TM223

Automation Studio diagnostics

PERFECTION IN AUTOMATION
www.br-automation.com

Prerequisites and requirements	
Training modules	TM210 – Working with Automation Studio TM213 – Automation Runtime
Software	Automation Studio 4.2.5
Hardware	X20CP1586

Table of contents

1 Introduction

Automation Studio offers a plethora of tools to support you during development and any diagnostics tasks that occur later on. The integrated diagnostic tools help to create a system overview including the detailed recording of system behavior. Automation Studio is therefore the ideal tool for programming, commissioning and diagnostics.

The diagnostics process begins by selecting the right tool for the application or situation at hand. This training module shows how the individual diagnostic tools can be used individually or in combination. Numerous tasks contribute to better understanding and provide examples for practical use.

Automation Runtime provides access to diagnostics information. For this reason, a lot of diagnostic data is available directly in the System Diagnostics Manager. This can be called using a web browser. Data is also available in the control application using library functions.

Figure 1: Automation Studio diagnostics

The diagnostics tools for integrated motion control are described in the training module "TM410 – Working with Integrated Motion Control". The diagnostics tools for integrated safety are dealt with in the training module "TM510 – Working with SafeDESIGNER".

1.1 Learning objectives

This training module uses selected examples illustrating different diagnostic possibilities during programming, commissioning and servicing to help you learn how to work with the various diagnostic tools.

- You will learn the criteria for selecting the correct diagnostic tool.
- You will learn how to evaluate and store general system information.
- You will learn how to observe and record process values.
- You will learn about the possibilities for diagnosing the system and application.
- You will learn which Automation Runtime configuration options are relevant to which diagnostic tools.

1.2 Symbols and safety notices

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

2 The correct diagnostic tool

Selecting the correct diagnostic tool makes it possible to quickly and effectively localize a problem.

Analyzing irrelevant data yourself or sending it to someone else for examination can lead to substantial delays in finding a solution.

The Logger can be used to recognize a cycle time violation. However, the cause of the cycle time violation would not be best diagnosed by using the Logger in this case.

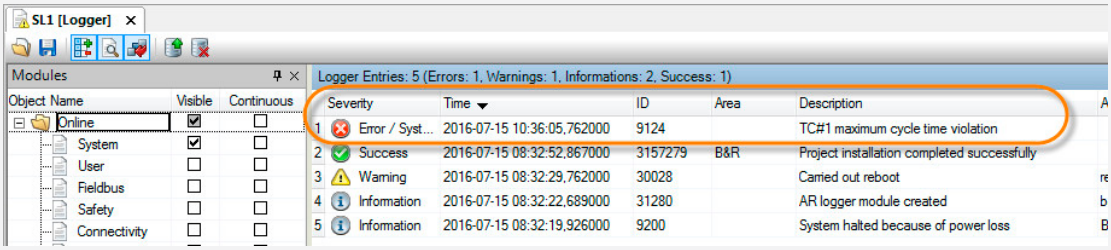


Figure 2: Cycle time violation in the Logger window

The cause of the error in the Logger will be given as a cycle time violation in Task Class #1. The backtrace data also refers to a task where the cycle time violation occurred.

Situation 1

Since a multitasking system allows one task to be interrupted by a higher priority task, it is possible that this higher priority task is the cause of the cycle time violation. This is because the higher priority task has a longer execution time in this cycle, which means that the task used in the Logger no longer has a chance to be completed within its configured cycle time or tolerance.

Situation 2

Several tasks are executed one after another cyclically in the same task class. If one of the previous tasks takes longer to complete, then the task shown in the Logger will also not be the cause of the cycle time violation.

In both cases, the Logger window would be the **wrong diagnostic tool** to determine the error. This problem can only be detected using the Profiler ("Perform runtime measurements in the Profiler" on page 27), which displays the chronological sequence of individual tasks as well as the time needed for them to complete.

2.1 The checklist

A checklist doesn't just help when trying to analyze a problem during servicing; it is also very useful beforehand while programming.

The information collected here can help those called in later to troubleshoot errors to solve problems more quickly by providing the actual data instead of requiring further inquiries.

Figure 3: Checklist

There are a number of different ways to analyze a problem. Combining different localization and analysis strategies can considerably increase effectiveness when trying to locate errors.

The methodology of locating errors

The methodology used when searching for errors is extremely important as it allows the available tools to be applied selectively. This requires asking a series of questions that suit the actual environment, beginning with the machine and progressing to the controller.

- Analyzing the problem
- Eliminating other possible errors
- Measuring signals

With an optimal overview, specific areas can be isolated and analyzed in greater detail.

Environment and general conditions

Immediately applying various analysis strategies is not necessarily the best idea since it is possible that the problem has nothing to do with the machine, but rather the environment it is in.

Looking at general conditions during runtime (shift or product/batch changes, clock changes (e.g. daylight savings time), room temperature(s), replaced sensors, user actions, etc.) allows the error search to be narrowed.

Once potential errors in the machine's environment have been ruled out, analyzing the Automation Studio project itself can begin.

One-time problem or a recurring error?

If errors can be reproduced during certain actions, they can be analyzed in the program code using the debugger.

Program errors that seem to occur due to no particular action or with no regularity are extremely difficult to reproduce, and even this reproduction is not always reliable.

Errors that do not occur cyclically can be analyzed more easily by making the necessary preparations in the application (e.g. automatically enabling the Profiler).

Error in program or program sequence?

Runtime errors occur if certain general conditions are not taken into consideration when the process is executing.

Examples of errors when running programs:

- Division by zero
- No evaluation of return values from functions
- Overflow when accessing array elements (e.g. loop counters)
- Access to non-initialized pointer

What information is needed when relaying the problem?

If additional people are needed to help in analyzing a problem, it is necessary to provide a detailed description of what it is.

- Detailed checklist filled in
- What actions have already been taken?
- What environmental conditions can be ruled out?
- Can the problem be reproduced in an office environment?

The more detailed the actions taken have been documented and information collected, the better the chances that the problem can be found (see also training manual "TM920 – Diagnostics and service for end users").

Software versions (also include any installed upgrades)		
Software	Version	Description, remark
•	•	•
•	•	•
Hardware used (also include installed operating systems)		
Model number	Revision Serial number	Description, remark
•	•	•
•	•	•
Can the problem be reproduced, or did it occur only once?		
•		
What actions need to be taken to reproduce the problem?		
•		
When did the problem begin? Have there been any changes in the software and/or hardware configuration or machine environment since then?		
•		
In what state is the CPU, and what is the LED status of the accompanying components?		
•		
What information has been loaded from the CPU for analysis purposes (no screenshots!)? e.g. Logger, Profiler data etc.		
•		

Table 1: Checklist for relaying information

2.2 Overview of diagnostic tools

Automation Studio provides appropriate tools that can handle diagnostics during programming, commissioning and servicing.

Only by selecting the right diagnostic tool is it possible to accurately and quickly access the necessary information.

The Automation Studio help is a constant companion during development, commissioning and servicing and it provides detailed information about the various diagnostic tools.

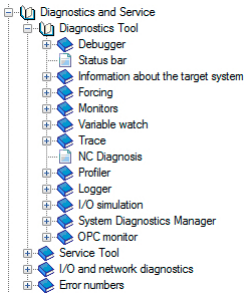


Figure 4: Diagnostics and service in the help system

Exercise: Open the help documentation for the diagnostic tools

The diagnostic tools are covered in section "Diagnostics and service". Get an overview of the structure of the help in this area.

Diagnostics and service

Requirements for the completion of exercises in this training module

The descriptions and images in this chapter refer to the X20 CPU-based project designed in both training manual TM210 (Working with Automation Studio) as well as TM213 (Automation Runtime). The following exercises can be done with any Automation Studio project.

Figure 5: X20 CPU

Recording data with the Profiler must be carried out on real hardware because no diagnostically conclusive measurement results arise in the Automation Runtime Simulation (ARsim).

3 Reading system information

System information can be read from the target system in Automation Studio as well as with the aid of an Internet browser.

Reading system information	
"Status bar" on page 9	Information about the status of the connection, Automation Runtime version and the operating state of the controller
"Information about the target system" on page 10	Display of memory information, battery status and date/time configuration options for the controller
"Online comparison" on page 10	Comparison of software versions in the project and on the controller. This function is also available for hardware. The modules configured in the project and actually present on the controller can be compared.
"Error analysis in Logger" on page 13	Displays events that occur on the target system at runtime.
"System Diagnostics Manager (SDM)" on page 43	System Diagnostics Manager (SDM) is a Web-based interface integrated directly into Automation Runtime. A standard Internet browser can be used to analyze important target system information.

Table 2: Reading system information

3.1 Controller operating status

A number of options are available in Automation Studio for evaluating the operating status of a controller:

- "Status bar" on page 9
- "Information about the target system" on page 10
- "Online comparison" on page 10

This information can also be read and displayed using System Diagnostics Manager ("System Diagnostics Manager (SDM)" on page 43).

3.1.1 Status bar

The status bar is located at the bottom of the Automation Studio window.

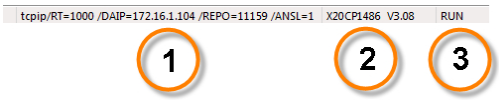


Figure 6: The status bar

The status bar includes the following information:

1.Connection settings

2.CPU type and Automation Runtime version

3.Operating state of the controller

Project management \ The workspace \ Status bar

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

3.1.2 Information about the target system

With an active online connection you can query information about the target system using **<Online> / <Info>** in the main menu or using **<Online Information...>** in the Physical View shortcut menu.

The target system's clock can be set manually or synchronized to that of the PC in this dialog box.

The information dialog box contains the following elements:

- Test the status of the internal backup battery
- Target system type and Automation Runtime version
- Hardware node number
- Available memory on the target system
- Date and time of the target system.
- Setting the date and time

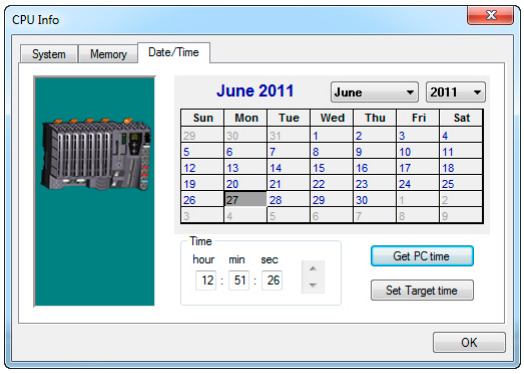


Figure 7: Setting the date and time on the target system

In order to get diagnostically conclusive data, it is necessary for the time and date on the controller to be correct. Date and time are set in the online information dialog box or by using the library functions. In addition, the controller has a configuration option for time synchronization with a time server.

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

3.2 Online comparison

To get an overview, it is necessary to check whether the hardware and software configurations in the opened project match those in the target system. This check is supported by the comparison functions in Automation Studio.

3.2.1 Online software comparison

The online software comparison can be used to compare the status and versions of tasks on the target system and compare them with the software configuration in the project.

The online software comparison is opened by selecting**<Online> / <Compare> / <Software>** from the main menu.

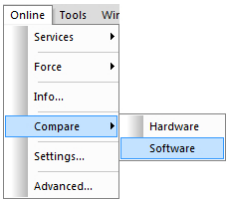


Figure 8: Enabling the online software comparison

The following information is analyzed:

- Comparison of tasks contained in the project with those on the target system
- Target memory where the task is running
- Operating state of individual tasks
- Versions and timestamp of the last build

A window with two columns is opened. The left side shows the elements configured in the software configuration, while the right side shows the configuration active on the target system.

In this example, the task "**LampTest**" on the target system has been stopped, whereas the task "**Loop1**" is not present on the target system.

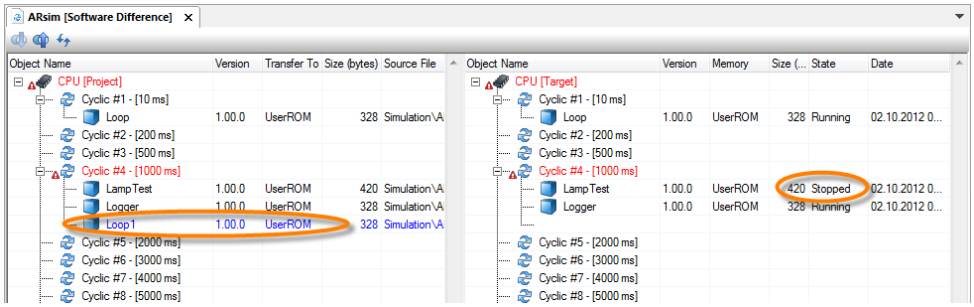


Figure 9: Online software comparison

Diagnostics and service \ Diagnostic tools \ Monitors \ Online software comparison

3.2.2 Online hardware comparison

The online hardware comparison can be used to compare the hardware configuration in the project with the one actually being used at runtime. The online hardware comparison can be activated by selecting **<Online> / <Compare> / <Hardware>** from the main menu.

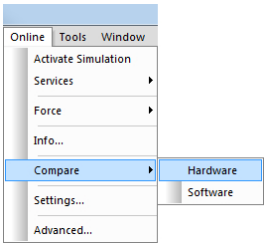


Figure 10: Opening the online hardware comparison window

The window is divided into two halves. The left side shows the hardware configuration in the project. The right side shows the hardware configuration which is used at runtime. Differences are indicated by a red warning triangle.

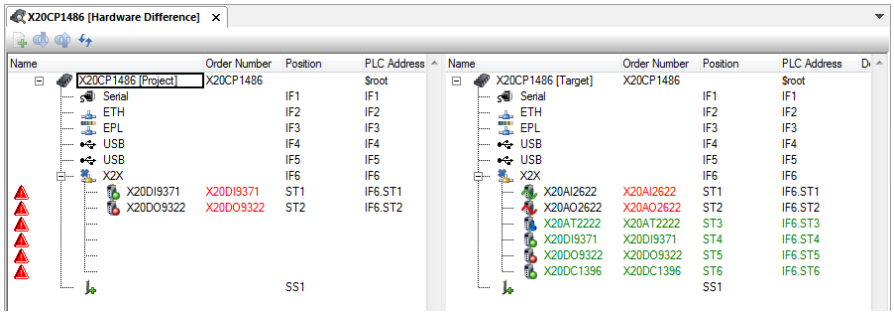


Figure 11: Online hardware comparison

In this configuration, two digital modules were configured on the X2X Link bus, but two analog modules are being used on the target system. All other identified modules have not been configured in the project.

Diagnostics and service \ Diagnostic tools \ Monitors \ Online hardware comparison

3.2.3 Online comparison of automation components

Using the comparison for automation components, the configuration objects in the active configuration are compared with the configuration objects on the controller. An overview shows the differences on an object level and the comparison of the configuration objects in detail. Then the parameter values are applied to the Automation Studio project.

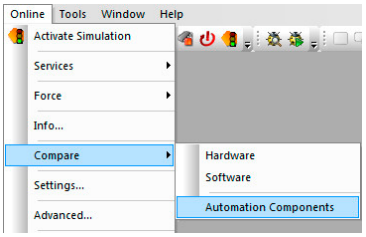
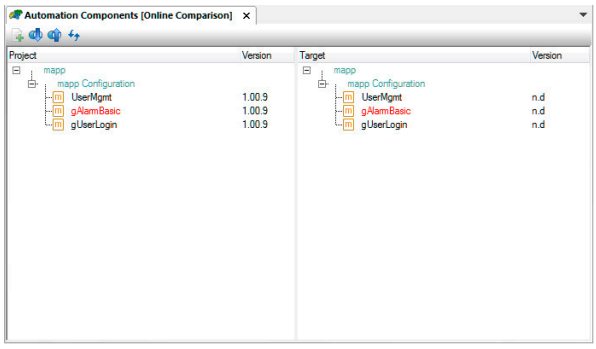


Figure 12: Opening the comparison for automation components

The online comparison for automation components can be activated by selecting **<Open> / <Compare> / <Automation Components>** from the main menu.



The window is divided into two halves. The left side shows the automation components in the project. The right side shows the configuration objects that are loaded at runtime. Differences are indicated using different text colors. The possibility to upload complete configuration objects is offered.

Figure 13: Online comparison of automation components

Applying parameter values

The detailed comparison shows the values of the configuration object in the Automation Studio project alongside the current values on the controller. The values for selected parameters are applied to the Automation Studio project by selecting a parameter or an entire parameter group (1) and then uploading via the toolbar (2).

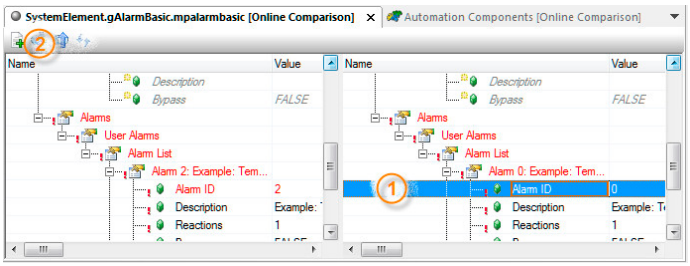


Figure 14: Applying parameter values in the detailed comparison

Diagnostics and service \ Diagnostic tools \ Monitors \ Online comparison of automation components

- Overview comparison
- Detailed comparison

3.3 Error analysis in Logger

Automation Runtime logs all fatal errors (e.g. cycle time violations), warnings and information messages (e.g. warm restarts) that take place when the application is executed.

This log is stored in the controller's memory and is available after restarting.

Diagnostics and service \ Diagnostic tools \ Logger window

- Opening the Logger window
- Operating the Logger \ Storing / Loading Logger data
- User interface description \ Backtrace

3.3.1 Logger with an active online connection

The Logger can be opened from the Physical View using the **<Open> / <Logger>** menu item, **<Open Logger>** in the controller's shortcut menu, or using the keyboard shortcut **<CTRL> + <L>**.

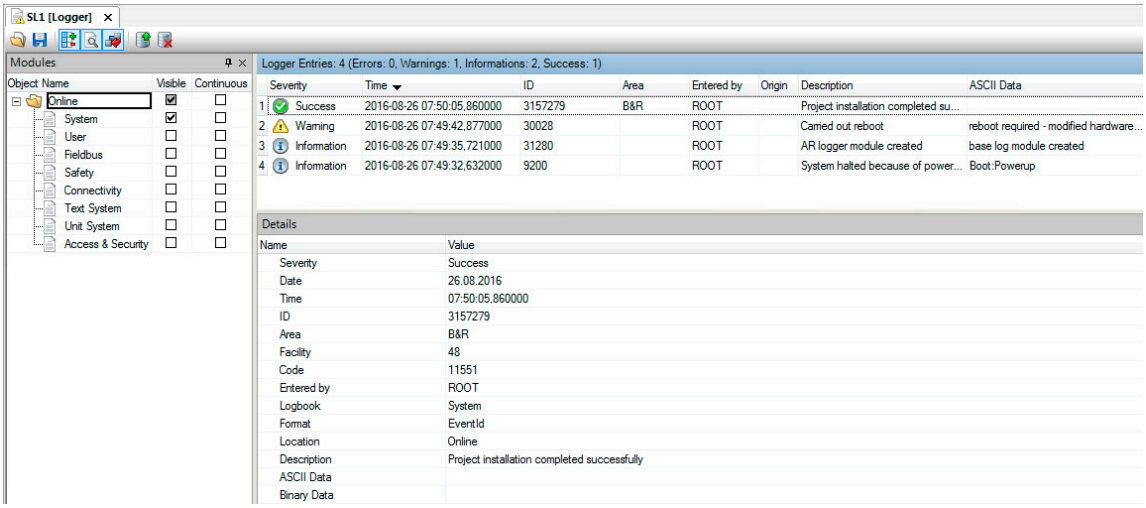


Figure 15: Log window

The entries displayed in this image show the events logged by Automation Runtime after creating offline installation and the subsequent startup of the controller.

Exercise: Cause a cycle time violation and check the entries in the Logger

Using the "**Loop**" program, a cycle time violation can be triggered by incrementing the "**udEndValue**" variable in the Watch window.

Once an online connection has been reestablished between Automation Studio and the target system after restarting, open the Logger window and look for the cause of the boot into service mode.

1) Open the Watch window in the "**Loop**" task's software configuration.

   See "Monitoring and modifying variables" on page 18.

2) Increment the "**udEndValue**" variable until a cycle time violation occurs (loss of connection and restart in SERVICE mode).
3) Open the Logger from the Physical View.
4) Look for the cause of booting in service mode.
5) Select the entry and press **F1**.

Once opened, the Logger indicates the cause of booting in SERVICE mode. Using backtrace, the program that is causing the problem can be quickly identified in this case.

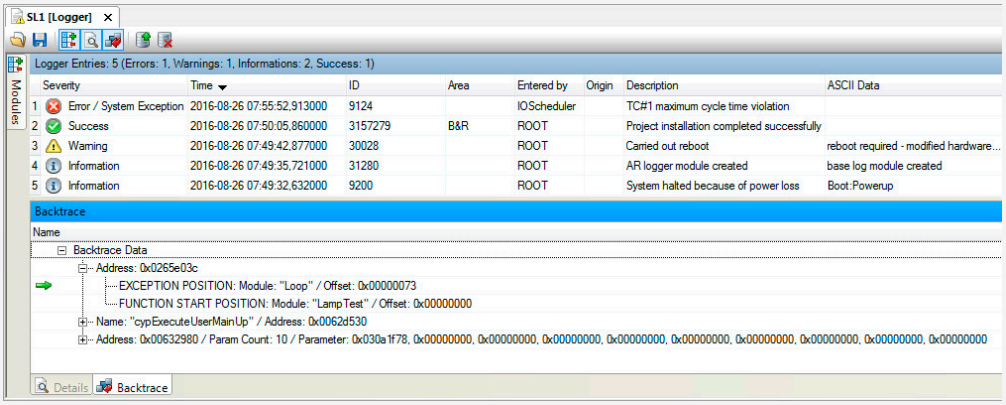


Figure 16: Cycle time violation in the Logger

Once an entry is selected in the Logger, pressing **<F1>** displays a detailed error description in the Automation Studio help documentation. Additional information about error entry is available by displaying the backtrace.

This opens a description of the selected error number in Automation Help.

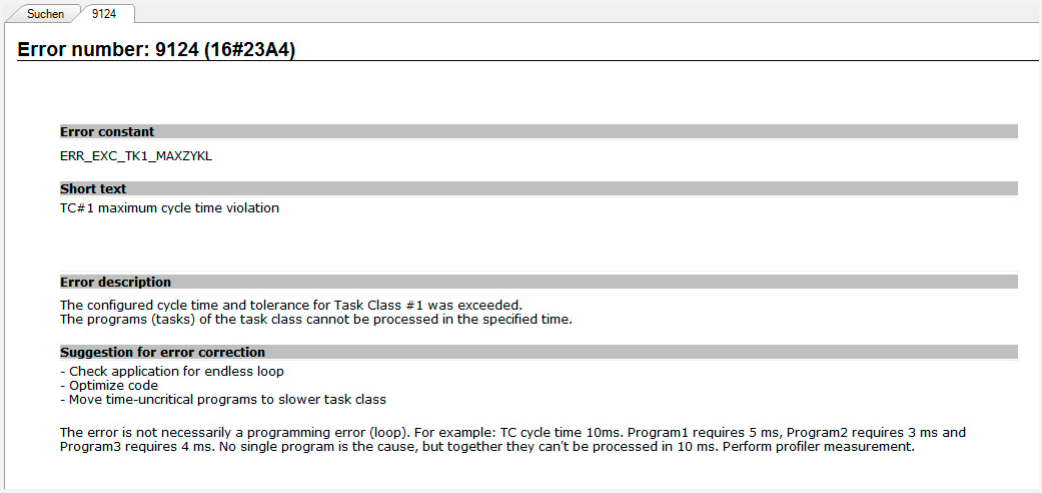


Figure 17: Context-sensitive help for Automation Runtime errors

3.3.2 Offline evaluation of Logger data

Logger records can also be evaluated without a connection to the controller.

Nonetheless, the data itself must always be uploaded by means of an existing online connection to Automation Studio or with the aid of System Diagnostics Manager.

- See "System Diagnostics Manager (SDM)" on page 43.

Use case

The Logger data is saved by the service engineer individually using System Diagnostics Manager or as a system dump. The transferred data can now be opened in Automation Studio and analyzed.

In Automation Studio, Logger entries can be saved and reloaded from the Logger's toolbar.

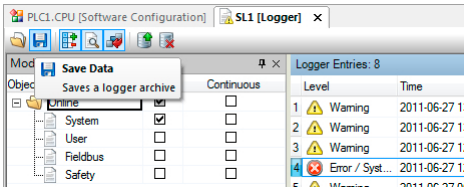


Figure 18: Saving Logger entries

3.3.3 Generating user log data

Logger functions can also be used by the application program to log certain events that occur within that application.

This is handled using the functions in the **ArEventLog** library. The functions contained in this library allow 32-bit event IDs to be entered. They are distinct within the system. Errors, warnings, information and successes are differentiated. Event IDs from the user and from the system can be easily differentiated.

The user cannot enter any event IDs used by the system in the Logger. This is prevented by the library.

The 32-bit event IDs are put together as follows:

Bits 31-30	Bit 29	Bit 28	Bits 27-16	Bits 15-0
Severity	1 .. Customer	Reserve	Facility	Code

User event IDs are generated according to the represented circuit diagram or by using the ArEventLog-MakeEventID() function.

This facility makes 12 bits available to clearly identify ranges. For user event IDs, the facility is divided up as follows:

- Values 0 .. 15: Customer applications
- Values 16 .. 4096: Range for device manufacturers and special cases

Programming \ Libraries \ Configuration, system \ ArEventLog

Applications:

- Logging service actions (e.g. battery replacement)
- Logging user actions (e.g. forbidden entries)
- Retrieving events of an exception task and entering them in the Logger
- Logging events with deactivated module monitoring (e.g. moduleOK = FALSE)

Exercise: Generate user Logger file

In the existing Automation Studio project, create a user Logger file called "UsrEvLog". The message **"This is a user warning"** is entered in this Logger file. Concerning message type, it should be a warning (severity = warning). The library example of the ArEventLog library is used for this purpose. After importing and transferring the sample program, the individual functions are enabled via the Watch window.

1) Import "LibArEventLog_ST" library example to the Logical View
2) Carry out project installation
3) Generate user Logger file using Watch window
4) Enter warning in the user Logger file using the Watch window

After generating the user Logger file and the Logger entry, the "userlogger" Logger file with the respective entry in the Logger becomes visible.

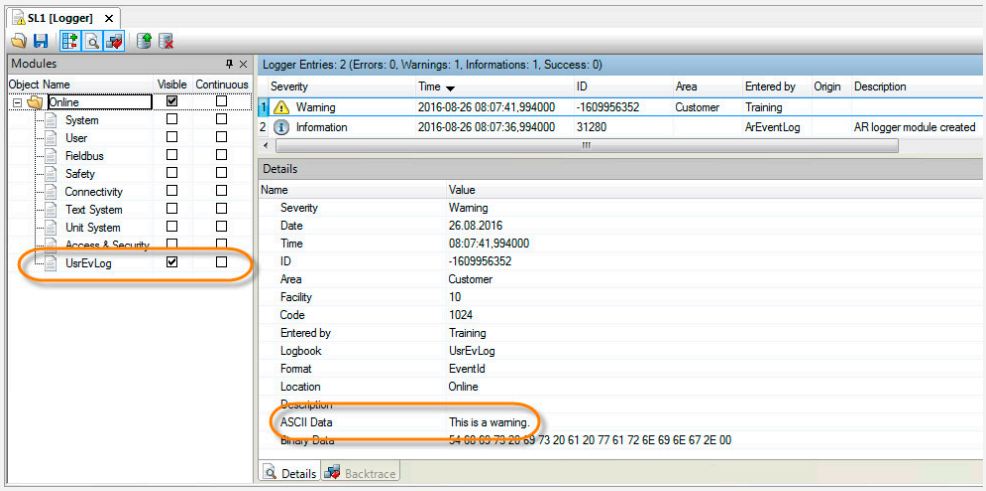


Figure 19: The Logger entry generated by the ArEventLog library

Basically, the sequence when creating a separate program with the ArEventLog library would look like the following:

- Generate user Logger file "userlogger" with ArEventLogCreate()
- Read ident of user logbook with ArEventLogGetIdent()
- Generate 32-bit event ID with the ArEventLogMakeEventID() function
- Write Logger entry in the user logbook with ArEventLogWrite()

Programming \ Libraries \ Configuration, system \ ArEventLog

Programming \ Examples \ Adding examples

Programming \ Examples \ Configuration, system information, runtime control \ Create and evaluate user logbook

4 Monitoring and analyzing process values

Process values can be monitored, analyzed and modified in many different ways in Automation Studio.

Monitoring and analyzing process values	
"Monitoring and modifying variables" on page 18	The Watch window allows variable values on the target system to be monitored and modified.
"Recording variables in real time" on page 22	The trace function makes it possible to record several values in real time over a defined period of time. This data can be uploaded using Automation Studio and displayed in the form of a curve.
"Monitor and force I/O" on page 26	The I/O monitor makes it possible to analyze the values of I/O variables and unused I/O channels as well as network quality.
"Searching for errors in the source code" on page 35	A wide variety of diagnostic functions are available for both text-based and visual programming languages.

Table 3: Monitoring and analyzing process values

Requirements for the exercises in this section

The descriptions and images in this chapter refer to the Automation Studio project "CoffeeMachine".

- Transfer the "CoffeeMachine" project to Automation Runtime simulation (ARsim)
- Connect to Automation Runtime Simulation (ArSim)

Figure 20: ArSim symbolic representation in the System Designer

4.1 Monitoring and modifying variables

The Watch window allows the values of variables on the target system to be displayed, monitored and modified.

Variable lists are saved in the Watch window for diagnostic and function tests with use and are reused at a later time.

Exercise: Operate and diagnose the "CoffeeMachine" application

Operate the "**CoffeeMachine**" application using the Watch window in Automation Studio.

Insert the variables from the following table into the Watch window. Test the process sequence of the "**CoffeeMachine**" application by manipulating and monitoring the values of the variables.

Overwriting of process variables during operation may only be carried out by authorized personnel.

Automation Software \ Sample programs \ B&R coffee machine

The following process variables are required for this task:

Function Range of values	Process variable	Description
Type of coffee 0 - 2	gMainLogic.par.coffeeType	Index for recipe selection
Recipe 0 - 100	gMainLogic.par.recipe.coffee gMainLogic.par.recipe.milk gMainLogic.par.recipe.sugar gMainLogic.par.recipe.water	Parameters of the recipe of a selected coffee type (0,1,2)
Coffee price 0.0 - 10.0	gMainLogic.par.recipe.price	Price for the selected recipe
Payment 0 - 10	gMainLogic.par.givenMoney	Inserted coins are compared with the price of coffee.
Switch on/off 0 - 1	gMainLogic.cmd.switchOnOff	Switch on the machine - wait for it to heat up
Preparation 1	diStartCoffee	Start preparing the selected drink
Water temperature	gHeating.status.actTemp	Current water temperature
Messages	gMainLogic.cmd.vis.messageIndex	Index of displayed messages
Process sequence	gMainLogic.status.progressStep	Representation of process progress Value = 1 corresponds to filling. Value = 2 corresponds to dispensing the cup.

Table 4: Required process variables in order to operate the "CoffeeMachine" project

4.1.1 Adding the process variables to the Watch window

- Transfer the "**CoffeeMachine**" project to Automation Runtime simulation (ARsim)
- Open the Watch window from the "**mainlogic**" task in the software configuration.
- Insert the variables from the table above using the toolbar or by pressing the **<Ins>** key.

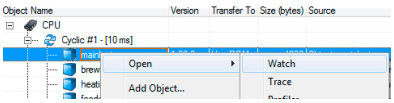


Figure 21: Open the Watch window

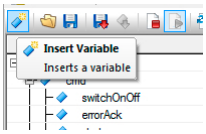


Figure 22: Inserting variables

Once the variables have been inserted in the Watch window, the process sequence of the application can be simulated.

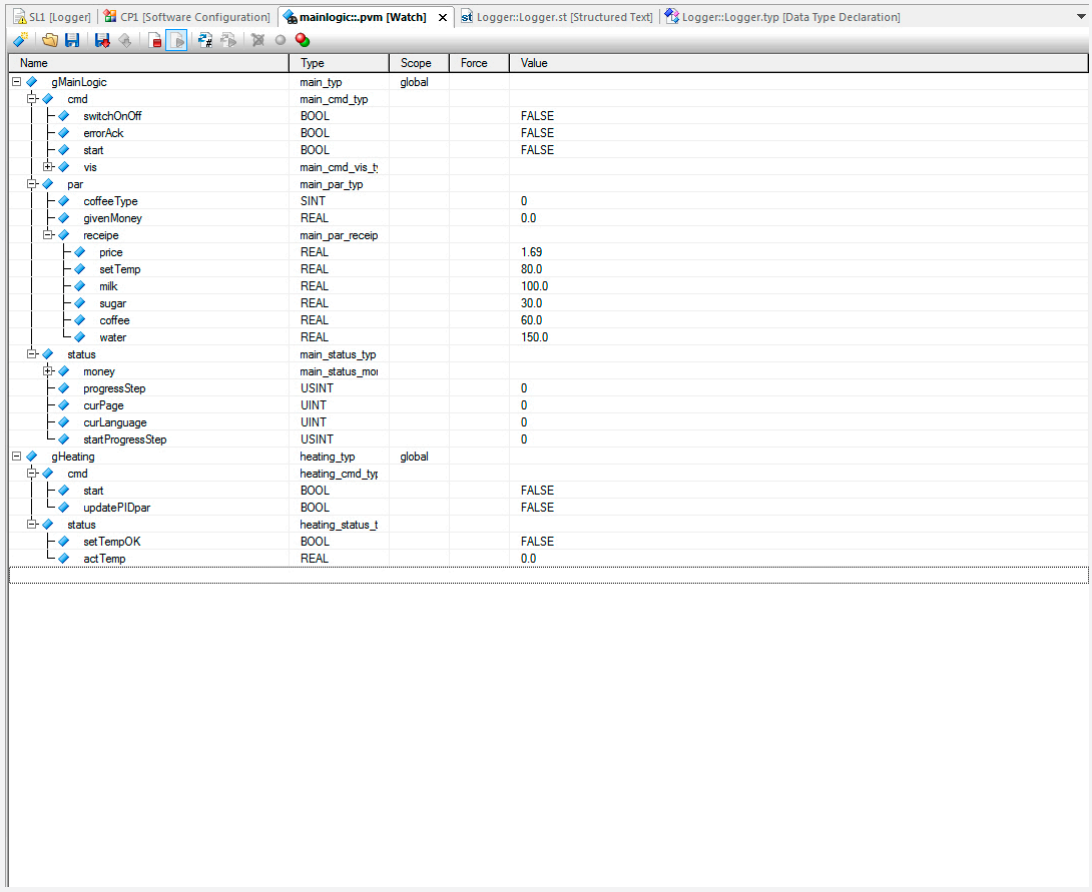


Figure 23: Displaying the variables in the Watch window

4.1.2 Turn on the coffee machine and operate it from the Watch window

1) Switching on
   "**gMainLogic.cmd.switchOnOff = 1**"
2) Check the status of the process
   Reaching the temperature setpoint is indicated by "**gMainLogic.cmd.vis.messageIndex = 2**".
3) Insert coins
   The coins inserted with "**gMainLogic.par.givenMoney**" must be greater than or equal to the coffee price "**gMainLogic.par.recipe.price**".

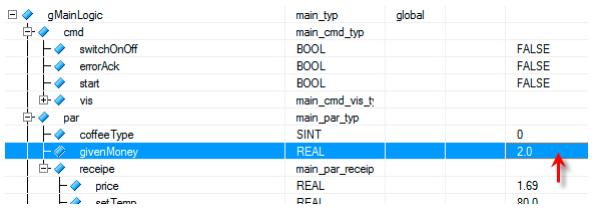


Figure 24: Simulate coin insertion

4) Start the preparation phase

"**diStartCoffee = 1**"

5) Monitor the progress of the process
Monitor the "**gMainLogic.status.progressStep**" variable.
Value = 1 corresponds to filling; Value = 2 corresponds to dispensing the cup.

The variable list in the Watch window should be saved for later use. This way the used variable list can be restored at any time.

Variables on the controller can be monitored and modified using the Watch window.

In addition to their values, additional information about the variables such as their scope, data type and I/O type, is displayed.

Variables can also be managed in separate lists to handle various other tasks.

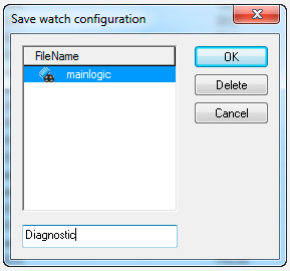


Figure 25: Saving the variable list

Diagnostics and service \ Diagnostics tool \ Variable watch

4.1.3 Writing to variable values simultaneously

If a value is changed in the Watch window, it will be transferred to the controller immediately after **<Enter>** is pressed.

The controller will then apply the new value in the next cycle.

In order to enter several values in the Watch window without **write request**, archive mode must be turned be on.

Archive mode can be started or ended using the "**Archive mode**" icon in the toolbar.

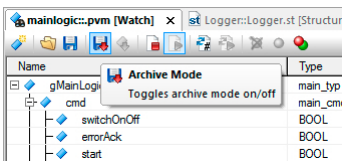


Figure 26: Enabling archive mode

After the values for the variables that need modification have been entered in the Watch window, all of them will be sent to the controller by clicking the "**Write values**" pushbutton in the toolbar.

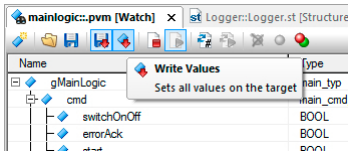


Figure 27: Changing all values in archive mode

Diagnostics and service \ Diagnostic tools \ Watch window \ Archive mode

4.2 Recording variables in real time

When using the Watch window, the variables on the controller are polled by Automation Studio.

However, this type of asynchronous accessing of the actual value changes in the Automation Runtime task class system leads to the following limitations:

- Value displayed asynchronously to the task class
- Unable to determine series of value changes and their dependencies

The "**Trace**" function can be used to record changes in values on the target system in real time and synchronous to the task class.

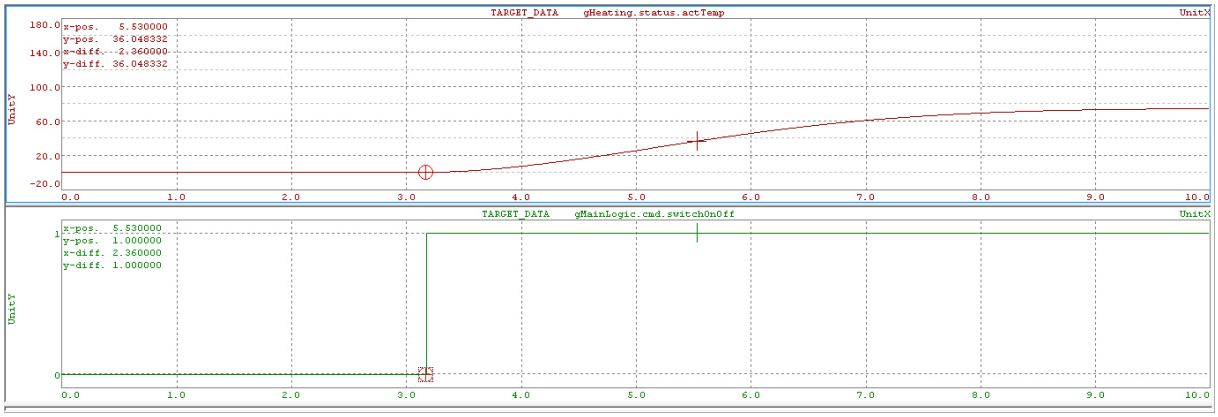


Figure 28: Example of a Trace recording

The following example shows how another process is started when the state of a particular variable is changed. The measurement cursor can be used to establish the time difference between the corresponding value changes of both curves.

By analyzing recordings, processes in the application can be optimized and errors detected.

The Trace dialog box is started for the corresponding task in the software configuration using the **<Open> / <Trace>** shortcut menu.

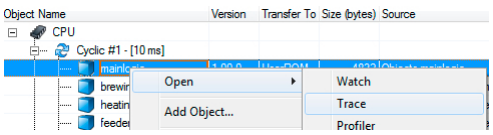


Figure 29: Opening the Trace window in the software configuration

A new Trace configuration must be inserted in the Trace dialog box by clicking the "**Insert Trace Configuration**" pushbutton.

The variables to be recorded are added to this Trace configuration by clicking the **"Insert New Variable"** pushbutton.

Exercise: Record a curve that depends on other variables

In the "**CoffeeMachine**" process sequence, the water temperature goes through a warming up phase after starting. Changing the coffee type also changes the target temperature; the water temperature is then continuously checked until the target temperature is reached.

This task shows how temperature regulation – in this case with a distinct overshoot – can be easily analyzed by recording the temperature profile in real time. The following process variables are required to do this:

Activity	Range of values	Process variable
Select coffee type	0-2	gMainLogic.par.coffeeType
Switching on/off	0-1	gMainLogic.cmd.switchOnOff
Water temperature	-	gHeating.status.actTemp

Table 5: Required process variables for the Trace recording

4.2.1 Opening the Trace window and adding variables

- Open the Trace window for the "**mainlogic**" task.
- Insert a new Trace configuration.
- Insert the process variables needed for the recording.
  The Trace configuration looks like this:

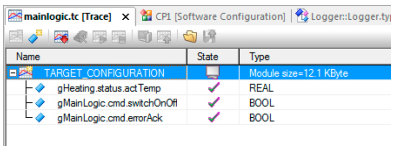


Figure 30: Trace configuration

Values are recorded cyclically in the context of the task class. The period and start condition of the recording can be configured in the Trace configuration's properties.

In this example, the recording starts when the coffee machine is switched on (**gMainLogic.cmd.switchOnOff = 1)**.

4.2.2 Editing Trace configuration

- Configure the recording buffer and trigger condition
- Open the properties dialog box for the Trace configuration.

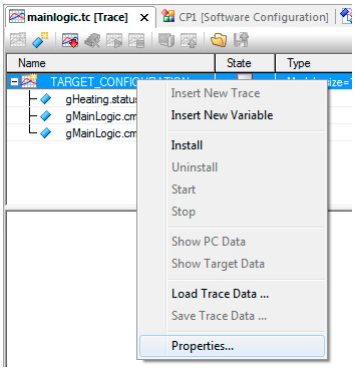


Figure 31: Trace properties

- Configuring the recording buffer
  The size of the recording buffer can be set to 30,000 entries on the "**General**" property page.
- Changing the Trace mode
  A trigger condition for starting the recording can be configured on the "**Mode**" property page (**gMainLogic.cmd.switchOnOff = 1**).

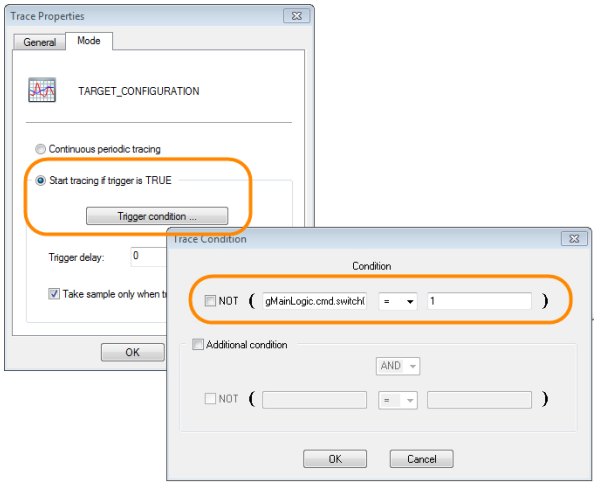


Figure 32: Configuring the trigger condition

The dialog box for selecting the variables for the trigger condition is opened by pressing the **<Space bar>**.

Once the recording itself has been configured, it will be transferred to the target system by clicking the "**Install**" pushbutton.

In this case, recording does **not** take place manually with the "**Start**" icon in the toolbar, but rather when the start condition has been met.

4.2.3 Starting the process and analyzing Trace data

- Opening the Watch window
  Open the Watch window and insert the necessary variables according to the table above.

  If the Watch configuration for the task was saved in "Monitoring and modifying variables" on page 18, it will reopen automatically.

- Turn on the coffee machine
  Set the "**gMainLogic.cmd.switchOnOff**" variable.
- Changing the type of coffee
  Change the "**gMainLogic.par.coffeeType**" to 0, 1 or 2. Changing the type of coffee will also change the temperature setpoint. Changes to the actual temperature are recorded in the Trace window.

Recording can be paused at any time by clicking the "**Stop**" icon in the toolbar. The results are displayed by clicking the "**Show target data**" icon after the upload has taken place.

Data is recorded when the start condition has been met. Values can be modified as needed in the Watch window. After the data has been uploaded from the target system, the recording will look something like this (depending on how the values of the variables have been changed):

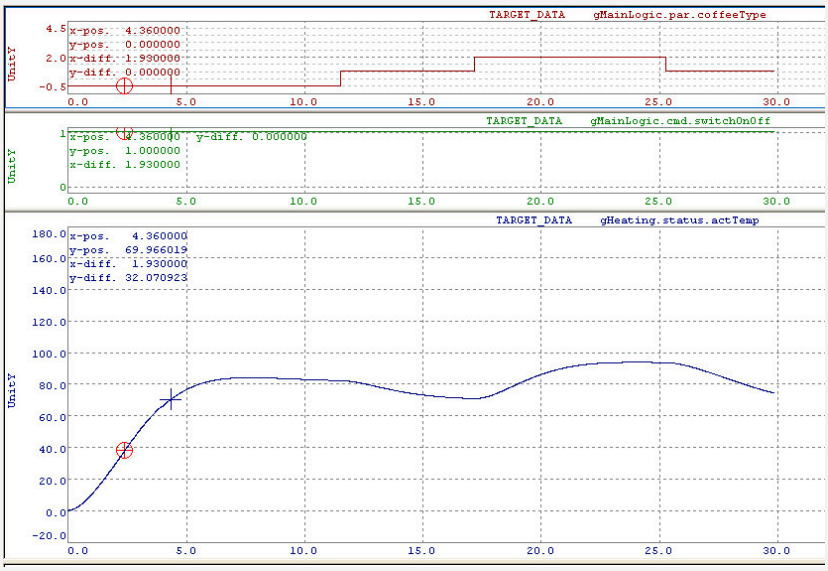


Figure 33: Trace data

Using the measurement cursor, value changes and differences are determined exactly.

Diagnostics and service \ Diagnostic tools \ Trace window

4.3 Monitor and force I/O

Double-clicking on a module in the Physical View opens the I/O mapping window. The physical status of the I/O channels is displayed when there is an active online connection and the appropriate monitor mode is selected.

Figure 34: Turn on monitor mode

The "**Force**" option makes it possible to assign any of the I/O data points – regardless of their actual physical value – a value for that channel, e.g. in order to test the program sequence. The "Force" function is enabled in the I/O allocation and in the Watch window that is linked to the I/O data points.

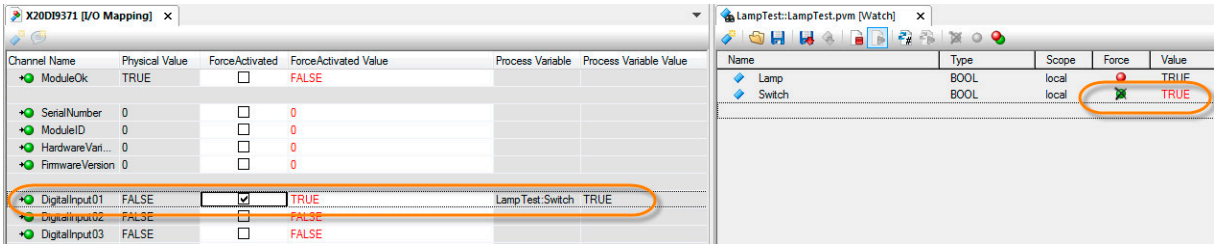


Figure 35: I/O mapping in monitor mode; forcing I/O channels in the I/O mapping and in the Watch window

Forcing I/O data points for input modules

The "force" value of a channel on an input card (e.g. X20DI9371) is "simulated" by Automation Runtime. The application program's process sequence then works with the "force" value and not with the actual input state.

Forcing I/O data points for output modules

The "Force" value of a channel on an output card (e.g. X20DO9322) is written directly to the output of the corresponding hardware, regardless of what value the application program has written to it.

When commissioning of the system is completed, it must be ensured that there are no force operations still in effect. This can be done automatically by **restarting** the system or using the **<Online> / <Force> / <Global force off>** menu item.

Diagnostics and service \ Diagnostic tools \ Monitors \ Mapping I/O channels in monitor mode

Diagnostics and service \ Diagnostic tools \ Force

Diagnostics and service \ I/O and network diagnostics

5 Software analysis during programming

There are several different diagnostic tools available in Automation Studio that provide support when designing the application software.

Not only that, there are ways to detect application/software errors in both Automation Runtime as well as the actual source code.

Software analysis during programming	
"Perform runtime measurements in the Profiler" on page 27	The Profiler can be used to measure and display important system data such as task runtimes, system and stack loads, etc.
"Line coverage" on page 37	Line coverage indicates the lines of the source code that are currently being executed.
"Debugging the source code" on page 37	The debugger makes it easier to search for errors in the source code of a program or library.
"Evaluating event IDs, status variables and return values" on page 40	Status variables are used to evaluate the status of or error in a function call within the application program.
"Using variables in the programs" on page 41	The output window is used to display information about ongoing processes, e.g. building, downloading, generating the cross-reference list, displaying search results, etc.

5.1 Perform runtime measurements in the Profiler

Automation Runtime continually records all runtime behavior. Using the Profiler, the runtimes of individual user tasks, the CPU load and different system events can be recorded. Open the Profiler from the software configuration by selecting **<Open> / <Profiler>** from the shortcut menu.

In the Profiler, different views for displaying the recorded data are offered. These are switched in the toolbar of the Profiler. A table view, a graphic view and a raw data view are available. The most important functions of the Profiler, such as opening the configuration, and starting and stopping the recording, are also controlled via the toolbar.

Figure 36: Toolbar in the Profiler window

For the following tasks, it's recommended to leave open the **Watch window** for the tasks "Loop" and "Loop1", the **software configuration** and the **Profiler**.

Diagnostics and service \ Diagnostics tools \ Profiler

- Recording Profiler data
- FAQs

5.1.1 Configuring the Profiler and carrying out recording

In order to get diagnostically conclusive measurement results, the recording duration and the recorded event types must be configured. For the following tasks, only the execution times of the user tasks, task classes and exceptions are recorded. Short instructions follow, explaining how to edit the configuration and record data.

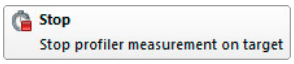
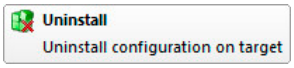
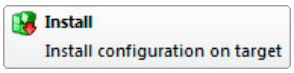
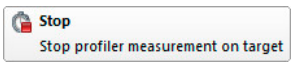
1	In order to edit the configuration, you must stop the current recording.	
2	The existing Profiler configuration is uninstalled from the target system.	
3	To open the dialog box for making configurations, click the "Configuration" icon in the Profiler toolbar.	
4	Transfer your changes to the target system using the "Install" icon in the Profiler toolbar. Recording begins again immediately with the new configuration.	
5	The current recording is stopped with the "stop" symbol.	
6	The Profiler data is loaded to and displayed in Automation Studio via the "upload data object" symbol.	

Table 6: Overview of the required steps for changing the Profiler configuration and uploading the Profiler data

Make the configurations shown in the following images:

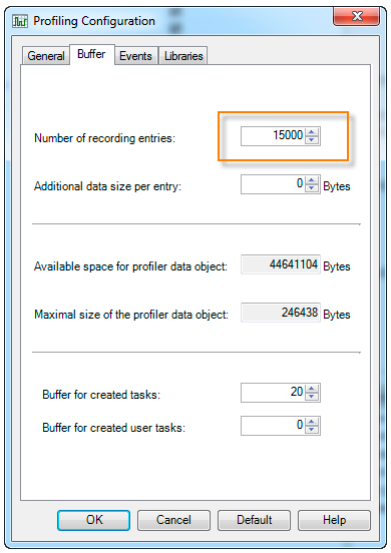


Figure 37: Profiler configuration: Recording buffer

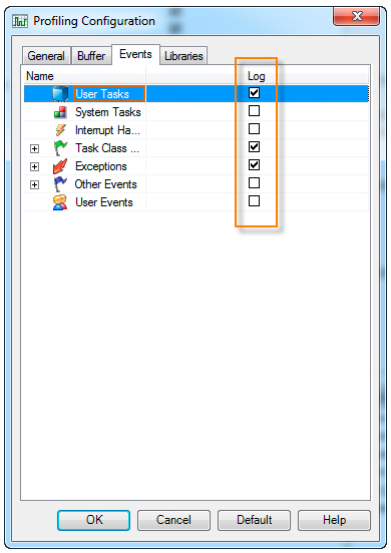


Figure 38: Profiler configuration: Event types

If the Profiler data is relayed to third parties, it is recommended to log all events in the "**Events**" tab. Filtering can be performed later.

Diagnostics and service \ Diagnostic tools \ Profiler \ Preparing the Profiler

Exercise: Measure the execution time of the "Loop" program

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

Use the **Profiler** to monitor the runtime behavior of the task and the available idle time.

1) Setting the "udEndValue" variable to 50,000

   The first step is to set the value of the "udEndValue" variable to 50,000.

2) Increasing the value of the "udEndValue" variable in steps

Results should be analyzed in the Profiler between each of these steps.

5.1.2 Profiler table and graphic views

Different views are available for analyzing the Profiler data. The data uploaded in the preceding task is now displayed in table or graphic form.

Table display of Profiler data

A table view of the Profiler data shows – with the appropriate filtering – the execution time and CPU load of each task.
This view is opened with the "**Table**" icon in the toolbar.

Name	CPU Usage [%]	Tolerance Count	Object Priority	Call Count	Minimal Net Time [µs]	Average Net Time [µs]	Maximal Net Time [µs]	Minimal Gross Time [µs]
Cyclic #1	2.145		230		217.356	218.539	219.832	965.138
loop	1.943		230	15	201.461	202.663	203.198	201.461
Cyclic #4	0.142		198		216.306	216.306	216.306	216.306
System Tasks	6.059							
Interrupt Ha...	1.061							
Idle Tasks	90.593							

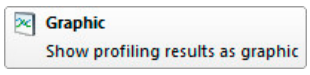
Figure 39: Analyzing the CPU load with the Profiler

In order to get informative Profiler graphs, the call count of the slowest task class in the system must be >3. The recording duration of the Profiler is set via the buffer size in the Profiler configuration.

Diagnostics and service \ Diagnostic tools \ Profiler \ Recording Profiler data \ Analyzing Profiler data \ Analyzing Profiler data in table form

Visual display of Profiler data

Switch to the graphic display of the Profiler data via the "visual" symbol in the toolbar. The representation can be shown as in the following image. Comprehensive analysis options are available via filter, zoom and measurement cursor functions.



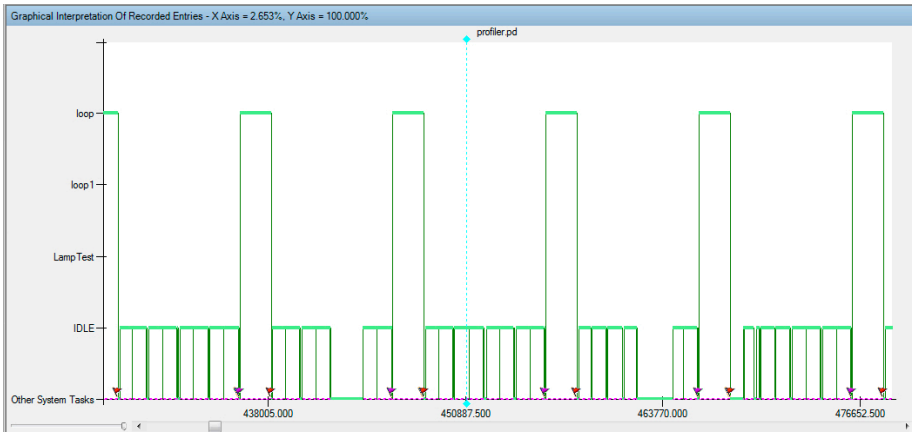


Figure 40: Result of the first Profiler measurement

Diagnostics and service \ Diagnostic tools \ Profiler \ Recording Profiler data \ Analyzing Profiler data \ Show Profiler data as graph

- Navigating in the graph view
- Zooming in the graph view

5.1.3 Measuring program runtime in the Profiler

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.

Exercise: Increase the "udEndValue" variable in steps

The Profiler is restarted in this step. Increase the value of the variable "**udEndValue**" in steps (e.g. 10000). Use the Profiler to monitor the execution time of the "**Loop**" task. In addition, when a value is changed, the recording is stopped and uploaded.

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.

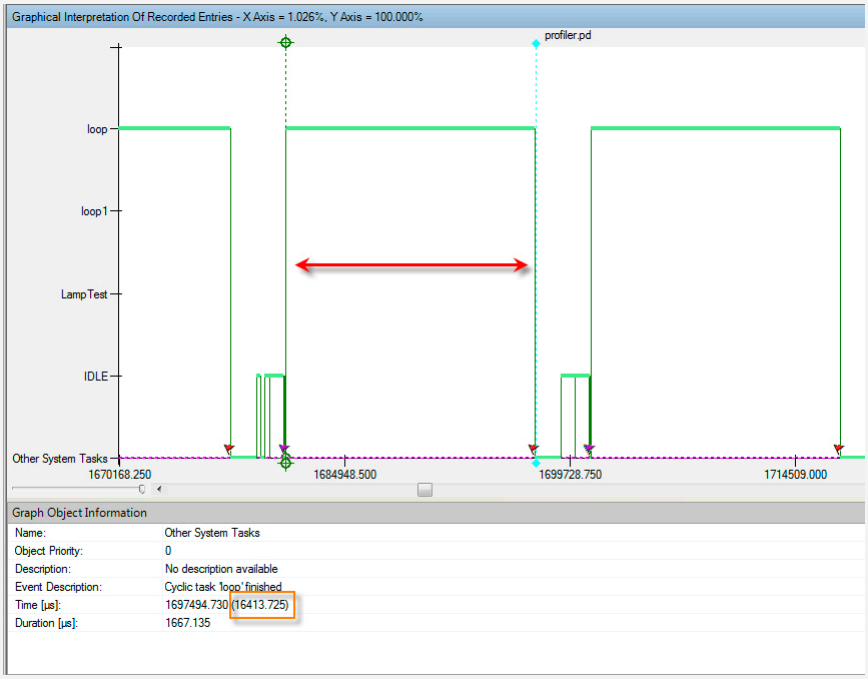


Figure 41: Exceeding the cycle time, effect of tolerance

With the value "tolerance count", it can be determined in the table view if the tolerance time is already active in a task class. In the following image, the middle runtime of task class #1 is 10.7 ms. The tolerance count shows that the configured execution time was already exceeded 271 times.

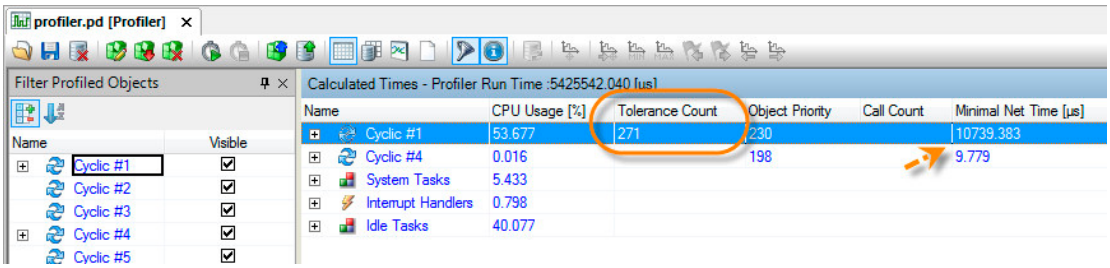


Figure 42: Parameter tolerance count in the Profiler

The example shows the effect of using the tolerance on the average CPU load. Task class #1 causes a CPU load of around 54% even though the configured cycle time of 10 ms was exceeded. In this case, when the tolerance becomes effective, the task class #1 is only called again every 20 ms.

Exercise: Record task classes with different priorities

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.

Based on the tasks "**Loop**" and "**Loop1**" use the Watch window 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 **"udiEndValue"** variable in the **"Loop"** task to 2,000. A Profiler measurement could look like the following:

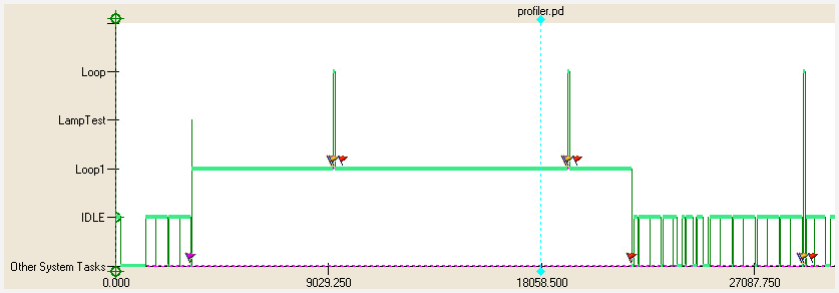


Figure 43: "Loop" task interrupts "Loop1" task - Display in the graphic view of the Profiler

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

5.1.4 Read Profiler data after an error

The Profiler data can be uploaded by the target system at any time after stopping the recording. In case of an error, the Profiler data is also retained. It can then be determined which event, for example, led to the system being restarted.

The Profiler configuration should be adjusted so that the Profiler data is kept after restarting. The target memory for Profiler data and Profiler configuration is set to "USERROM".

Diagnostics and service \ Diagnostic tools \ Profiler \ Preparing the Profiler \ General settings

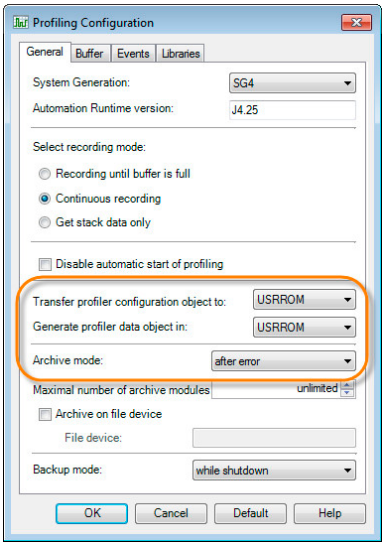


Figure 44: Settings for the target memory of Profiler data and the Profiler configuration

Exercise: Cause a cycle time violation and evaluate the Profiler data

A cycle time violation can occur by increasing the value of the "**udEndValue**" variable in the "**Loop**" task.

After restarting the target system in service mode, open the Profiler and load the Profiler data from the target system.

1) Cause a cycle time violation by setting the "**udEndValue**" variable to the value 500000 in the Watch window
2) After restarting into service mode, open the Profiler in the software configuration by selecting the <**Open**> / <**Profiler**> menu item.

If the configured **cycle time + tolerance** was exceeded during runtime, Automation Runtime triggers an exception. If the application program is not configured to handle this exception, the target system will restart in SERVICE mode.

In the Profiler, data is uploaded by clicking on the **"Upload data object"** icon in the toolbar. If there is an error, a new Profiler file is generated upon restart, which is given a timestamp. The corresponding file can be selected from a list during the uploading process.

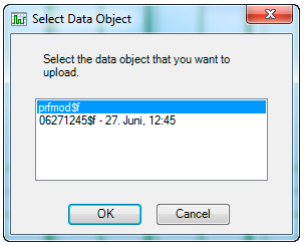


Figure 45: Selecting the Profiler data

Profiler data can be filtered to limit the events being displayed. Which events should be displayed depends on the situation itself.

Diagnostics and service \ Diagnostic tools \ Profiler \ Recording Profiler data \ Analyzing Profiler data

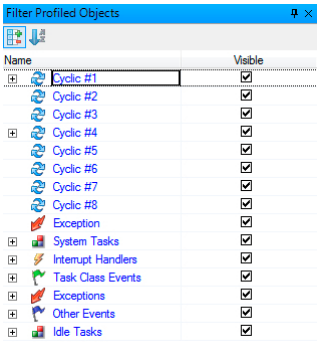


Figure 46: Filtering Profiler data

At a certain point in time (e.g. when too many loop cycles have occurred), the time it takes to complete the task exceeds the configured cycle time and tolerance. This event (exception) is indicated by an appropriate icon.

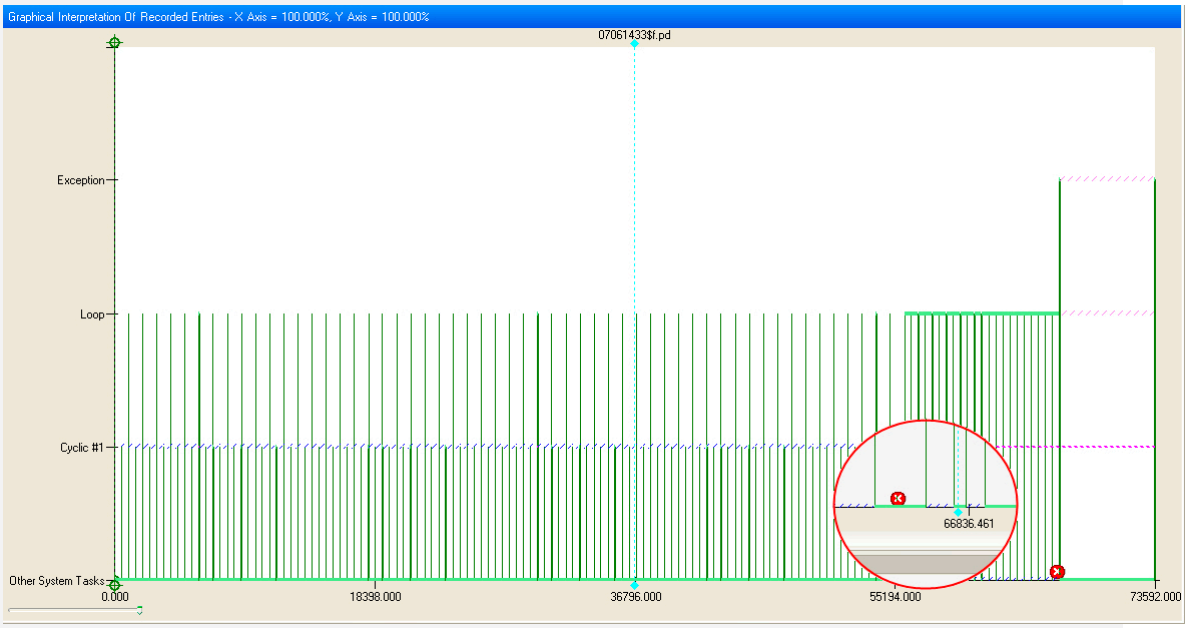


Figure 47: Exception in the Profiler data

To analyze the cause, the data that comes before this point in time must be observed.

Using the measurement cursor and zooming in as necessary on the Profiler data are two ways that the data can be analyzed.

"Loop" task execution time

The image shows that the task "Loop" is normally carried out in less µs (blue arrow). In the case of cycle time violation, the configured cycle time + tolerance is exceeded (red arrow).

This image shows how a simple application is recorded in the Profiler. The cause of a problem is generally harder to detect in real applications since there are usually several tasks / task classes running.

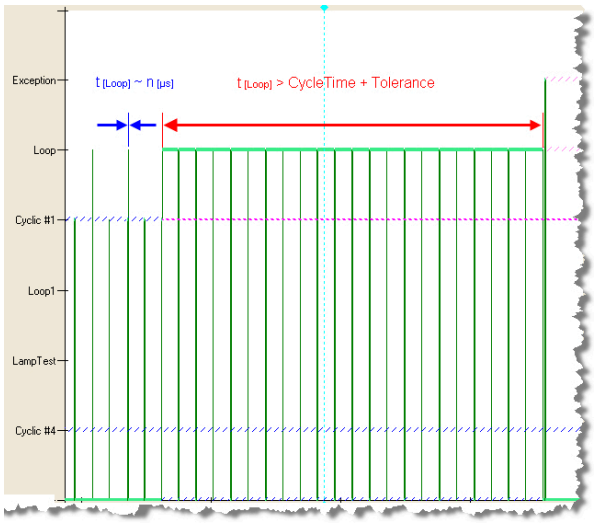


Figure 48: Determining the cycle time violation

Example:

Two tasks are running in task class #1 that usually finish executing within the configured task class cycle.

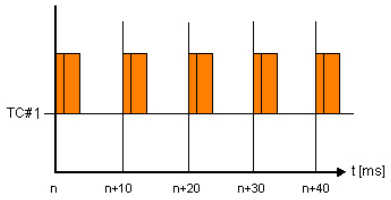
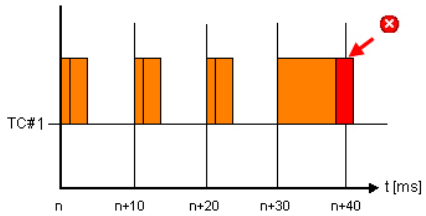


Figure 49: Timing diagram - Execution times in task class #1



If it takes longer to complete the first task (beyond n+30 ms in the diagram) and the completion time for both tasks together exceeds the configured cycle time plus tolerance, then it will be the second task that is entered as the cause of the error although it is not really the reason for the cycle time violation.

Figure 50: Timing diagram - Execution times in task class #1 exceed the configured cycle time

The sequence of events can be analyzed chronologically by evaluating the raw data ("**Output data**" icon in the toolbar).

This list shows the start and finish entries for the "Loop" task. If the chronological sequence were to be traced further, you would see that although the task itself has been started, it has not ended.

Raw Data Of Recorded Entries

Nr.	Name	Event	Event Description
465	TickGen	0x000...	'TickGen' is now running - 'IOScheduler' was waiting
466	Cyclic #1	0x001...	'Cyclic #1' is now running - 'TickGen' was waiting
467	Cyclic #1	0x1e0...	Task class 'Cyclic #1' started
468	Cyclic #1	0x1e0...	Input scheduler of task class 'Cyclic #1' finished
469	Loop	0x020...	Cyclic task 'Loop' started
470	Loop	0x020...	Cyclic task 'Loop' finished
471	Cyclic #1	0x1e0...	End of cyclic programs in task class 'Cyclic #1'
472	Cyclic #1	0x1e0...	Output scheduler of task class 'Cyclic #1' started
473	DdXZXAcc.IF6	0x007f...	'DdXZXAcc.IF6' is now running - 'Cyclic #1' was waiting
474	tEpIV2If.IF3	0x007f...	'tEpIV2If.IF3' is now running - 'DdXZXAcc.IF6' was ready
475	DdXZXAcc.IF6	0x007f...	'DdXZXAcc.IF6' is now running - 'tEpIV2If.IF3' was delayed and waiting

Figure 51: Raw data for a Profiler recording

5.2 Searching for errors in the source code

When it comes to software, statistics have shown that there are usually around two to three errors contained in every 1,000 lines of code.

Automation Studio provides extensive diagnostic tools for locating the source of program errors.

5.2.1 Monitor mode in the program editor

Monitor mode is started by selecting the "**Monitor**" icon in the programming editor toolbar.

Figure 52: Enable monitor mode

Monitor mode is available for each programming language and allows variables to be observed in several ways:

Tooltips in text-based programming languages

The value is shown as a tooltip of a variable in both text-based and visual programming languages.

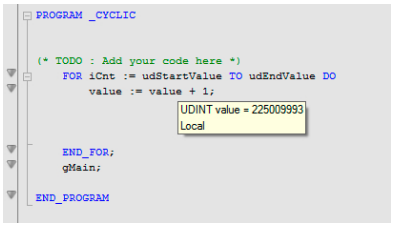


Figure 53: Tooltip in the source code

Value display in visual programming languages

The value is shown right beside the variable name in visual programming languages.

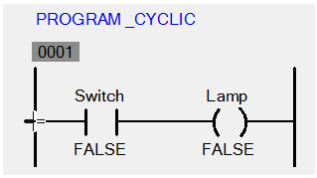


Figure 54: Visual programming language in monitor mode

Watch window

The Watch window is shown next to the source code when monitor mode is enabled. It is also possible to open the Watch window from a task's shortcut menu in the software configuration or from the online software comparison.

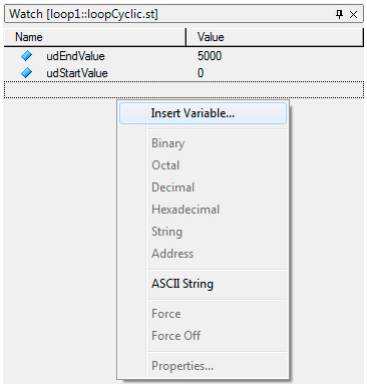


Figure 55: Watch window view

5.2.2 Powerflow

The path of a signal can be displayed in visual programming languages such as Ladder Diagram.

This signal path is enabled by selecting the "**Powerflow**" icon in the toolbar.

Exercise: Powerflow in Ladder Diagram

Enable Powerflow in the "**LampTest**" program (from TM210 – Working with Automation Studio).

The path of the signal can be observed in the Watch window by changing the value of the "**Switch**" variable.

1) Open the "**LampTest**" program when there is an online connection
2) Enable monitor mode
3) Add the "**Switch**" and "**Lamp**" variables to the Watch window
4) Set the "**Switch**" variable

Once the contact condition for the "**Switch**" variable has been met, the signal is then shown for the "**Lamp**" variable.

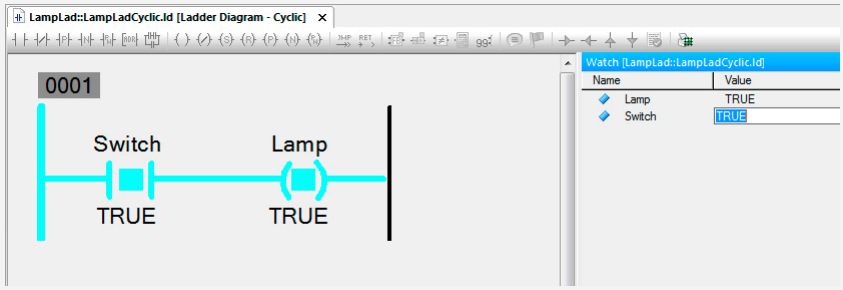


Figure 56: Power Flow enabled in Ladder Diagram

Diagnostics and Service \ Diagnostic tools \ Monitors \ Programming languages in monitor mode \ Powerflow

5.2.3 Line coverage

If line coverage is enabled for text-based programming languages, the marker indicates the lines of code that are currently being executed.

This makes it possible to see exactly which lines are being run at which time. Line coverage is enabled via the "Line Coverage" icon in the toolbar.

Diagnostics and service \ Diagnostics tool \ Monitors \ Programming languages in monitor mode \ Line coverage

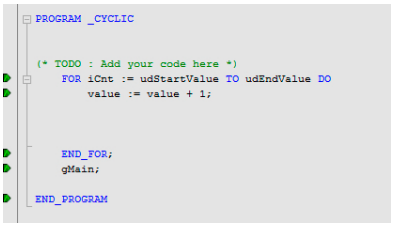


Figure 57: Line coverage in Structured Text

5.2.4 Debugging the source code

The debugger is an important tool for programmers that makes it easier to find errors in the source code of a program or in a library.

Debugging possibilities in Automation Studio

- Line-by-line execution of a program and the monitoring parallels of variables in the auto-watch window
- Stopping the application using breakpoints
- Stepping into called functions (e.g. in library functions / function blocks as long as the source code is available).

Exercise: Find errors in a Structured Text program using the debugger

Create a Structured Text program called "**dbgTest**".

Add a USINT array called "**AlarmBuffer**" with a length of 10 and a UINT variable named "**index**" to the "**dbgTest.var**" file.

In the cyclic part of the program, the array initializes with any value (e.g. 112).

The following – faulty – program code contains one of the most commonly made errors.

Program code

```
PROGRAM _CYCLIC
    FOR index :=0 TO 10 DO
        AlarmBuffer[index] := 112;
    END_FOR
END_PROGRAM
```

Table 7: Faulty program subroutine

When the program is started, the value 10 is written to each of the ten elements of the array; the program seems to be working.

Error description: There is write access to index 0 to 10. The array only has 10 elements (index 0 to 9). This type of error is often difficult to detect at first glance and causes the program to overwrite the following memory locations.

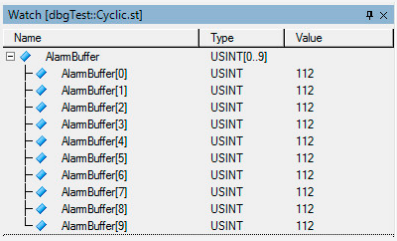


Figure 58: Watch window in monitor mode

The information in the debugger as well as the variables (and their values) in the Watch window will be used to try to analyze the error situation.

Monitor mode and debugger in the toolbar

Monitor mode can only be enabled if a program editor is open.

Figure 59: Enabling monitor mode

The debugger can be turned on and off from the menu bar.

Figure 60: Turning the debugger on/off

Adding variables to the Watch window

After activating monitor mode, the "AlarmBuffer" variable is added to the Watch window.

The left side of the window shows the program code, whereas the Watch window is shown on the right side.

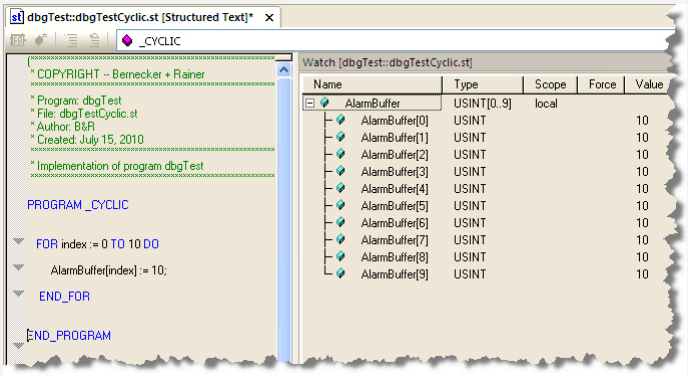


Figure 61: Monitor mode in the program editor

Setting breakpoints

- Move the cursor to the first line in the FOR loop.
- Set a breakpoint using the <**Debug**> / <**Toggle Breakpoint**> menu item or by pressing the <**F9**> key.

Enable the debugger

The debugger and any set breakpoints must be enabled from the menu.

If the debugger hits a breakpoint, then the active line is indicated by a yellow marker.

Reaching a breakpoint stops the entire application running on the target system.

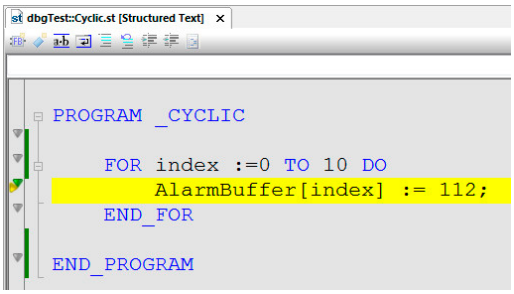


Figure 62: Active line in the debugger

"Step into" debugging

First, the elements of the "**AlarmBuffer**" array are manually changed to the value 0 in the Watch window.

The "**Step Into**" command or <**F11**> can be used to execute the program code one line at a time.The active line is always indicated by the yellow marker.

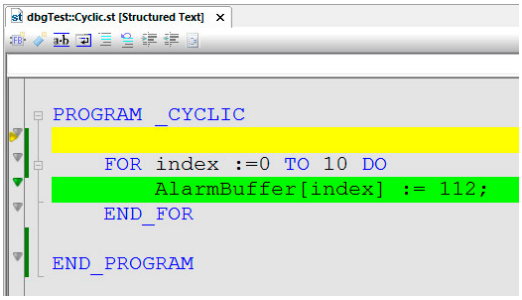
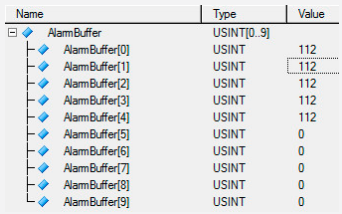


Figure 63: Active step marked yellow

If the <**F11**> key is pressed several times, each iteration of the loop causes the value of an element of the array to be changed. Continue through each step with the <**F11**> key until the new value has been assigned to the last element of the array.



Name	Type	Value
AlarmBuffer	USINT[0..9]	
AlarmBuffer[0]	USINT	112
AlarmBuffer[1]	USINT	112
AlarmBuffer[2]	USINT	112
AlarmBuffer[3]	USINT	112
AlarmBuffer[4]	USINT	112
AlarmBuffer[5]	USINT	0
AlarmBuffer[6]	USINT	0
AlarmBuffer[7]	USINT	0
AlarmBuffer[8]	USINT	0
AlarmBuffer[9]	USINT	0

Figure 64: Step-by-step writing to the variables

In this case, the "index" variable receives the value 9, which also corresponds to the upper limit of the array ([0..9]).

If continuing step by step with <**F11**>, the loop will iterate once more with a index outside of the array.

This type of error can be detected by the IEC Check library.

- See "IEC Check library" on page 40.

Diagnostics and service \ Diagnostics tool \ Debugger

5.2.5 IEC Check library

The IEC Check library contains functions for checking division operations, range violations, proper array access as well as reading from or writing to memory locations.

The corresponding checking function is called by the program (supported IEC 61131-3 languages or Automation Basic) before each of these operations is carried out.

With the IEC Check library, the user can can use a dynamic variable (REFERENCE TO) to determine what should happen when divided by zero, out of range errors or illegal memory access occurs.

Programming \ Libraries \ IEC Check library

5.2.6 Evaluating event IDs, status variables and return values

Return values and status values of functions and function blocks must be evaluated in the user program. The terms "status" and "event IDs" should be understood as synonyms in this context.

The following example shows a function block being called. This function returns a status that can be used to determine whether an error has occurred during the call. A list of the status values or event IDs are documented in the description of the function block in the Automation Help.

```
2:  (*Read information from an AR logger user module*)
    Logger.AsARLogGetInfo_0.enable := 1;
    Logger.AsARLogGetInfo_0.pName :=  ADR('usrlog');
    Logger.AsARLogGetInfo_0;  (*Call the Functionblock*)

    (*AsArLogGetInfo successful*)
    IF Logger.AsARLogGetInfo_0.status = 0 THEN
        Logger.Step := 0;
    ELSIF Logger.AsARLogGetInfo_0.status = ERR_FUB_BUSY THEN
        (*Busy*)
    ELSE  (*Go to Error Step*)
        Logger.Step := 10;
    END_IF
```

Figure 65: Status evaluation of function blocks

5.3 Using variables in the programs

The proper usage of variables in the different programs that are in Logical View can be checked by creating a cross-reference list or explicitly searching for a known variable name.

5.3.1 Generating cross-reference list

The cross-reference list indicates which process variables, functions and function blocks can be used at which point in the project.

The cross-reference list is optional and can be generated when the project is compiled (built); the results are then displayed in the output window under the "**Cross Reference**" tab.

To generate a cross-reference list, you must activate it using the following menu option: **<Project> / <Settings>**. Alternatively, the cross-reference list is generated via the menu item **<Project> / <Build cross-reference>**.

Frequent search tasks can be handled easily with the help of the cross reference list.

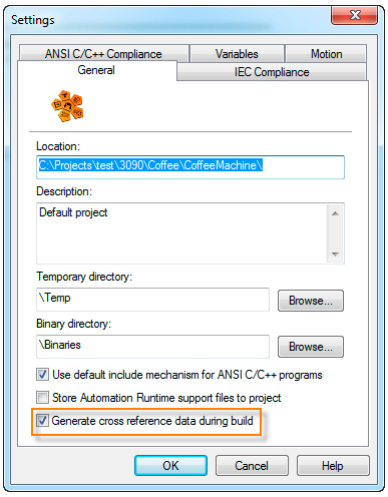


Figure 66: Enabling the cross-reference list during a project build

Exercise: Generate a cross reference list

Create a cross-reference list for the open project.

1) Enable the cross-reference list in the project settings.
2) Building the project
3) Check the cross-reference list in the output window.

You can analyze the cross-references for variables and their attributes in the output window.

If a variable is selected on the left side, its usage and the type of access will be displayed in the source code or in the I/O allocation on the right side.

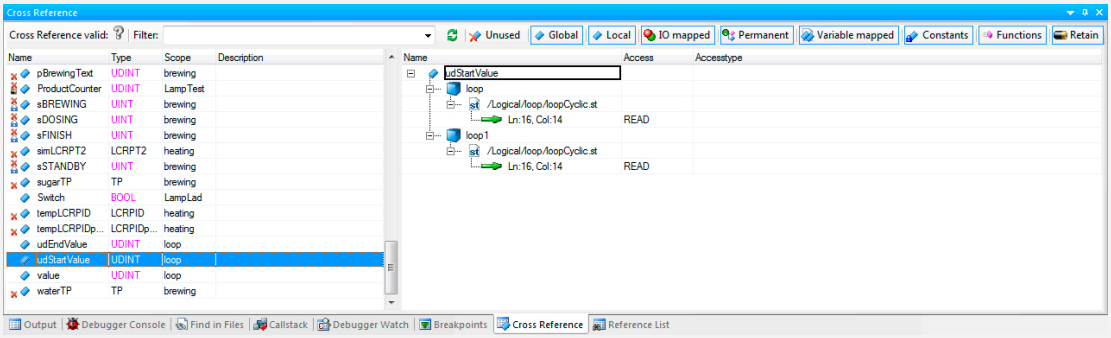


Figure 67: Display in the cross reference list

Project management \ Workspace

- General project settings
- Output window \ Cross-references
- Menus \ Project

5.3.2 Searching in files

If you are looking for parts of names, product IDs or comments, you can search for it in the project files.

To search for a variable, use **<Edit> / <Find and Replace – Find in Files>** or press **<CTRL> + <Shift> + <F>**.

A search term is entered into the dialog box, and the results of the search are displayed in the output window in the "Find in Files" tab.

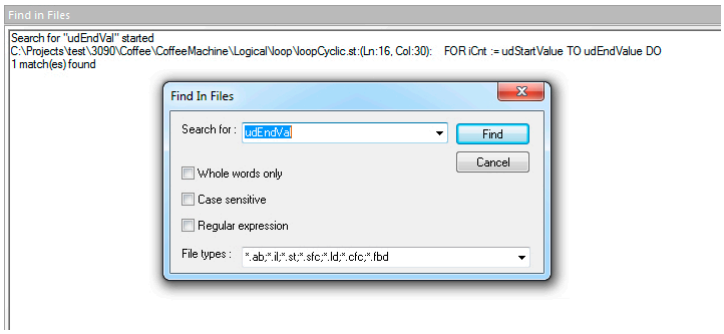


Figure 68: Searching in files

Double-clicking on a result in the output window opens the respective source file and places the cursor at the corresponding position.

Project management \ Workspace \ Find in files

6 Making preparations for servicing

It is necessary during the configuration, commissioning and testing of the application to prepare the machine or system for service activities that may occur later.

6.1 System Diagnostics Manager (SDM)

System Diagnostics Manager (SDM) contained in Automation Runtime V3.0 and higher can be used to diagnose the controller using a standard web browser from any location (Intranet or Internet).

The only requirement for these diagnostics is an Ethernet connection to the controller.

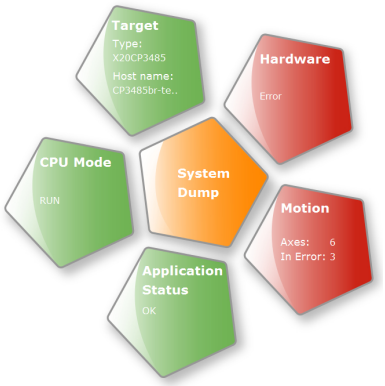


Figure 69: SDM startup screen

Diagnostics and service \ Diagnostics tools \ System Diagnostics Manager

6.1.1 Enabling the SDM

SDM is automatically activated when creating a new project. The SDM configuration is opened in Physical View using the **<Configuration>** option in the controller's shortcut menu.

System Diagnostics Manager requires the web server service, which is also an integral component of Automation Runtime.

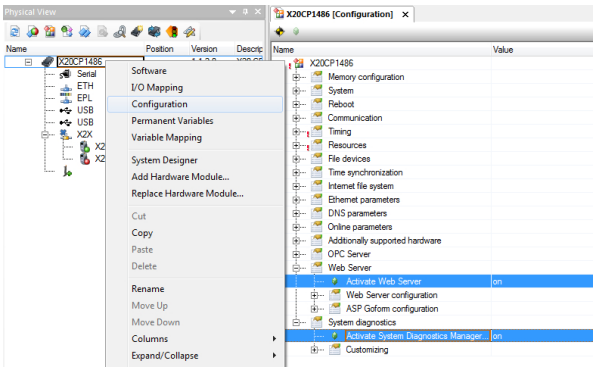


Figure 70: Opening the configuration, SDM and web server settings

Exercise: Check the System Diagnostics Manager configuration

Check whether the web server and System Diagnostics Manager are enabled in the Automation Runtime configuration.

1) Open the AR configuration from the controller's shortcut menu in the Physical View.

   The options "Web Server" and "System Diagnostics" must be enabled.

6.1.2 Accessing the SDM

The connection to the SDM is made via the controller's IP address or hostname. The SDM is accessed using a web browser via the link "**http://ip-address/SDM**".

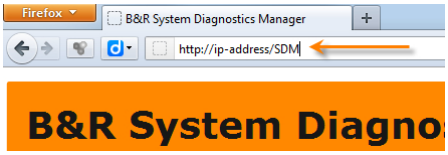
Step 1 - Connect PC to controller	Step 2 - View SDM in web browser
	

Figure 71: Connecting PC to controller via network cable

Figure 72: Accessing the SDM via web browser

Table 8: Overview – "Establishing a connection to the SDM"

The target system IP address can be checked in the controller's Ethernet configuration. if there is already an active connection to the controller, SDM can be opened using the **<Extras> / <System Diagnostics Manager>** menu option.

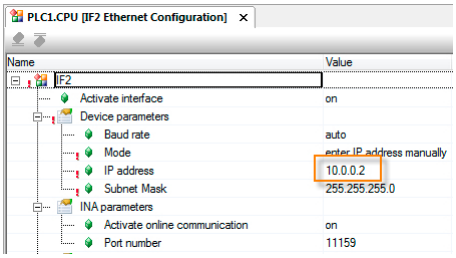


Figure 73: Check the network settings in the controller's Ethernet configuration

Diagnostics and service \ Diagnostics tools \ System Diagnostics Manager

Exercise: Access SDM from a web browser

Enter the URL for accessing the SDM: e.g. http://10.0.0.2/SDM

1) Start the web browser
2) Enter the URL for accessing the SDM.

Exercise: Evaluate Logger with SDM and in Automation Studio

Assumption: The system has booted into SERVICE mode for no apparent reason. Unfortunately, Automation Studio is not available on site. Since System Diagnostics Manager is enabled on the target system, it is possible to establish a TCP/IP connection to the controller using a web browser.

Once the SDM is opened, the Logger entries can be displayed from the navigation menu (Logger). Upload the Logger file "**$arlogsys**" and open it up with Logger in Automation Studio.

1) Establish a connection to the SDM and change to the "Logger" page
2) Upload the "$arlogsys" module.
3) Save the file using the .br file extension.
4) Open the Logger in Automation Studio.
5) Load the Logger file with the .br file extension.
6) Highlight the error entry in the Logger and press the **<F1>** key to receive detailed information

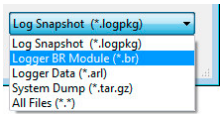


Figure 74: Select the file type with the .br file extension in Automation Studio

If a system dump was generated in the System Diagnostics Manager with the option "Parameters + Data Files", then it can also be opened in the Logger. The Logger files contained in the system dump are shown after being opened in Automation Studio.

Automation Runtime events are shown in the SDM Logger without additional supplementary information. This can only be displayed in Automation Studio.

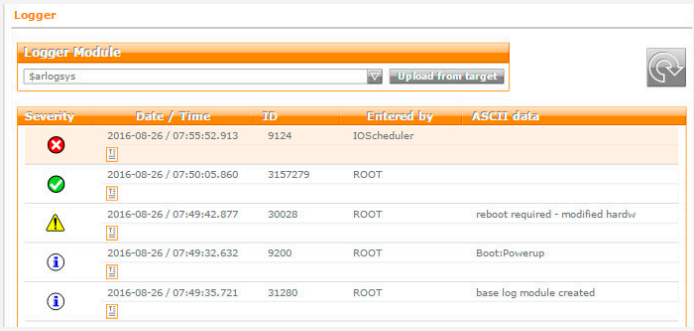


Figure 75: Logger entries in System Diagnostics Manager

The online data must be deselected in Automation Studio's Logger; otherwise, both the online entries as well as the entries loaded from the file are displayed.

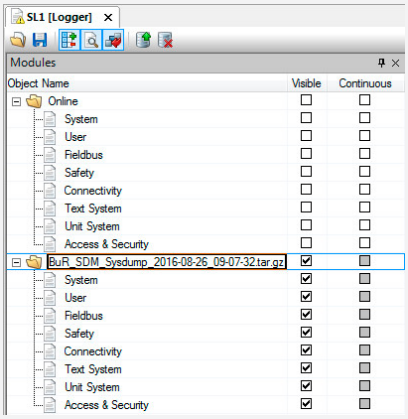


Figure 76: Disabling the online Logger entries

6.2 Query the status of the battery

Depending on the used target system, a battery is used for data buffering. Further details about this are available in the data sheet of the used controller.

The backup battery for the real-time clock and the nonvolatile variables (retain, permanent) being used in the application can be monitored from the application itself.

Figure 77: "4A0006.00-000" lithium battery

Note the replacement of the battery in the service instructions for the machine / system. The life of the battery is specified in the documentation for the controller being used.

The battery status can be queried in a number of ways:

- Using the AsHW library in the application
- Using the controller's I/O mapping
- Using the online information dialog box
- Using the System Diagnostics Manager

The I/O allocation is opened using the **<I/O Allocation>** option in the Physical View shortcut menu for the controller. The variable attached to the "BatteryStatusCPU" can be evaluated in the application program as needed.

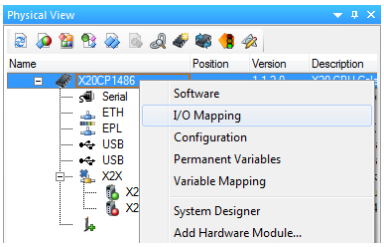


Figure 78: Opening the controller's I/O mapping

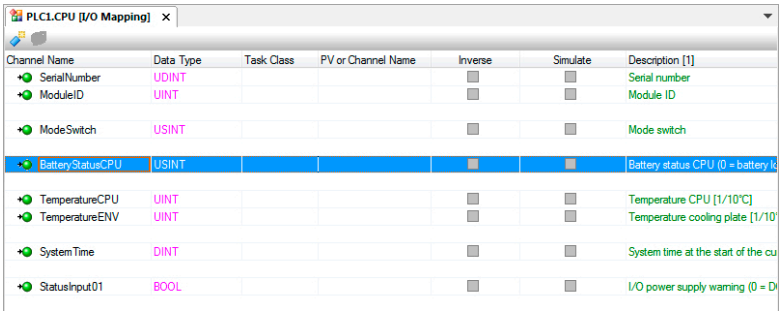


Figure 79: Querying the battery status in the controllers I/O allocation

The values of the status data points in the I/O mapping of the controller can be monitored using monitor mode.

see: "Monitor and force I/O" on page 26

6.3 Runtime Utility Center service tool

Runtime Utility Center also includes functions useful for commissioning, maintenance, diagnostics and servicing. The Runtime Utility Center is present on the Automation Studio installation media device and can be installed individually.

Runtime Utility Center is started in Automation Studio by selecting <**Extras**> / <**Runtime Utility Center**>.

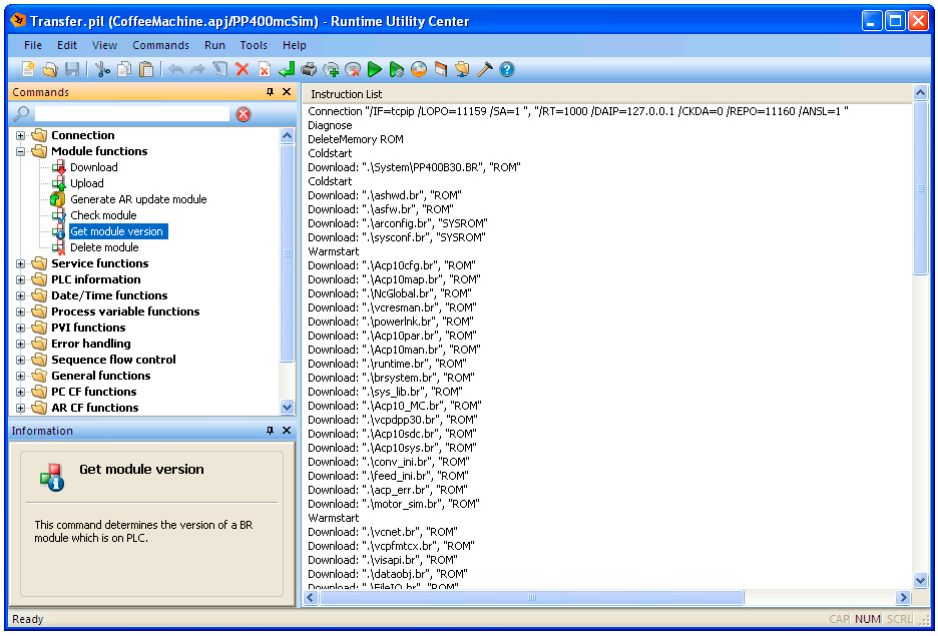


Figure 80: Runtime Utility Center

After an Automation Studio project is built, a project list file (.pil) is generated that is displayed when Runtime Utility Center is opened.

Diagnostics and service \ Service tools \ Runtime Utility Center

6.3.1 Backing up and restoring variable values

One function of Runtime Utility Center is to load variable values from the controller and to restore them at a later point in time.

Exercise: Save the variable values.

Due to mechanical damage to the system, the CPU must be replaced. To prevent e.g. recipe variables or other process data from being lost, the necessary information on the CPU can be uploaded using Runtime Utility Center and then transferred later to the new CPU.

Create a Runtime Utility Center list that saves the values of variables in the "Loop" task.

1) Open Runtime Utility Center from Automation Studio.
2) Create a new list by selecting **<File> / <New>** from the main menu

3) Insert the command for establishing a connection by selecting **<Command> / <Connection>**

   The IP address of the target system needs to be entered in the connection settings.

4) Insert the command for loading the variable list by selecting **<Command> / <Process variable functions> / <Variable list>**

   Only the variables in the "**Loop**" task are being backed up in this example. The list can be stored to any directory.

5) Execute the functions with **<F5>**.

Diagnostics and service \ Service tools \ Runtime Utility Center \ Operation \ Commands \ List functions

Diagnostics and service \ Service tools \ Runtime Utility Center \ Operation \ Commands \ Establish connection, wait for reconnection

Diagnostics and service \ Service tools \ Runtime Utility Center \ Operation \ Menus \ Start

The variable values from the "Loop" task are backed up using the directory and filename specified.

Exercise: Restore the variable values

The variable values backed up in the last task now need to be transferred to the controller using Runtime Utility Center.

Create a Runtime Utility Center list that restores the values of variables.

1) Open Runtime Utility Center in Automation Studio Create a new list by selecting <**File**> / <**New**> from the menu
2) Insert the command for establishing a connection by selecting <**Commands**> / <**Connection**>
3) The IP address of the target system needs to be entered in the connection settings.
4) Insert the command for writing the variable list by selecting <**Command**> / <**Process variable functions**> / <**Transfer variable list to PLC**>

   The variable list saved in the last task can be selected in the <**Browse**> dialog box.

5) Execute the functions with <**F5**>.

The variable values saved in the file are written to the corresponding variables in the "**Loop**" task.

If every variable from every task is backed up and then transferred back to the controller, be aware that writing to variables sequentially can cause unexpected behavior in the process.

If this situation is still necessary, it is recommended to put the controller in SERVICE mode first.

6.3.2 Passing on the project with Runtime Utility Center

Completed Runtime Utility Center project lists can be passed on as executable applications for servicing and installing on machines.

The only thing necessary for this is a PC with a physical connection to the CPU.

Exercise: Reinstall the controller

The controller should be reinstalled without using Automation Studio. Start Runtime Utility Center in Automation Studio after the project has been built. Installation package creation is started in Runtime Utility Center by selecting <**Extras**> / <**Create installation package**>.

Requirements:

- Executable Automation Runtime installed on the controller.
- The IP address of the CPU must be known.

Creating an installation package

- Compile the project in Automation Studio.
- Start Runtime Utility Center in Automation Studio by selecting **<Extras> / <Runtime Utility Center>**
- Generate a Runtime Utility Center installation image by selecting **<Extras> / <Create installation package>**
- After the destination directory is specified, the generation process begins by pressing the **<Start>** button.
  The Runtime Utility Center can be closed once the creation process has been completed.

An executable image for the project installation has been created without the aid of Automation Studio.

Passing on and executing an installation package

- Open Windows Explorer and switch to the destination directory specified during the installation package creation process.
- Execute the "**Start.bat"** batch file

To pass on the Runtime Utility Center installation to others, the contents of the destination directory can be copied to a data storage device.

The **"Start.bat"** file needs to be run on the target computer.

Diagnostics and service \ Service tool \ Runtime Utility Center \ Creating a list / data medium

7 Summary

Automation Studio offers several different tools for localizing problems and errors.

They need to be used sensibly in combination with analytical thinking. To be able to use these diagnostic tools effectively, it is necessary to get an overview of the situation, clarify the general conditions and examine these conditions from a certain distance.

Figure 81: Diagnostics

Only then can the circumstances be cleared up and analyzed in detail. A comprehensive overview of potential errors can be achieved by excluding and reducing the number of possible error sources, making it considerably easier to correct any errors that may still occur.

Seminars and training modules

At the Automation Academy, you'll develop the skills you need in no time!
Our seminars make it possible for you to improve your knowledge in the field of automation engineering.

Automation Studio seminars and training modules

Programming and configuration	Diagnostics and service
SEM210 – Basics SEM246 – IEC 61131-3 programming language ST* SEM250 – Memory management and data storage SEM410 – Integrated motion control* SEM441 – Motion control: Electronic gears and cams** SEM480 – Hydraulics** SEM1110 – Axis groups and path-controlled movements** SEM510 – Integrated safety technology* SEM540 – Safe motion control*** SEM610 – Integrated visualization*	SEM920 – Diagnostics and service for end users SEM920 – Diagnostics and service with Automation Studio SEM950 – POWERLINK configuration and diagnostics* If you don't happen to find a seminar on our website that suits your needs, keep in mind that we also offer customized seminars that we can set up in coordination with your sales representatives: SEM099 – Individual training day Please visit our website for more information**** .****: **www.br-automation.com/academy**

Overview of 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: Electronic gears and cams TM1110 – Integrated Motion Control (Axis Groups) TM1111 – Integrated Motion Control (Path Controlled Movements) TM450 – Motion Control Concept and Configuration 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 TM611 – Working with mapp View TM630 – Visualization Programming Guide TM640 – Alarm System, Trends and Diagnostics TM670 – Advanced Visual Components TM920 – Diagnostics and service TM923 – Diagnostics and Service with Automation Studio TM950 – POWERLINK Configuration and Diagnostics TM280 – Condition Monitoring for Vibration Measurement TM480 – The Basics of Hydraulics TM481 – Valve-based Hydraulic Drives TM482 – Hydraulic Servo Pump Drives TM490 – Printing Machine Technology In addition to the printed version, our training modules are also available on our website for download as electronic documents (login required): Visit our website for more information: **www.br-automation.com/academy**

Process control seminars and training modules

Process control standard seminars	Process control training modules
SEM841 – Process control training: Basic 1 SEM842 – Process control training: Basic 2 SEM890 – Advanced Process Control Solutions	TM800 – APROL System Concept TM810 – APROL Setup, Configuration and Recovery TM811 – APROL Runtime System TM812 – APROL Operator Management TM813 – APROL web portal TM820 – APROL solutions TM830 – APROL Project Engineering TM835 – APROL ST-SFC Configuration TM840 – APROL Parameter Management and Recipes TM850 – APROL Controller Configuration and INA TM860 – APROL Library Engineering TM865 – APROL Library Guide Book TM870 – APROL Python Programming TM880 – APROL reporting TM890 – The Basics of LINUX

* SEM210 - Basics is a prerequisite for this seminar.
** SEM410 - Integrated motion control is a prerequisite for this seminar.
*** SEM410 - Integrated motion control and SEM510 - Integrated safety technology are prerequisites for this seminar.
**** Our seminars are listed in the Academy\Seminars area of the website.
**** Seminar titles may vary by country. Not all seminars are available in every country.