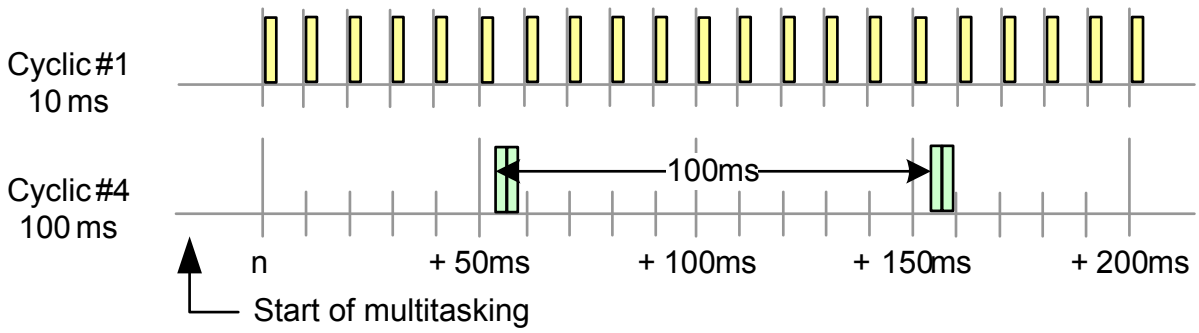

Figure 42: Executing tasks in different task classes

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

Exercise: Move the "Loop" task to 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) Move the **"Loop"** task from task class #4 to task class #1.
3) Open the Watch window for both tasks.
4) Monitor the **"udValue"** variable.

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

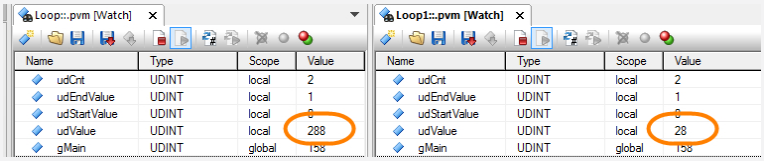

Figure 43: Cycle times in different task classes

4.3.4 Starting task classes

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

The starting point is always the 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 performance \ Start of cyclic tasks

4.3.5 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 for the following tasks:

- Online communication
- Visual Components application
- File access

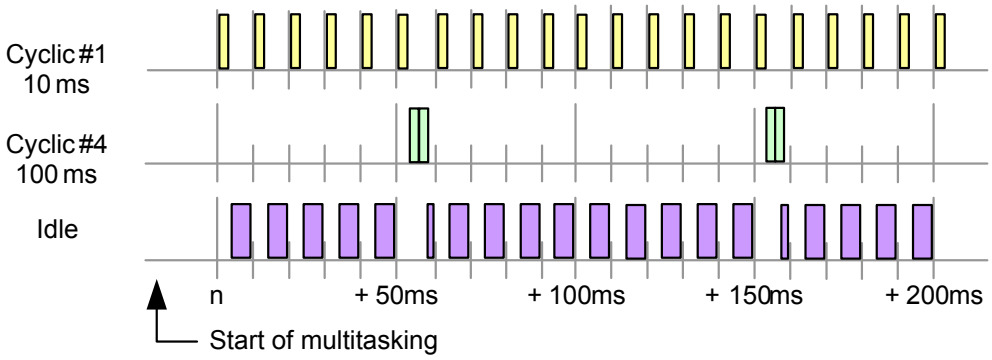

Figure 44: Idle time in the cyclic system

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

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

4.3.6 Handling I/O mappings - I/O scheduler

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

The I/O channels are mapped using the configurable cycle of the I/O fieldbus system being used.

This cycle can be configured in the properties of the respective fieldbus. To do so, select **<Configuration>** from the X2X Link interface's shortcut menu.

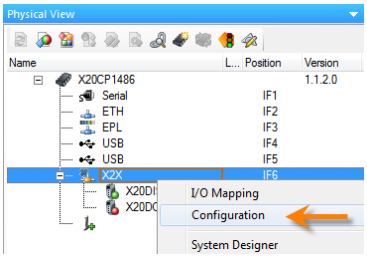

Figure 45: Opening the properties of the X2X Link interface

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

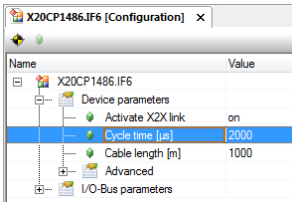

Figure 46: X2X cycle time setting

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

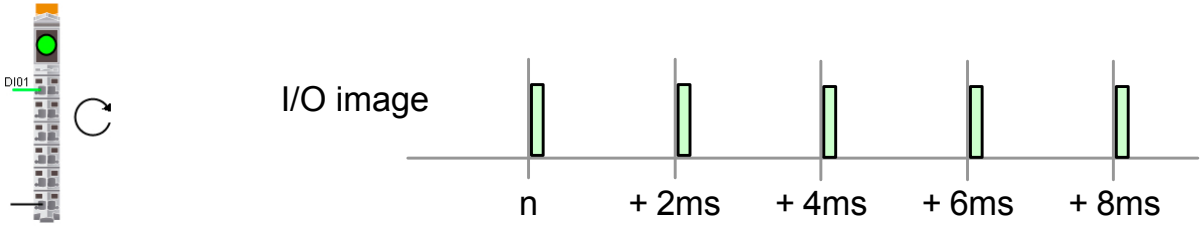

Figure 47: Reading the input image

For the application, this means that the input data is not older than the configured cycle time when the task class starts; the output data will be written using this image 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 under the "Timing" category located in the system configuration.

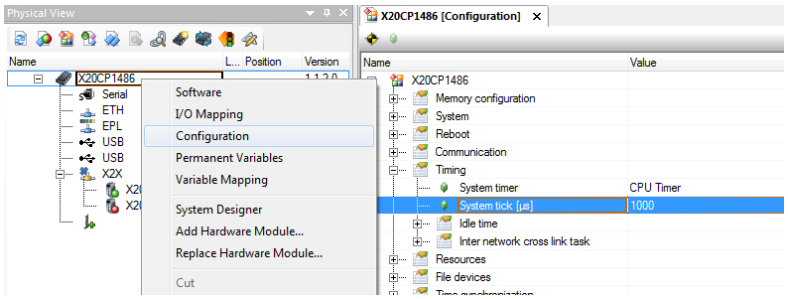

Figure 48: Configurable system tick

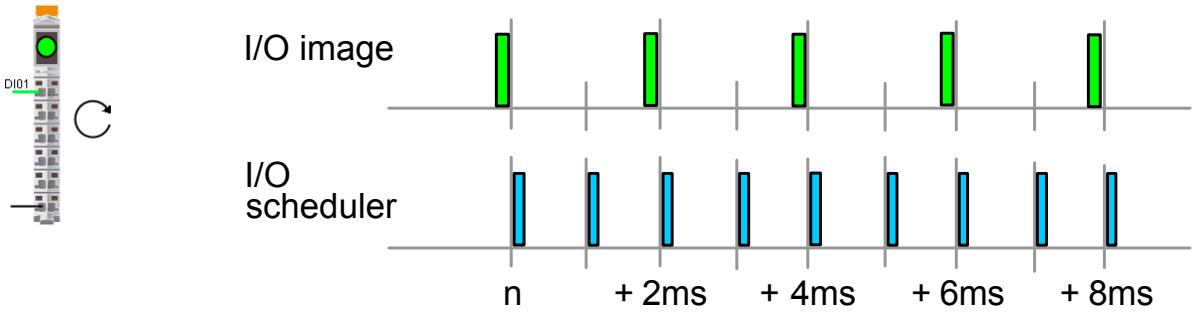

Figure 49: Reading the input image

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

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

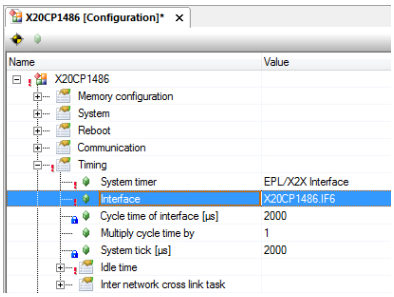

Figure 50: Synchronizing the system tick

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.

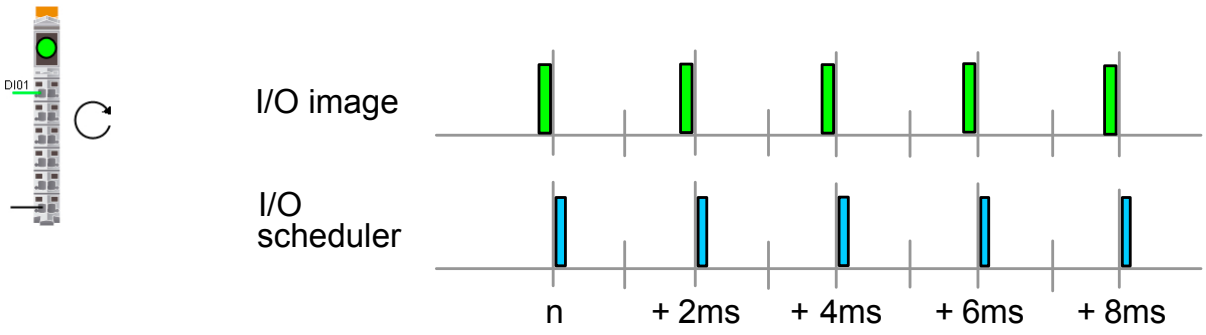

Figure 51: Reading the input image

I/O management assigns I/O data from the image 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 task class #1 every 10 ms, providing the tasks therein with the input image for the assigned variables (green arrow).

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

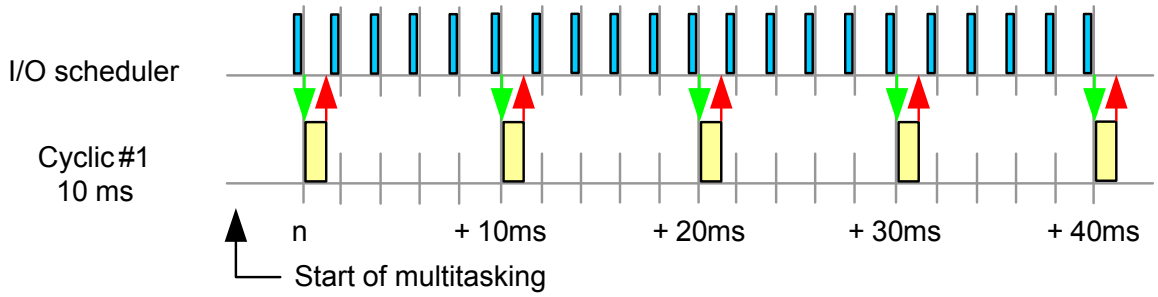

Figure 52: I/O handling for task class #1

Task class #4

The I/O scheduler starts task class #4 at n+50 / n+150, providing the tasks therein with the input image 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 image (red arrow).

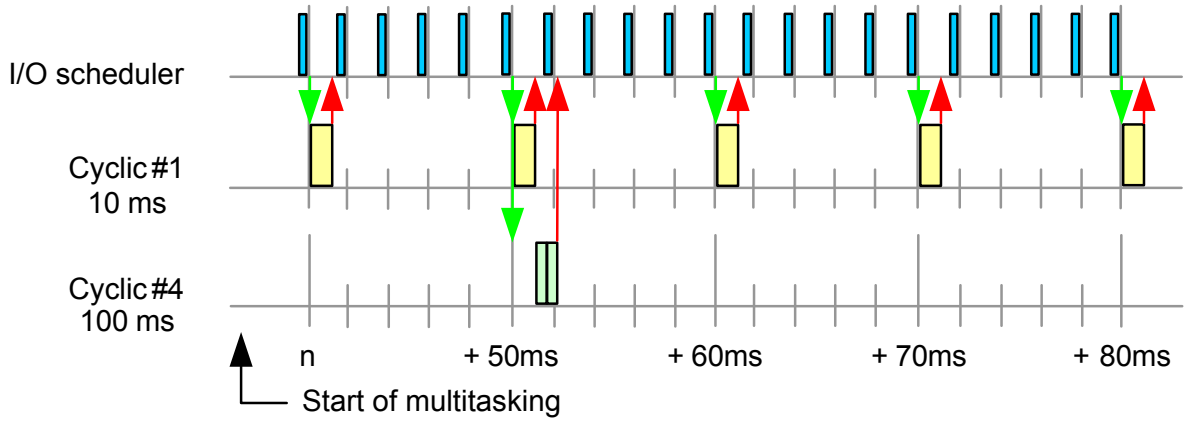

Figure 53: I/O handling for task class #4

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

4.3.7 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 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 tolerance can be configured to prevent the system from booting in service mode when the cycle time is exceeded.

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

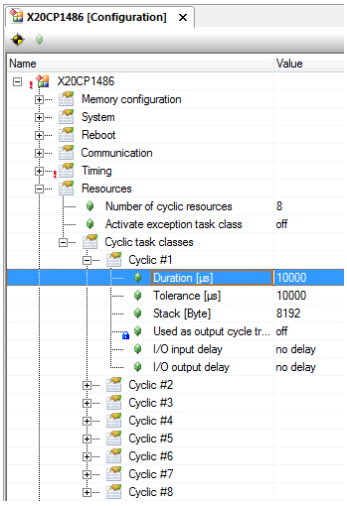

Figure 54: Configuring the cycle time and tolerance

If the configured runtime of a task class is exceeded, the next start of the task class is shifted to the beginning of its next regular cycle. This can lead to timing problems in the application.

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

Exercise: Trigger a cycle time violation

In the **"Loop"** task, which runs in task class #1, change the value of the **"udiEndValue"** variable in the Watch window to trigger a cycle time violation.

Use the **Profiler** to monitor the runtime behavior of the task and the available idle time. After triggering 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: Set the "udEndValue" variable to 50,000

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

The first step is to set the value of the "udEndValue" variable 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>** from the shortcut menu.

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

It is possible to switch the output by selecting the corresponding workbook.

Figure 55: Changing the Profiler view

By default, Automation Runtime records all runtime behavior.

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

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

To open the window for making configurations, click the **"Configuration"** icon in the Profiler toolbar.

Make the configurations shown in the following images:

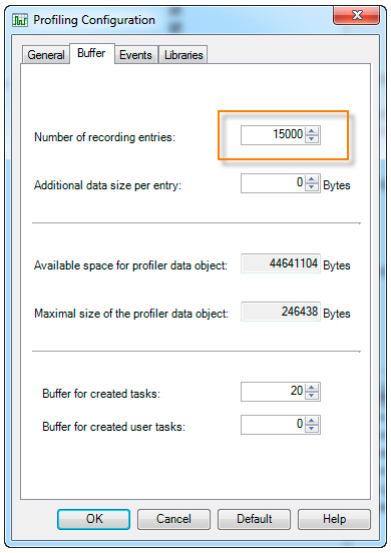

Figure 56: Profiler configuration 1

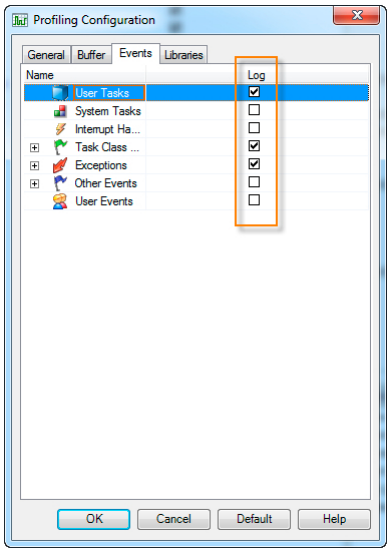

Figure 57: Profiler configuration 2

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

Halt 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 **"Graph"** icon.

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

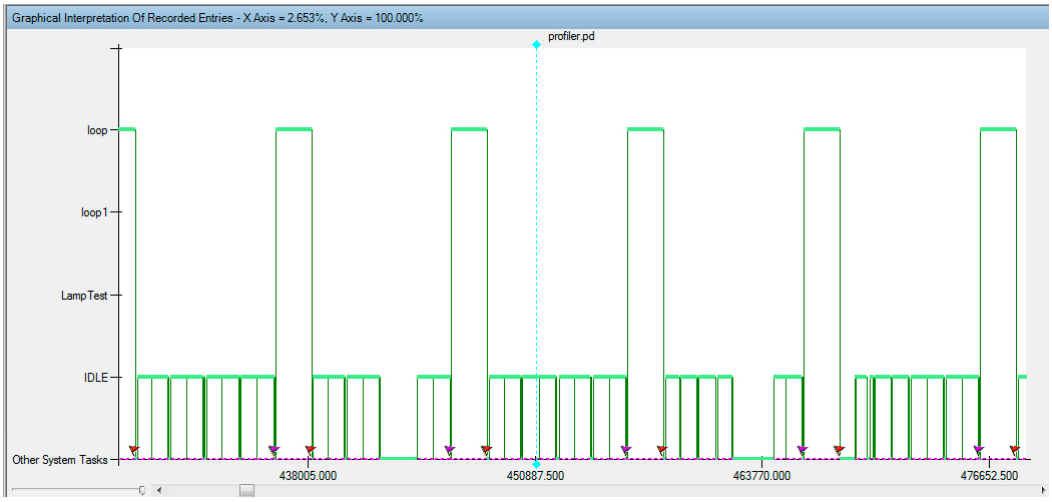

Figure 58: Result of the first Profiler measurement

Exercise: Check the Profiler recording

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

Diagnostics and service \ Diagnostics tools \ Profiler

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

- In this step, restart the Profiler using the "Start" icon in the Profiler toolbar.
- Increase the value of the **"udEndValue"** variable in steps (e.g. 10,000).
- 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 has already reached 16.5 milliseconds.

The tolerance – defined in the properties of task class #1 as 10 milliseconds – now takes effect.

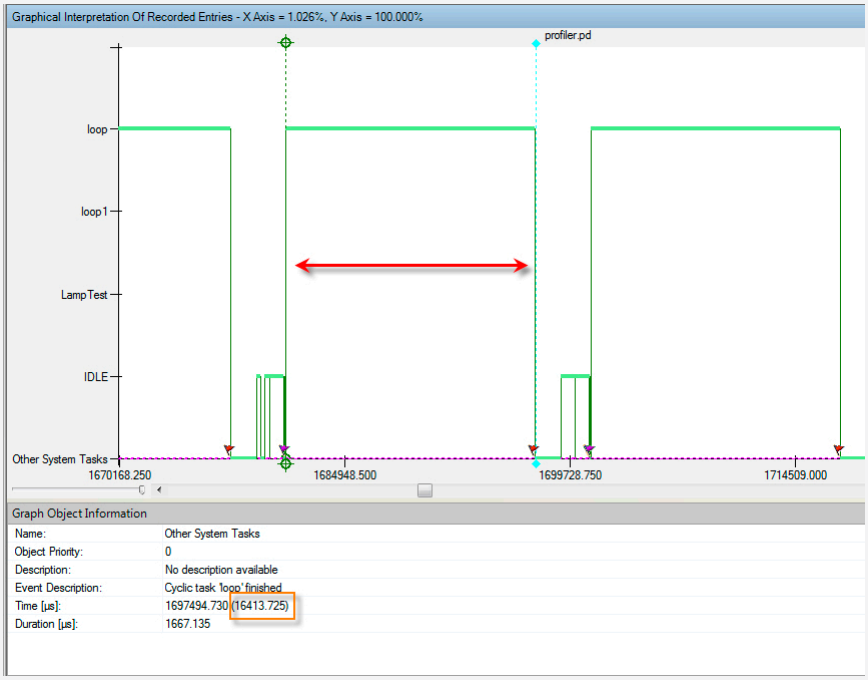

Figure 59: Exceeding the cycle time, effect of tolerance

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

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

4.3.8 Cycle time violation

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

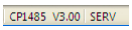

Figure 60: X20CP1485 in Service mode

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

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

Diagnostics and service \ Diagnostic tools \ Logger window

The Logger is opened from the **Software configuration** by selecting **<Open>** / **<Logger>**.

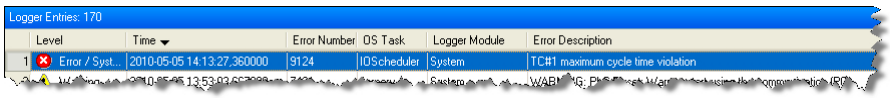

Figure 61: Analyzing the cause of the error in Logger

A cycle time violation in task class #1 is entered as the cause of the error.

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.

4.3.9 Responding to a cycle time violation in the application program

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

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

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

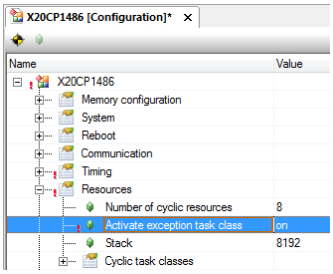

Figure 62: Enabling the exception task class

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

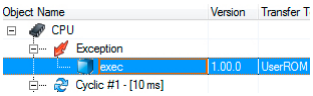

Figure 63: Configuring the exception task class

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

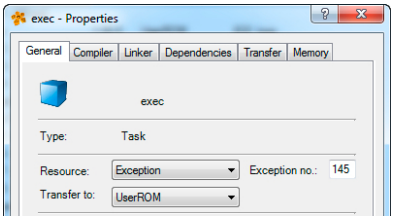

Figure 64: Configuring the exception task

A cycle time violation (exception no. 145) occurring at runtime will call this task, which contains the response of the application.

Refer to the help system for the different exceptions possible.

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

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

4.4 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.

The I/O management in Automation Runtime is designed to meet the highest requirements for **response time** and **configurability**.

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

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

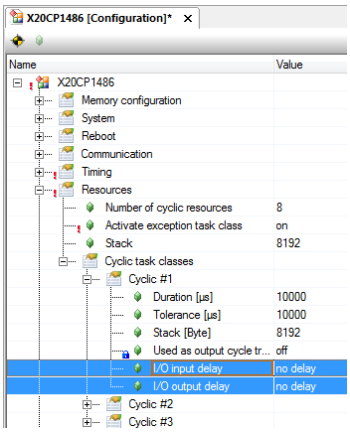

Figure 65: Configuration for executing output data

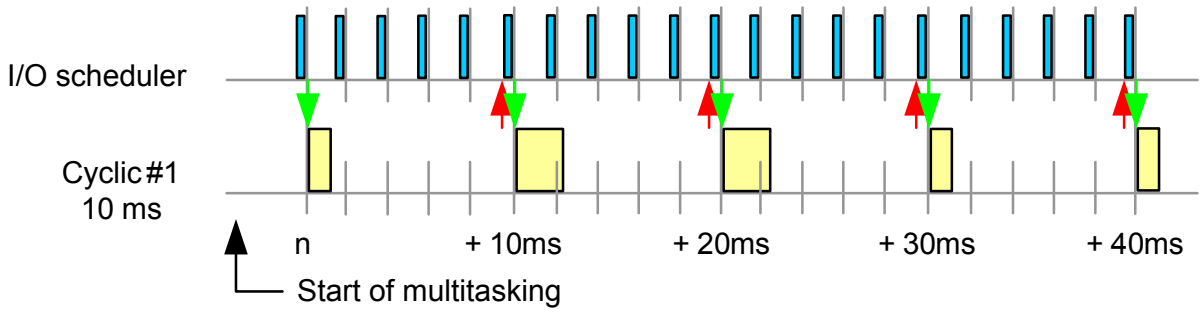

Figure 66: Jitter-free writing of output image at end of cycle

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

4.4.1 Startup

Booting the controller transfers the **active I/O configuration** to the I/O modules and initializes the interfaces and fieldbus devices.

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

4.4.2 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 startup, and the I/O mapping, which assigns the I/O data to the corresponding variables.

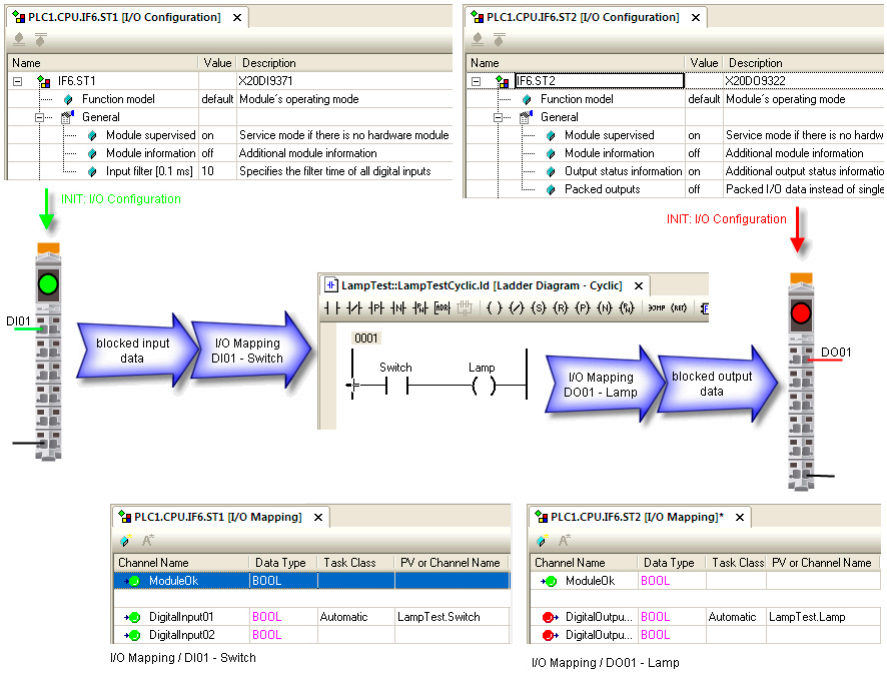

Figure 67: Basic I/O functionality

4.4.3 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 **<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.

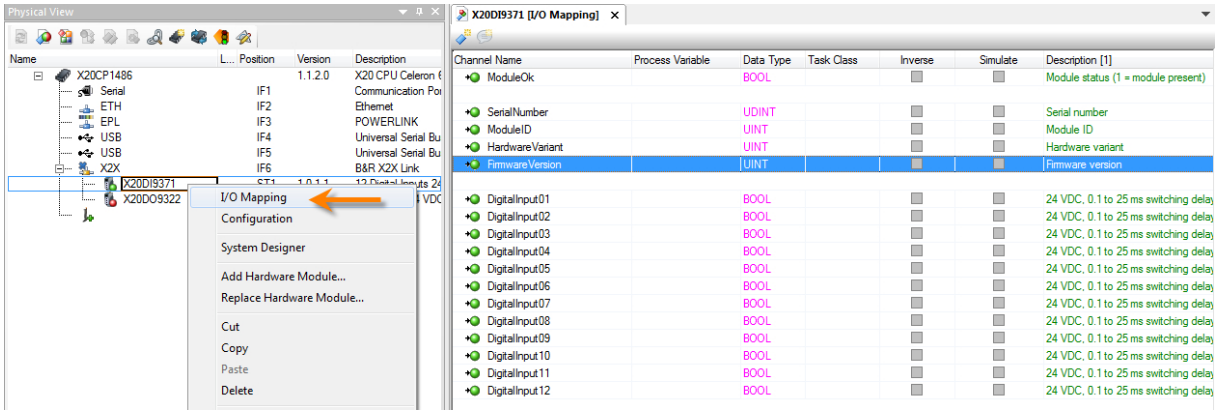

Figure 68: I/O mapping of a digital input card

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

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

Programming \ Editors \ Configuration editors \ I/O mapping

4.4.4 The 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 continues to open up more and more options and operating modes in which they can be used.

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

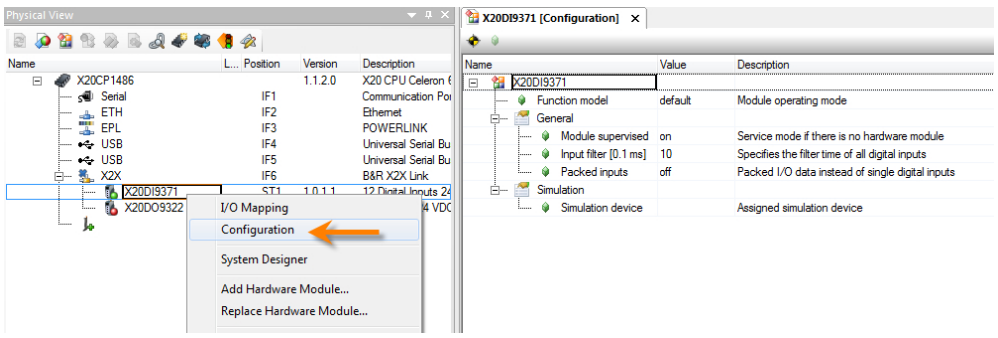

Figure 69: I/O configuration of a digital input module

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

4.4.5 Error handling for I/O modules

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

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

The **"Module supervised"** property in the **I/O configuration** can be used to enable (default setting) or disable the monitoring of an I/O module.

Monitoring active

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

- The module defined for a slot is not present (not connected).
- The module physically connected in a 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

If monitoring is disabled, mapping a variable to the **"ModuleOK"** channel makes it possible to react to possible error situations from the application.

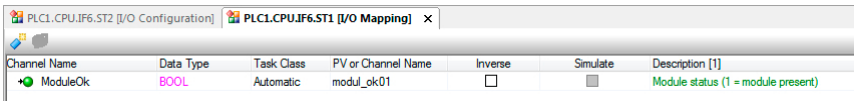

Figure 70: Module monitoring via the application

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 executing and changing one or more programs.

5.1 Task interrupted by another task

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

Exercise: Interpret the task classes in a system using the Profiler

Using the **"Loop"** and **"Loop1"** tasks, open the Watch window to change the final value of the **"udiEndValue"** variable so that the **"Loop1"** task is interrupted exactly **twice** by the **"Loop"** task.

Set the starting value for the **"udiEndValue"** variable in the **"Loop"** task to 2,000.

The Profiler measurement looks like this:

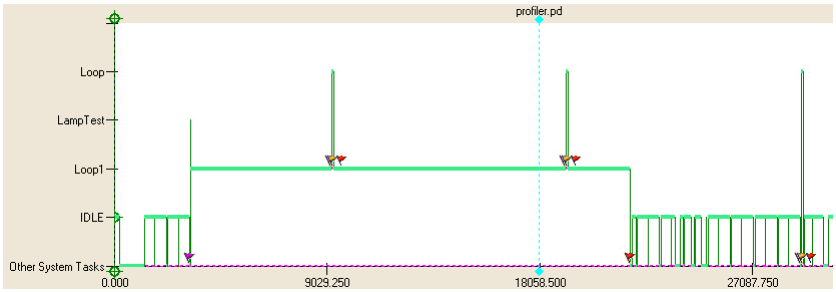

Figure 71: Interrupting tasks

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

5.2 Synchronizing the task class with the I/O cycle

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

This is not sufficient, however, for processing high-speed I/O data via POWERLINK or X2X.

As described in 4.3.6 "Handling I/O mappings - I/O scheduler", data retrieval can be synchronized with the fieldbus I/O cycle. In order for the data to also be processed in the I/O retrieval cycle, the task class must be called more quickly and more often.

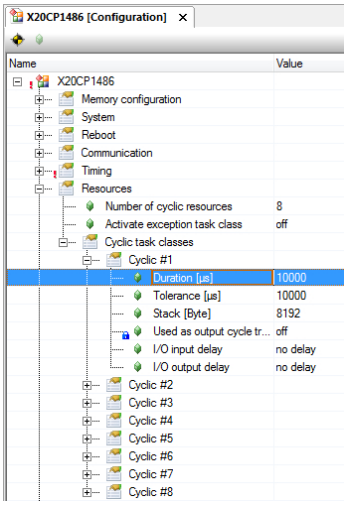

Figure 72: Configuring the cycle time for task class #1

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:

- Download during development – No effect on the live production system
- Download during runtime – Effects on the live 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.

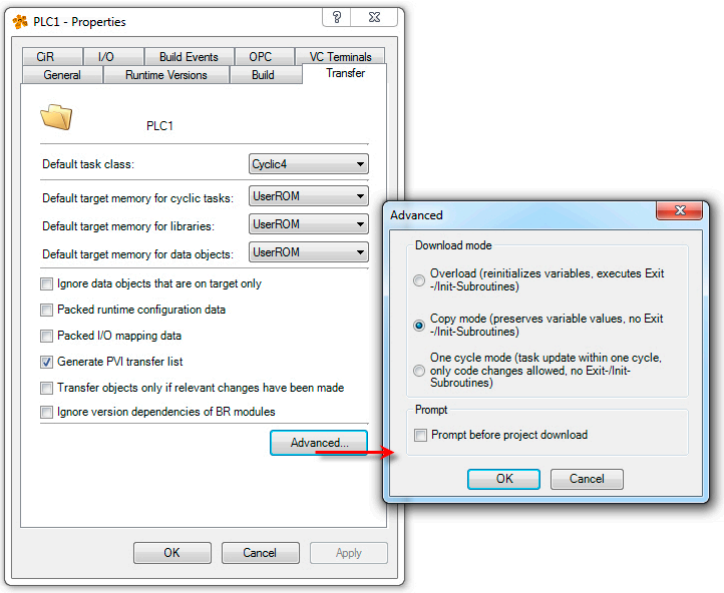

Figure 73: Setting the download mode in the Configuration View

Download during development – Overload mode

Overload mode should be used when frequent changes are made to the program code during development. This is the default download mode for a new configuration.

Overload mode is not suitable for a live production system.

Download during runtime – One cycle mode

This download mode is suitable for making changes to a live production system. In this mode, only one program is exchanged per cycle. This guarantees the fastest possible changeover times for the live production system.

Download during runtime – Copy mode

This download mode is suitable for making changes to a live production system. Programs are transferred over multiple cycles.

Real-time operating system \ Target systems \ SG4 \ Download \ Copy mode

- Overload
- One cycle mode
- CopyMode

5.4.2 The exit subroutine

A task's exit subroutine is called when the task is uninstalled (deleted).

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

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 the optimal usage of resources.

Figure 74: Automation Runtime target systems

The timing of the application can be configured by adapting 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.

7 Appendix

7.1 "Loop" sample program

"Loop" program

The "Loop" program is written in the Structured Text programming language. The program contains a FOR loop with variable starting and ending values. By increasing the ending value, it is possible to increase the CPU load and trigger a cycle time violation.

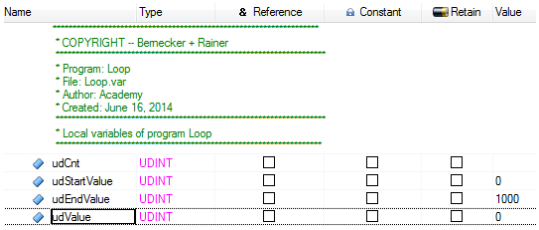

Figure 75: "Loop" variable declarations

```
PROGRAM _INIT
    udEndValue := 50;
END_PROGRAM

PROGRAM _CYCLIC

    (* implementation of program Loop *)
    FOR udCnt := udStartValue TO udEndValue DO
        udValue := udValue + 1;
    END_FOR

    gMain := gMain + 1;

END_PROGRAM
```

Figure 76: Implementing "Loop" in Structured Text

Training modules

TM210 – Working with Automation Studio
TM213 – Automation Runtime
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
TM540 – Integrated Safe Motion Control
TM600 – Introduction to Visualization
TM610 – Working with Integrated Visualization
TM630 – Visualization Programming Guide
TM640 – Alarm System, 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
TM923 – Diagnostics and Service with Automation Studio
TM950 – POWERLINK Configuration and Diagnostics

TM1010 – B&R CNC System (ARNC0)
TM1110 – Integrated Motion Control (Axis Groups)
TM1111 – Integrated Motion Control (Path Controlled Movements)
TM261 – Closed Loop Control with LOOPCONR
TM280 – Condition Monitoring for Vibration Measurement
TM480 – The Basics of Hydraulics
TM481 – Valve-based Hydraulic Drives
TM482 – Hydraulic Servo Pump Drives