

Automation Studio Diagnostics

TM223



Requirements

Training modules:	TM210 – The Basics of Automation Studio TM211 – Automation Studio Online Communication TM213 – Automation Runtime
Software	Automation Studio 3.0.90
Hardware	X20CP1485

TABLE OF CONTENTS

1 INTRODUCTION.....	4
1.1 Training module objectives.....	5
2 THE CORRECT DIAGNOSTIC TOOL.....	6
2.1 Checklist.....	7
2.2 Overview of diagnostic tools.....	9
3 COLLECTING SYSTEM INFORMATION.....	10
3.1 CPU operating status.....	10
3.2 Error analysis in Logger.....	12
4 MONITORING AND ANALYZING PROCESS VALUES.....	17
4.1 Monitoring and modifying variables.....	17
4.2 Recording variables in real time.....	22
4.3 I/O monitor.....	26
5 SOFTWARE ANALYSIS DURING PROGRAMMING.....	27
5.1 Configuring the Profiler and evaluating data.....	28
5.2 Searching for errors in the source code.....	33
5.3 Using variables in programs.....	39
6 MAKING PREPARATIONS FOR SERVICING.....	41
6.1 Using the System Diagnostics Manager.....	41
6.2 Querying and displaying the battery status.....	45
6.3 Diagnostics Tool Runtime Utility Center.....	46
7 SUMMARY.....	50

1 INTRODUCTION

Automation Studio and Automation Runtime provide several different diagnostic functions for machine **programming** and **commissioning**, runtime system **analysis** and **service operations**.

The diagnostics process begins by selecting the right tool for the application or situation at hand.



Diagnostics

Diagnostic functions have been specially incorporated into Automation Runtime that allow the user to analyze the data it records either with or without Automation Studio.

The **System Diagnostics Manager** is an integral component beginning with Automation Runtime V3.0.

1.1 Training module objectives

You will learn how to work with the various diagnostic tools through the use of selected examples that illustrate different diagnostic possibilities during programming, commissioning and servicing.

You will learn how to...

- Use the correct diagnostic tool
- Collect system information to analyze problems
- Record process values
- Troubleshoot program errors
- Configure the target system correctly to handle subsequent diagnostic operations

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.

Level	Time	Error Number	OS Task	Error Description	ASCII Data	Binary Data
1 Error / Syst...	2011-06-27 12:44:34.961000	9124	IO Scheduler	TC#1 maximum cycle time violation		
2 Warning	2011-06-27 06:48:14.579000	27060	ROOT	NV memory (HDD/CF/RAM) has been exchanged	HDD/CF/RAM was changed	00 00 00 00
3 Warning	2011-06-27 06:47:53.665000	30028	ROOT	Carried out reboot	reboot required - modified hardware description (h...	00 00 00 00
4 Warning	2011-06-27 06:47:47.443000	9200	ROOT	Warning: System halted because of power loss	Boot.Powerup	00 00 00 00
5 Information	2011-06-27 06:47:48.140000	31280	ROOT	AR logger module created	base log module created	00 20 03 00

Example of a cycle time violation

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 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 would be the **wrong diagnostic tool** to determine this type of error. This problem can only be detected using the Profiler, which displays the chronological sequence of individual tasks as well as the time needed for them to complete.

2.1 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 (e.g. development or support departments) to solve problems more quickly by providing the actual data instead of requiring further inquiries.



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.

The correct diagnostic tool

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)
- Accessing uninitialized pointers

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 explanations in the checklist
- 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 TM220 – The Service Technician on the Job).

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: Checklist for relaying information

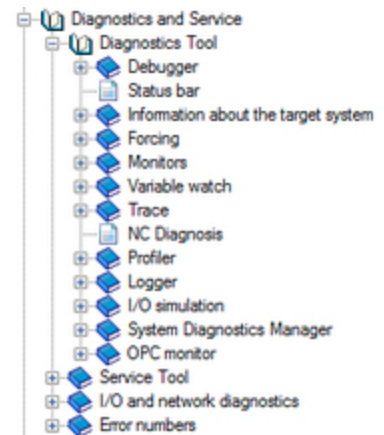
2.2 Overview of diagnostic tools

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 documentation – a constant companion during development, commissioning and servicing – provides detailed information about the various diagnostic tools.



Diagnostics in the help documentation

Task: Open the help documentation for the diagnostic tools

The diagnostic tools are covered in the section "Diagnostics and service".



Diagnostics and Service



This training manual applies some of the possible diagnostic tools using different tasks.

3 COLLECTING SYSTEM INFORMATION

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

Collecting system information

Status bar	Displays information about the connection status.
Information about the target system	Displays memory information, battery status and the time configuration.
Logger	Displays events that occur on the target system at runtime.
Software/Hardware monitor	Displays I/O channels and values of variables.
System Diagnostics Manager	The 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: Collecting system information

Requirements for the tasks in Chapter 3

The following exercises can be done with any Automation Studio project that uses the necessary hardware.

The descriptions and images in this chapter refer to the X20 CPU-based project designed in both Training Manuals TM210 (The Basics of Automation Studio) as well TM213 (Automation Runtime).



X20 CPU

- Executable project on the controller
- Online connection between Automation Studio and the controller

3.1 CPU operating status

There are several ways for Automation Studio to evaluate the operating status of a controller.

- Status bar for determining the operating status
- Information about the target system

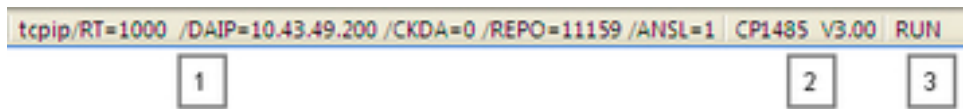


The information can also be estimated by using the System Diagnostics Manager ([Using the System Diagnostics Manager](#)).

3.1.1 Status bar

The status bar is located near the bottom of Automation Studio (bottom row).

The status bar includes the following information:



Status bar

- 1 Connection status and settings
- 2 CPU type and Automation Runtime version (online)
- 3 Operating status of the controller (see TM213 Automation Runtime)

Project Management / The Workspace / Status bar

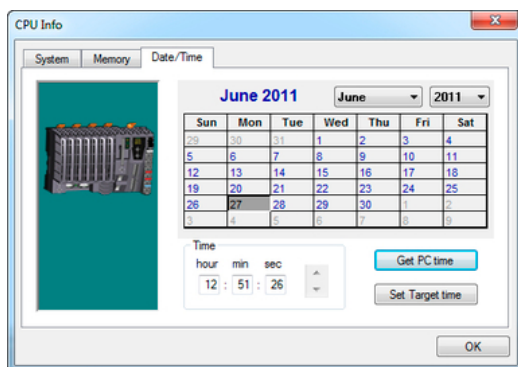
3.1.2 Information about the target system

Target system information can be queried from the **<Online / Info>** menu item or **<OnlineInfo...>** from the CPU's shortcut menu (physical view) whenever there is an online connection.

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

The "Info" dialog box includes the following information:

- Boot type
- Status of the internal backup battery
- CPU type and Automation Runtime version (online)
- Node number for INA communication
- Available memory on the target system
- Option of setting the date/time of the target system manually or synchronizing it with the PC



Setting the date and time on the target system

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

3.1.3 Tasks in monitor mode

The **software configuration's** monitor mode makes it possible to check the status of the tasks on the target system.

- Comparison of tasks contained in the project with those on the target system
- Memory where the task is running
- Whether tasks are running or have been stopped

The software configuration's monitor mode is started or ended by clicking the monitor icon in the toolbar.



Activate monitor mode

The tasks in the Automation Studio project are located on the left side, while the tasks on the target system are displayed on the right.

Object Name	Version	Transfer To	Size (bytes)	Description
Project				
Exception				
Cyclic #1 - [10 ms]				
loop	1.00.0	UserROM	632	A new program
Cyclic #2 - [20 ms]				
Cyclic #3 - [50 ms]				
Cyclic #4 - [100 ms]				
Lamp Test	1.00.0	UserROM	388	Control lamp and switch
loop1	1.00.0	UserROM	632	A new program
Logger	1.00.0	UserROM	4504	AsArLog example application
Cyclic #5 - [200 ms]				
Cyclic #6 - [500 ms]				
Cyclic #7 - [1000 ms]				

Object Name	Version	Memory	Size (...)	State	Date
Target					
Exception					
Cyclic #1 - [10 ms]					
loop	1.00.0	UserROM	632	Running	27.06.2011 13:27:58
Cyclic #2 - [20 ms]					
Cyclic #3 - [50 ms]					
Cyclic #4 - [100 ms]					
Lamp Test	1.00.0	UserROM	388	Running	27.06.2011 07:16:22
Logger	1.00.0	UserROM	4504	Stopped	27.06.2011 13:27:59
Cyclic #5 - [200 ms]					
Cyclic #6 - [500 ms]					
Cyclic #7 - [1000 ms]					

Task overview in monitor mode

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



[Diagnostics and Service / Diagnostics Tools / Monitors / Software configuration in monitor mode](#)

3.2 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.

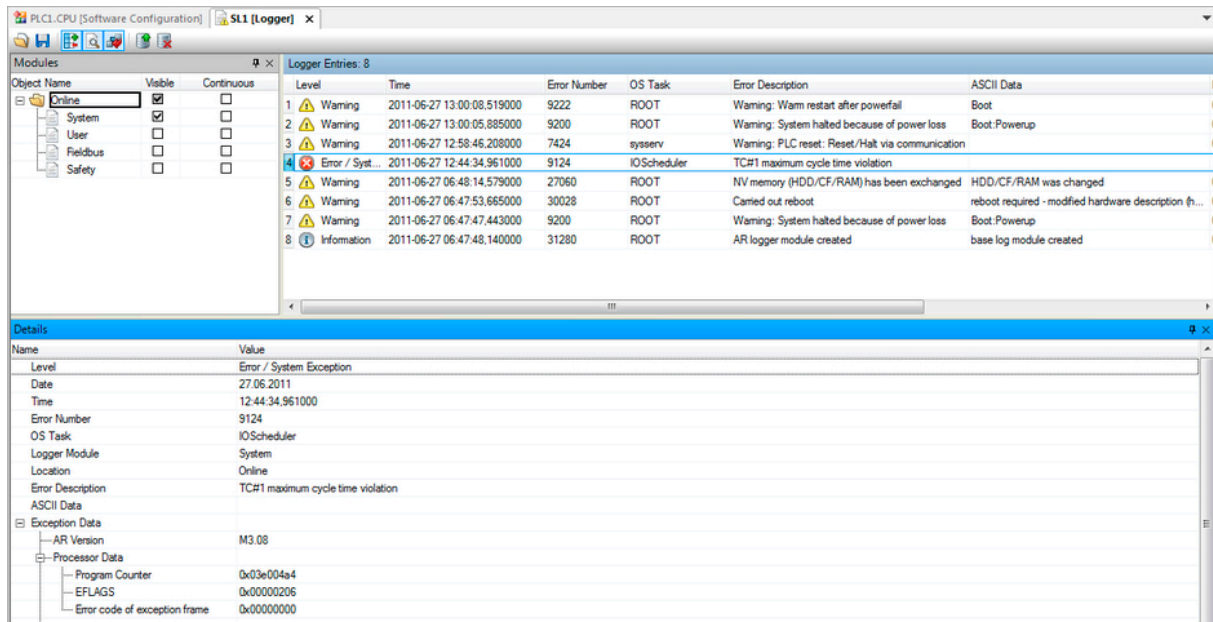
This log is stored in the controller's memory.



[Diagnostics and service / Diagnostic tools / Logger](#)

3.2.1 Logger with an active online connection

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



Logger window



The entries displayed in this image show the events logged by Automation Runtime after transferring CompactFlash data and booting the CPU.

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

Using the **"Loop"** task created in TM213, a cycle time violation can be achieved by incrementing the **"udiEndValue"** variable in the variable monitor.

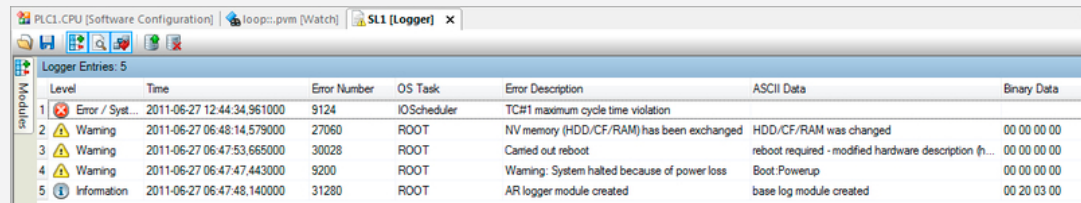
Once an online connection has been reestablished between Automation Studio and the target system (restart), open the Logger and look for the cause of the boot into service mode.

- 1) Open the variable monitor (Watch) in the **"Loop"** task's software configuration.
- 2) Increment the **"udiEndValue"** variable until a cycle time violation occurs (loss of connection and restart in service mode).
- 3) Open the Logger from the physical view. Look for the cause of booting in service mode.
- 4) Select the entry and press F1.

Collecting system information



Once opened, the Logger indicates the cause of booting in service mode.



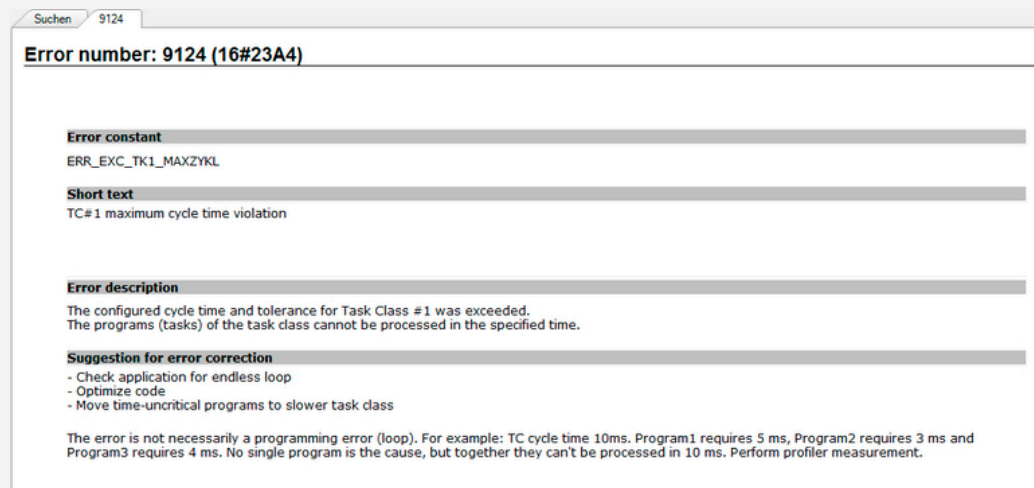
Level	Time	Error Number	OS Task	Error Description	ASCII Data	Binary Data
1 Error / Syst...	2011-06-27 12:44:34.961000	9124	IOScheduler	TC#1 maximum cycle time violation		
2 Warning	2011-06-27 06:48:14.579000	27060	ROOT	NV memory (HDD/CF/RAM) has been exchanged	HDD/CF/RAM was changed	00 00 00 00
3 Warning	2011-06-27 06:47:53.665000	30028	ROOT	Carried out reboot	reboot required - modified hardware description (h...	00 00 00 00
4 Warning	2011-06-27 06:47:47.443000	9200	ROOT	Warning: System halted because of power loss	Boot: Powerup	00 00 00 00
5 Information	2011-06-27 06:47:48.140000	31280	ROOT	AR logger module created	base log module created	00 20 03 00

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.



The Automation Studio help documentation displays information about the error indicated in the Logger's **"Error Number"** column.



Suchen 9124

Error number: 9124 (16#23A4)

Error constant
ERR_EXC_TK1_MAXZYKL

Short text
TC#1 maximum cycle time violation

Error description
The configured cycle time and tolerance for Task Class #1 was exceeded.
The programs (tasks) of the task class cannot be processed in the specified time.

Suggestion for error correction
- Check application for endless loop
- Optimize code
- Move time-uncritical programs to slower task class

The error is not necessarily a programming error (loop). For example: TC cycle time 10ms. Program1 requires 5 ms, Program2 requires 3 ms and Program3 requires 4 ms. No single program is the cause, but together they can't be processed in 10 ms. Perform profiler measurement.

Context-sensitive help for Automation Runtime errors

3.2.2 Offline evaluation of Logger data

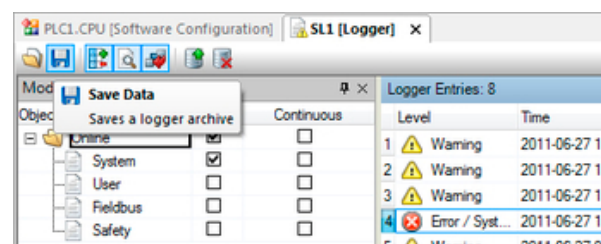
Logger records can also be evaluated **"offline"**.

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

Application case

Logger data is backed up on the system by a service technician. Retrieved files are analyzed in Automation Studio by the programmer.

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



Saving Logger entries

3.2.3 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 **AsArLog** library.



Programming / Libraries / Configuration, system information, runtime control / AsArLog

Applications:

- Logging service actions (e.g. replacing batteries)
- Logging user actions
- Entering exceptions that occur in the exception task

Generate user log data

Create a user logbook in the existing Automation Studio project. The event "**This is a test logger entry**" with user error number "**55000**" is to be entered in this logbook.



This task can be performed quite easily with a sample included in Automation Studio.

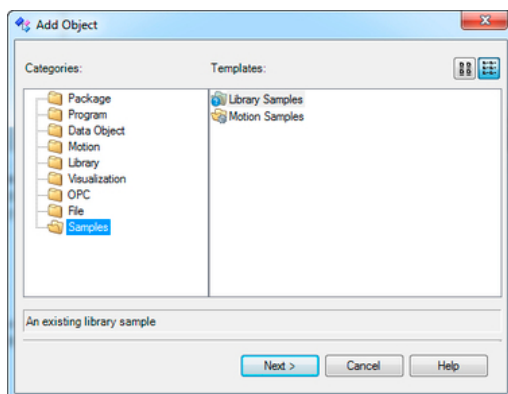
User error numbers are only allowed in the range "**50000 - 59999**".



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

Procedure:

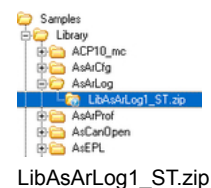
- Importing an example can be done in the Logical View by selecting **<Add object>** / **<Samples>** / **<Library samples>** from the shortcut menu.



Add sample to the project

Collecting system information

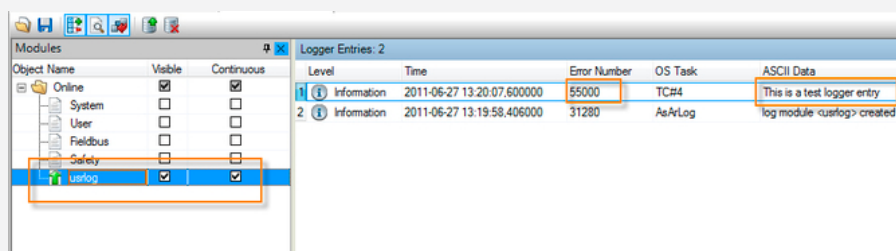
- Select the "LibAsArLog1_ST.zip" sample package in the "AsArLog" directory.
- Insert the program into the active configuration (automatic).
- Transfer the program to the controller.
- Do the steps regarding the Automation Studio help
- for creating the Logger module and entering the event with the help of the variable watch
 - Logger.Step = 1: Creates a module called "usrlog"
 - Logger.Step = 3: Writes the Logger information
- Upload the user log file in Automation Studio.



If the CPU is still in service mode after the last task, it can be booted back into RUN mode by performing a warm restart in Automation Studio.



Writing the values 1 and 3 to the step sequencer variable "Logger.Step" generates an entry in the "usrlog" Logger module.



Generating a user log file

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

Variable watch (Watch / NC Watch)	The variable monitor allows values of variables on the target system as well as the current status of an axis to be displayed, monitored and modified.
Variable oscilloscope (Trace / NC Trace)	A trace makes it possible to record several variables in real time over a set period of time. This data can be uploaded using Automation Studio and displayed in the form of a curve. The NC Trace function allows real-time data to be recorded directly from the drive.
I/O assignment in monitor mode	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.
Programming language monitors	The monitors for each of the programming languages include functions such as line coverage and signal flow display.

Table: Monitoring and analyzing process values



More information about analyzing NC data can be found in the motion training modules (TM4xx).

Requirements for the examples of this chapter

The descriptions and images in this chapter refer to the Automation Studio project "CoffeeMachine" using the "PC-based Simulation Runtime (ARsim)

- Transfer the „CoffeeMachine“ projekt to the „PC Based Simulation Runtime (ARsim)“
- Online connection between Automation Studio and ARsim

4.1 Monitoring and modifying variables

The variable monitor (Watch) allows the values of variables on the target system to be displayed, monitored and modified.

Variable lists can be saved in the variable monitor for the different diagnostic and function tests and reused at a later time.

Diagnose the "CoffeeMachine" application

Operate the "CoffeeMachine" application using the variable monitor in Automation Studio.

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

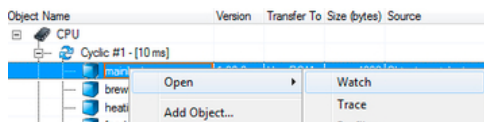
The following process variables are required for this task:

Action	Range	Process variable(s)
Select coffee type	0-2	gMainLogic.par.coffeeType

Action	Range	Process variable(s)
Coffee ingredients	0-100	gMainLogic.par.receipe.coffee gMainLogic.par.receipe.milk gMainLogic.par.receipe.sugar gMainLogic.par.receipe.water
Coffee price	-	gMainLogic.par.recipe.price
Payment	0.10	gMainLogic.par.givenMoney
Switching on/off	0-1	gMainLogic.cmd.switchOnOff
Start preparation	1	diStartCoffee
Water temperature	-	gHeating.status.actTemp
Messages	-	gMainLogic.cmd.vis.messageIndex
Process flowchart	-	gMainLogic.status.progressStep

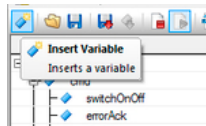
Step 1:

- Transfer the „**CoffeeMachine**“ project to the „PC Based Simulation Runtime (ARsim)“
- Open the variable monitor (Watch) from the "mainlogic" task in the software configuration.



Opening the variable monitor (Watch)

- Insert the variables from the table above using the toolbar or by pressing the <Ins> key.



Inserting variables



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

Name	Type	Scope	Force	Value
gMainLogic	main_type	global		
cmd	main_cmd_type			
switchOnOff	BOOL			FALSE
errorAck	BOOL			FALSE
start	BOOL			FALSE
vis	main_cmd_vis_t			
par	main_par_type			
coffeeType	SINT			0
givenMoney	REAL			0.0
receipe	main_par_receip			
price	REAL			1.69
setTemp	REAL			80.0
milk	REAL			100.0
sugar	REAL			30.0
coffee	REAL			60.0
water	REAL			150.0
status	main_status_type			
money	main_status_moi			
progressStep	USINT			0
curPage	UINT			0
curLanguage	UINT			0
startProgressStep	USINT			0
gHeating	heating_type	global		
cmd	heating_cmd_type			
start	BOOL			FALSE
updatePIDpar	BOOL			FALSE
status	heating_status_t			
setTempOK	BOOL			FALSE
actTemp	REAL			0.0

Displaying the variables in the variable monitor

Diagnosing the process sequence with the variable monitor

After the coffee machine is started (**gMainLogic.cmd.switchOnOff = 1**), the water is warmed up (**gHeating.status.actTemp**).

Once a certain temperature has been reached, the selected coffee type (**gMainLogic.par.coffeeType = 0,1,2**) gets ready for preparation.

Once the (simulated) coins have been inserted (**gMainLogic.par.givenMoney**) with the same or higher value than the price of the coffee (**gMainLogic.par.recipe.price**), the preparation of the coffee can be started (**diStartCoffee = 1**).

The status of the process sequence is output via status process variables (**gMainLogic.cmd.vis.messageIndex / gMainLogic.status.progressStep**).

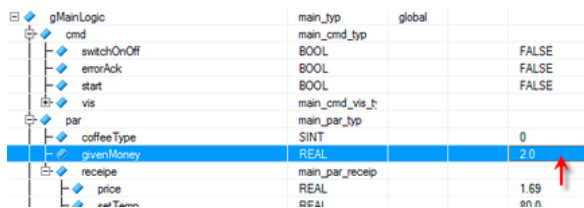
The kind of coffee can be preselected with the coffee type variable (**gMainLogic.par.coffeeType**). Sugar (**gMainLogic.par.recipe.sugar**) and milk (**gMainLogic.par.recipe.milk**) can also be modified.

Step 2:

- Turn on the coffee machine (**gMainLogic.cmd.switchOnOff = 1**).
- The "**gMainLogic.cmd.vis.messageIndex = 2**" variable is returned by the process sequence to indicate that the target temperature has been reached.

Monitoring and analyzing process values

- Enter a value for the "**gMainLogic.par.givenMoney**" that is the same or higher than the value of the "**gMainLogic.par.recipe.price**" variable.



Variable	main_type	global	Value
cmd	main_cmd_type		
switchOnOff	BOOL		FALSE
errorAck	BOOL		FALSE
start	BOOL		FALSE
vis	main_cmd_vis_t		
par	main_par_type		
coffeeType	SINT		0
givenMoney	REAL		2.0
recipe	main_par_recipe		
price	REAL		1.69
realTime	REAL		0.0

Simulate inserting the coins

- Start preparation of the coffee by setting the "**diStartCoffee**" variable to **1**.
- Monitor the "**gMainLogic.status.progressStep**" variable. This must move from the value 1 (filling) and reach the value 2 (ready to take away).

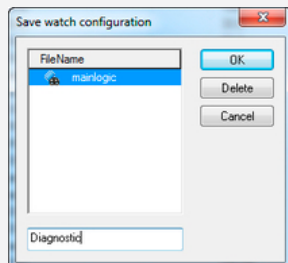


The variable list in the variable monitor should be saved for later use. This allows the process sequence to be reproduced at any time.



Variables on the controller can be monitored and modified using the variable monitor (Watch). In addition to their values, further information about the variables such as their scope, data type, I/O type, etc. is displayed.

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



Saving the variable list



Diagnostics and Service / Diagnostics Tools / Variable watch

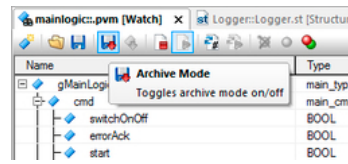
4.1.1 Writing to variable values simultaneously

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

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

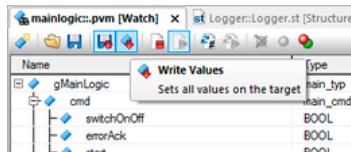
In order to enter several values in the variable monitor **without their being immediately applied**, archive mode must be activated.

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



Starting archive mode

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



Changing all values in archive mode



It is important to consider which variables have been inserted into the variable monitor when dealing with synchronous writing operations. Using archive mode incorrectly in this case can lead to undesirable behavior when changes are made to the process sequence.



[Diagnostics and Service / Diagnostics Tools / Variable watch / Archive mode](#)

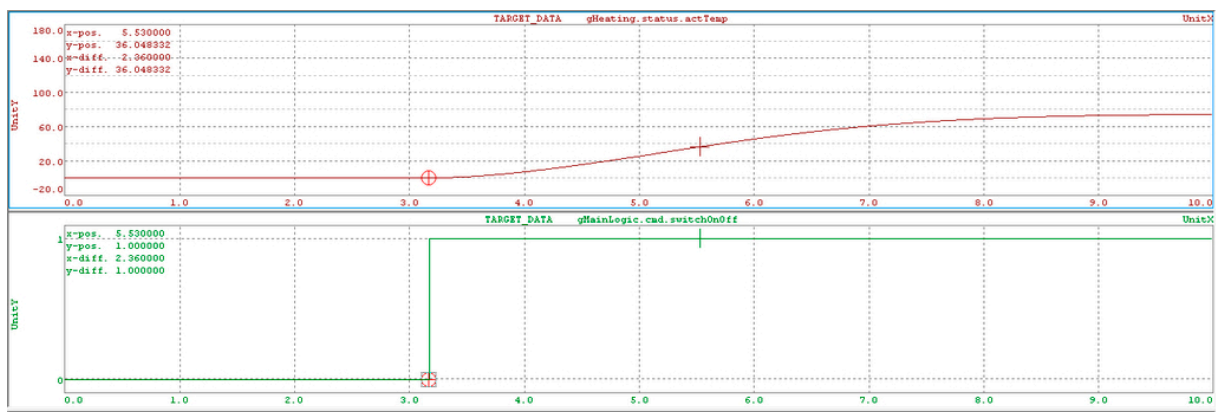
4.2 Recording variables in real time

When using the variable monitor, 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 synchronously in the context of the task class.

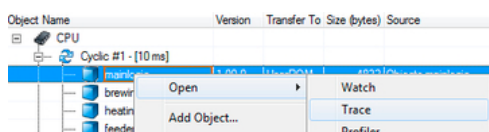


Example of a Trace recording

This 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 **<Open>** / **<Trace>** from the shortcut menu.



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“** icon in the toolbar.

Variables to be recorded can be added to the trace configuration by clicking the **„Insert new variable“** icon in the toolbar.

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 this 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 for this task:

Action	Range	Process variable(s)
Select coffee type	0-2	gMainLogic.par.coffeeType
Switching on/off	0-1	gMainLogic.cmd.switchOnOff
Water temperature	-	gHeating.status.actTemp

Step 1:

- Open the Trace dialog box for the "mainlogic" task.
- Insert a new Trace configuration.
- Insert the process variables needed for the recording.



The Trace configuration looks like this:

Name	State	Type
gHeating.status.actTemp	✓	REAL
gMainLogic.cmd.switchOnOff	✓	BOOL
gMainLogic.cmd.errorAck	✓	BOOL

Trace configuration

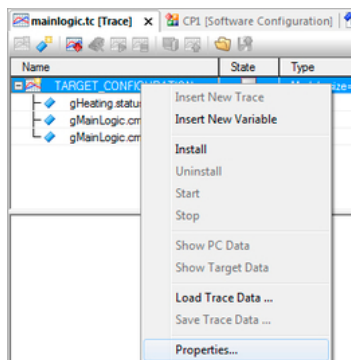
Values are recorded cyclically in the context of the task class. The period and start condition of the recording can be configured in the Trace configuration's properties.

In this example, these three variables are set to be recorded every 10 ms.

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

Step 2:

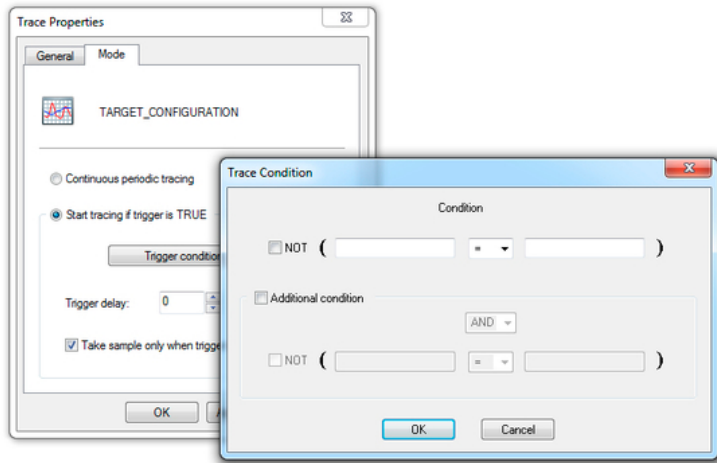
- Open the properties dialog box for the Trace configuration.



Trace Properties

Monitoring and analyzing process values

- Under the "**General**" tab, set the size of the recording buffer to 30,000 entries.
- The "**Mode**" tab allows you to configure a trigger condition to start the recording (**gMainLogic.cmd.switchOnOff = 1**).



Trace Condition



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

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

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.

Step 3:

- Open the variable monitor and insert the necessary variables according to the table above.
- Set the "**gMainLogic.cmd.switchOnOff**" variable.
- Modify the "**gMainLogic.par.coffeeType**" between 0, 1 and 2.

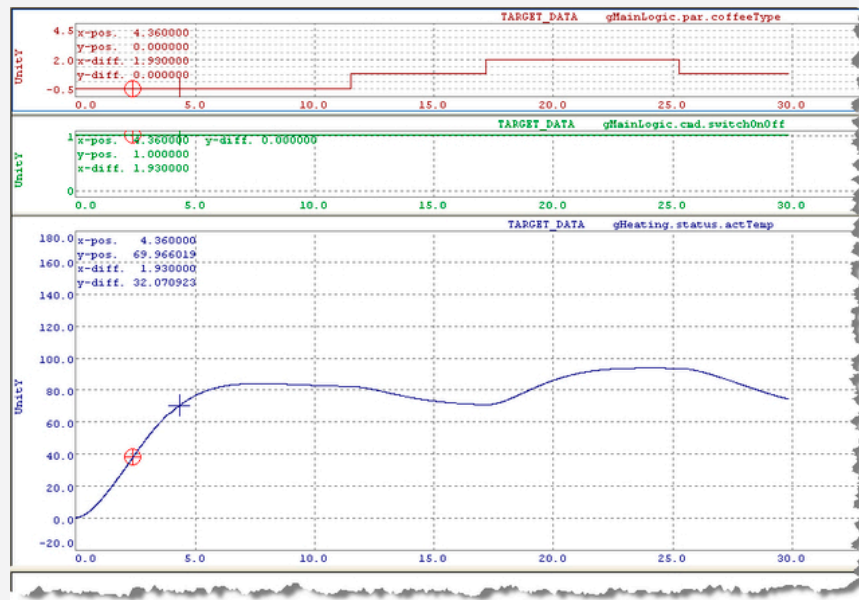


If the Watch configuration for the task in Monitoring and modifying variables was saved, it will reopen automatically when opening the watch window.

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 variable monitor. 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):



Trace data

Changes in value over time can be analyzed using the measurement cursor, which is the same for all variables on the time axis.



Diagnostics and Service / Diagnostics Tool / Trace

4.3 I/O monitor

Double-clicking on an I/O module in the physical view opens the I/O mapping window.

If monitor mode is switched on and an online connection is present, then the physical I/O states will be displayed.



Monitor

The screenshot shows the 'I/O Mapping' window for 'PLC1.CPU.IF6.ST1'. It contains a table of digital inputs and a 'Watch' window for the 'LampTest.pvm' project.

Channel Name	Data Type	Physical Value	Force	Force Value	PV or Channel Value	PV or Channel Name	Inv.
ModuleOk	BOOL	TRUE	<input type="checkbox"/>	FALSE			<input type="checkbox"/>
DigitalInput01	BOOL	FALSE	<input checked="" type="checkbox"/>	TRUE	TRUE	LampTest Switch	<input type="checkbox"/>
DigitalInput02	BOOL	FALSE	<input type="checkbox"/>	FALSE			<input type="checkbox"/>
DigitalInput03	BOOL	FALSE	<input type="checkbox"/>	FALSE			<input type="checkbox"/>
DigitalInput04	BOOL	FALSE	<input type="checkbox"/>	FALSE			<input type="checkbox"/>
DigitalInput05	BOOL	FALSE	<input type="checkbox"/>	FALSE			<input type="checkbox"/>
DigitalInput06	BOOL	FALSE	<input type="checkbox"/>	FALSE			<input type="checkbox"/>
DigitalInput07	BOOL	FALSE	<input type="checkbox"/>	FALSE			<input type="checkbox"/>
DigitalInput08	BOOL	FALSE	<input type="checkbox"/>	FALSE			<input type="checkbox"/>
DigitalInput09	BOOL	FALSE	<input type="checkbox"/>	FALSE			<input type="checkbox"/>
DigitalInput10	BOOL	FALSE	<input type="checkbox"/>	FALSE			<input type="checkbox"/>
DigitalInput11	BOOL	FALSE	<input type="checkbox"/>	FALSE			<input type="checkbox"/>
DigitalInput12	BOOL	FALSE	<input type="checkbox"/>	FALSE			<input type="checkbox"/>

Name	Type	Scope	Force	Value
Lamp	BOOL	local		TRUE
Switch	BOOL	local	<input checked="" type="checkbox"/>	TRUE

I/O configuration in 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.

4.3.1 I/O input data points

The "force" value of a channel on an input card (e.g. X20DI) is "simulated" by Automation Runtime. The application program's process sequence then works with the "force" value and not with the actual input state.

4.3.2 I/O output data points

The "Force" value of a channel on an output card (e.g. X20DO) 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 manually using the **<Online> / <Force> / <Global Force Off>** menu item.

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

Profiler	The Profiler can be used to measure and display important system data such as task runtimes, system and stack loads, etc.
Status variables	Status variables are used to evaluate the status of or error in a function call within the application program.
Debugger	The debugger makes it easier to search for errors in the source code of a program or library.
Line coverage	Line coverage indicates the lines of the source code that are currently being executed.
Output window	The output window is used to display information about ongoing processes, e.g. building, downloading, generating the cross-reference list, displaying search results, etc.

Requirements for the examples of this chapter

The following exercises can be done with any Automation Studio project that uses the necessary hardware.

The descriptions and images in this chapter refer to the X20 CPU-based project designed in both Training Manual TM210 (The Basics of Automation Studio) as well as TM213 (Automation Runtime).



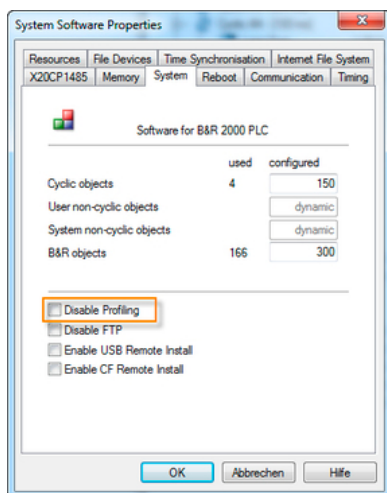
X20 CPU

- Executable project on the controller
- Online connection between Automation Studio and the controller

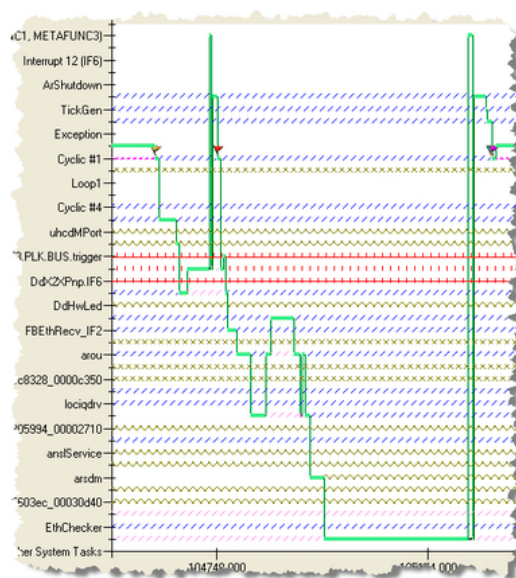
5.1 Configuring the Profiler and evaluating data

Automation Runtime can be configured to automatically record the runtime environment.

The Profiler can be configured using the CPU node's **<Properties>** menu item in the software configuration.



Profiler in the system software properties



Example for a profiling

If an error occurs (e.g. cycle time violation), the log can be loaded from the target system at any time and analyzed in Automation Studio.

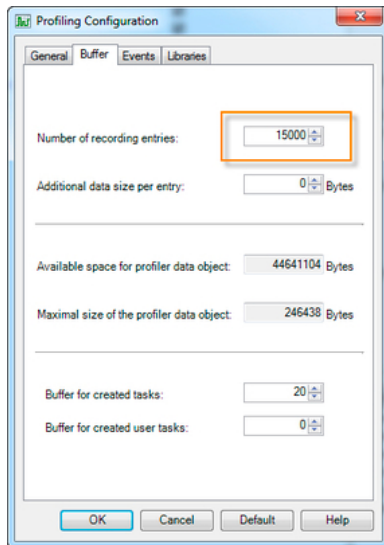


Diagnostics and Service / Diagnostics Tool / Profiler

5.1.1 Configuring the Profiler

The Profiler is opened from the software configuration using the **<Open>** / **<Profiler>** menu item.

The Profiler's configuration dialog box can be opened by clicking on the „**Configuration**“ icon in the toolbar.



Configuring the number of recording entries for the Profiler



When recording, it is recommended to log all of the events under the **"Events"** tab. This makes filtering possible later if there is an error or the Profiler data is passed on to someone else.

A change in the Profiler configuration is transferred to the target system by clicking the **"Install"** icon in the toolbar.



[Diagnostics and Service / Diagnostics Tool / Profiler / Preparing the profiler](#)

5.1.2 Analyzing Profiler data

Profiler data can be uploaded from the target system to the Automation Studio Profiler in order to perform runtime analysis of cyclical programs.

Cause a cycle time violation and evaluate the Profiler data

A cycle time violation can occur by increasing the value of the **"udiEndValue"** 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.



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.

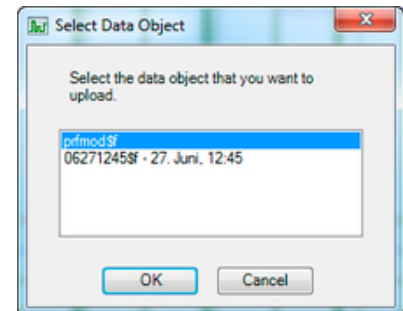
Procedure:

- Cause a cycle time violation by setting the **"udiEndValue"** variable to the value 500000 in the variable monitor.

Software analysis during programming

- After restarting into service mode, open the Profiler in the software configuration by selecting **<Open>** / **<Profiler>** menu item.

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. The corresponding file can be selected from a list during the uploading process.



Selecting the Profiler data



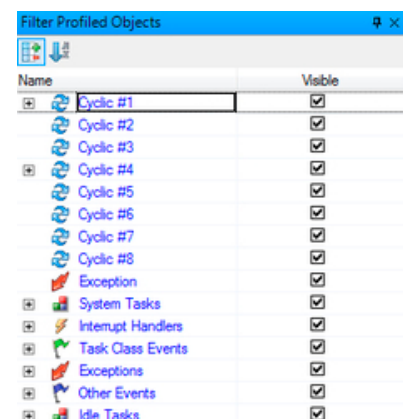
The **"Zoom"** button in the toolbar can be used to set the range or area for the Profiler data to be viewed. When analyzing the data, it is recommended to start at 100%. This can be done simply by pressing the **<ESC>** key.

The Project Explorer can be hidden in order to get as much on the screen as possible.

Profiler data can be filtered to limit the events being displayed.

Which events should be displayed depends on the situation itself.

When searching for the cause of the cycle time violation, the data can be filtered as shown in the image to the right.



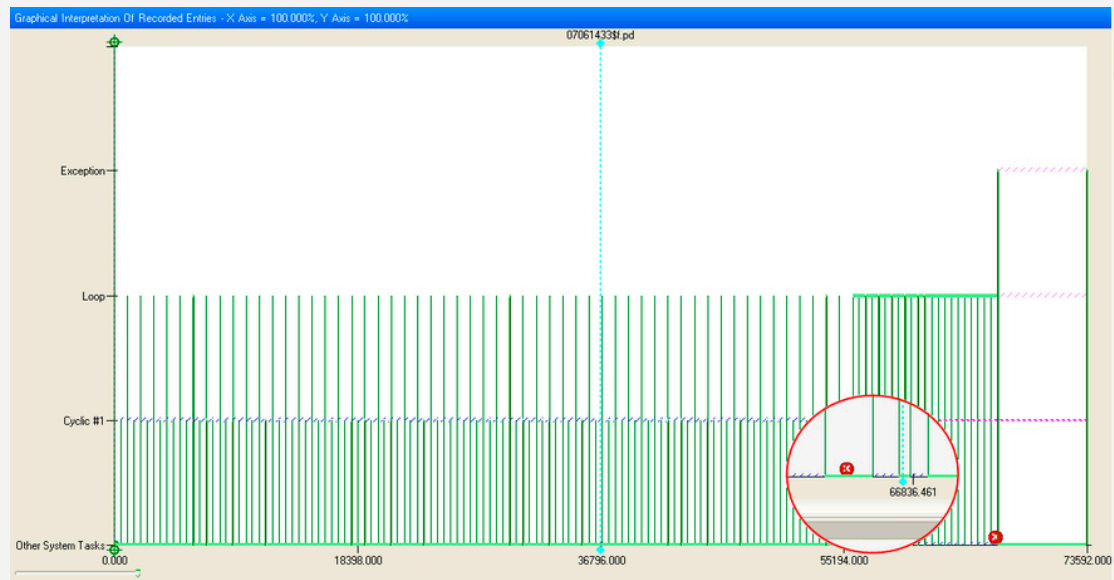
: Filtering Profiler data



Diagnostics and service / Diagnostics Tool / Profiler / Recording profiler data / Analyzing profiler data



At a certain point in time (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 the icon.



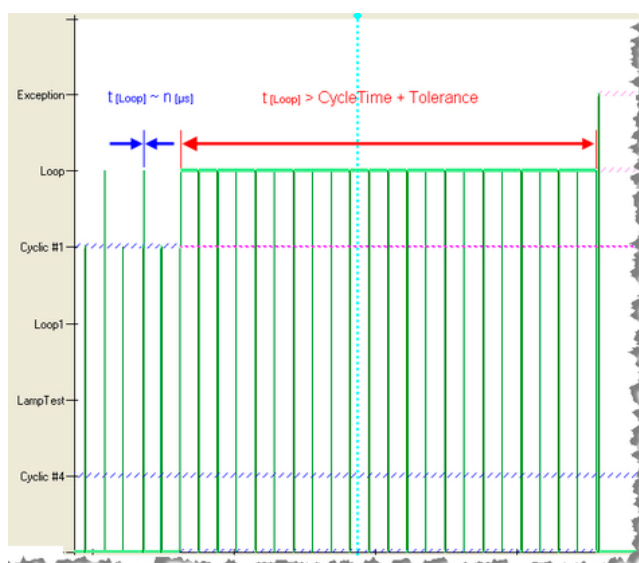
Exception in the Profiler data

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

As you can see in the following image, the "Loop" task usually finishes executing within only a few microseconds (blue arrow); a cycle time violation occurs if the configured cycle time plus tolerance is exceeded (red arrow).



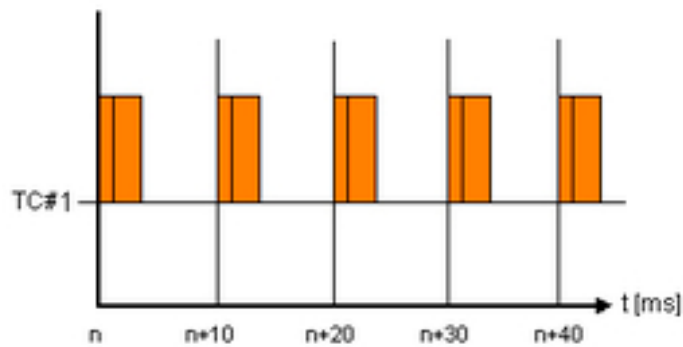
Determining the cycle time violation

Software analysis during programming

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.

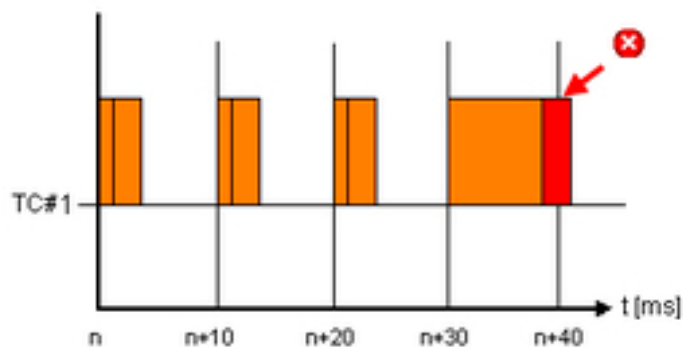
Example:

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



Time diagram

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.



Time diagram

The sequence of events can be analyzed chronologically by evaluating the raw data („RawData“ icon in the toolbar).

Raw Data Of Recorded Entries			
Nr.	Name	Event	Event Description
465	TickGen	0x000...	'TickGen' is now running - 'IO scheduler' 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	DdC2Acc.IF6	0x007...	'DdC2Acc.IF6' is now running - 'Cyclic #1' was waiting
474	IEpN2H.IF3	0x007...	'IEpN2H.IF3' is now running - 'DdC2Acc.IF6' was ready
475	DdC2Acc.IF6	0x007...	'DdC2Acc.IF6' is now running - 'IEpN2H.IF3' was delayed and waiting

Raw data for a Profiler recording

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.

5.1.3 Application performance

The Profiler is used to analyze the performance of tasks on the CPU.

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							

Analyzing the CPU load with the Profiler



Diagnostics and Service / Diagnostics Tools / Profiler / Recording profiler data / Analyzing profiler data / Analyzing profiler data in tables, form

5.2 Searching for errors in the source code

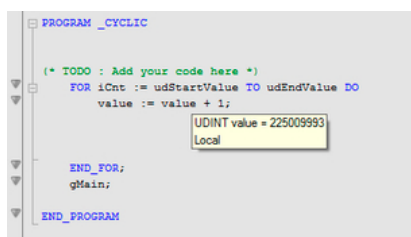
When it comes to software, statistics have shown that there are usually around two to three errors contained in every 1,000 lines of code.

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

5.2.1 Monitor mode in the program editor

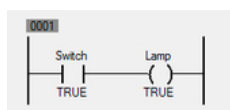
Monitor mode is available for each programming language and allows variables to be observed in several ways:

- Value of a variable as a tooltip in both textual and visual programming languages.



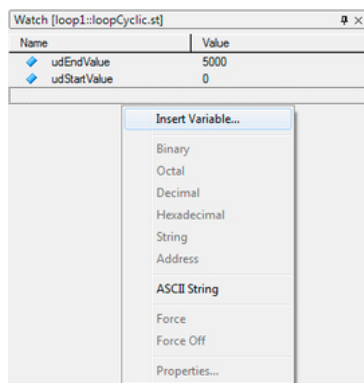
Tooltip in the source code

- Value directly by the variable in visual programming languages.



Visual programming language in monitor mode

- In the variable monitor (Watch) window.



Variable monitor view

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

5.2.2 Power Flow

The path of a signal (Power Flow) can be displayed in visual programming languages, e.g. Ladder Diagram.

Power Flow can be enabled via the **"Powerflow"** icon in the toolbar.

Power Flow in Ladder Diagram

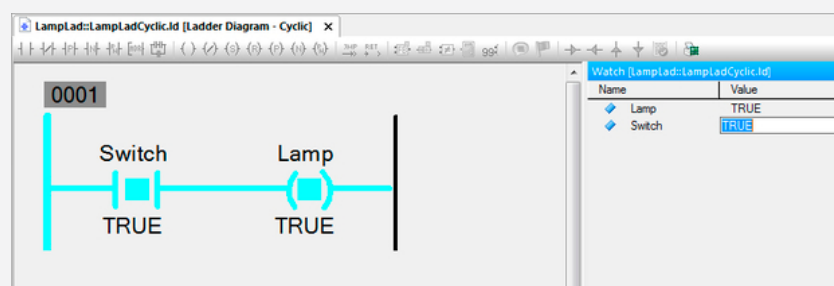
Turn on Power Flow in the **"LampTest"** program (from TM210).

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

- 1) Open the **"LampTest"** program with an online connection established.
- 2) Enable monitor mode.
- 3) Add the **"Switch"** and **"Lamp"** variables to the variable monitor.
- 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.



Power Flow enabled in Ladder Diagram

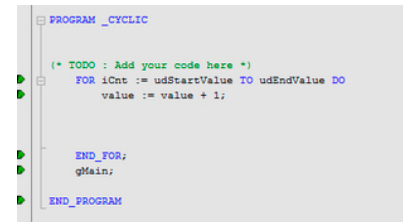


Diagnostics and Service / Diagnostics Tool / Monitors / Programming languages in monitor mode / Powerflow

5.2.3 Line coverage

If line coverage is enabled for textual 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.



Line coverage



Diagnostics und Service / Diagnostics Tool / Monitors / Programming languages in monitor mode / Line coverage

5.2.4 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 1131 languages or Automation Basic) before each of these operations is carried out.

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



Programming / Libraries / IEC Check library

5.2.5 Debugging the source code

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

Debugging possibilities in Automation Studio

- Line-by-line execution of a program while monitoring variables.
- Stopping the application at certain moments with user-defined breakpoints.
- Stepping into called functions (e.g. in library functions / FBKs as long as the source code is available).

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, use a loop to initialize the array with any value (e.g. 10).

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

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

Error description: The limits of the array are exceeded (0-10) since the array only contains 10 elements (0-9). This type of error is often difficult to detect at first glance and causes the program to overwrite the next variable memory location.

Step 1:

- Create a new Structured Text program called "**dbgTest**" in the logical view.
- Open the variable declaration dialog box for the program and insert the variables "**AlarmBuffer**" (data type USINT[0..9]) and "**index**" (data type UINT).
- Insert the program code for the task into the cyclic program.



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.

Name	Type	Scope	Force	Value
AlarmBuffer	USINT[0..9]	local		
AlarmBuffer[0]	USINT			10
AlarmBuffer[1]	USINT			10
AlarmBuffer[2]	USINT			10
AlarmBuffer[3]	USINT			10
AlarmBuffer[4]	USINT			10
AlarmBuffer[5]	USINT			10
AlarmBuffer[6]	USINT			10
AlarmBuffer[7]	USINT			10
AlarmBuffer[8]	USINT			10
AlarmBuffer[9]	USINT			10

Variable monitor in monitor mode

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

The debugger can be enabled in monitor mode when the program editor is open.



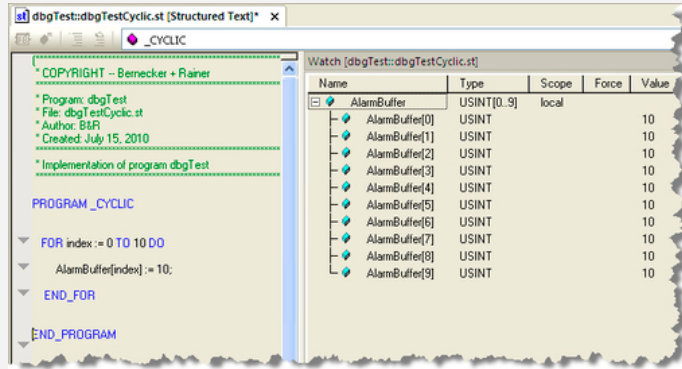
Monitor

Step 2:

- With monitor mode turned on, add the "AlarmBuffer" variable to the right side of the variable monitor (Watch) window.



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



Monitor mode in the program editor

Step 3:

- 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.



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



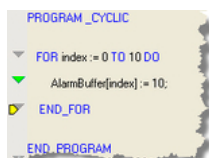
If the debugger hits a breakpoint, then the active line is indicated by a yellow marker.



Active line in the debugger

Step 4:

- Change the elements of the **"AlarmBuffer"** array to the value 0 in the variable monitor.
- The **"Step Into"** command (**<F11>**) can be used to execute the program code one line at a time. The active line is always indicated by the yellow marker.



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.

Watch [dbgTest::dbgTestCyclic.st]				
Name	Type	Scope	Force	Value
AlarmBuffer	USINT[0..9]	local		
AlarmBuffer[0]	USINT			10
AlarmBuffer[1]	USINT			10
AlarmBuffer[2]	USINT			10
AlarmBuffer[3]	USINT			0
AlarmBuffer[4]	USINT			0

Step-by-step writing to the variables

Step 5:

- Continue through each step with <F11> until the new value has been assigned to the last element of the array.



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 value outside of the array.



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



[Diagnostics and Service / Diagnostics Tool / Debugger](#)

5.2.6 Evaluating status variables and return values

Any value returned by a function must also be evaluated in the program itself.

Function of return values:

The following example shows a function call. This function returns a status that can be used to determine whether an error has occurred during the call (ERROR) or if access is not possible in this program cycle (BUSY).

```
0: ("Open Ethernet Interface")
Server.TcpOpen_0.enable := 1;
Server.TcpOpen_0.pllAddr := 0; ("Listen on all TCP/IP Interfaces")
Server.TcpOpen_0.port := 12000; ("Port to listen")
Server.TcpOpen_0.options := 0;
Server.TcpOpen_0; ("Call the Function")

IF Server.TcpOpen_0.status = 0 THEN ("TcpOpen successful")
  Server.sStep := 5;
ELSEIF Server.TcpOpen_0.status = ERR_FUB_BUSY THEN ("TcpOpen not finished -> redo")
  ("Busy")
ELSE ("Goto Error Step")
  Server.sStep := 100;
END_IF

5:
Server.Tcpclotl_0.enable := 1;
Server.Tcpclotl_0.ident := Server.TcpOpen_0.ident; ("Connection Ident from AsTCP_TCP_Open")
Server.Tcpclotl_0.ioctl := tcpSO_LINGER_SET; ("Set Linger Options")
Server.Tcpclotl_0.pData := ADDR[Server.linger_opt];
Server.Tcpclotl_0.dataLen := SIZEOF[Server.linger_opt];
Server.Tcpclotl_0;

IF Server.Tcpclotl_0.status = 0 THEN ("Tcpclotl successful")
  Server.sStep := 10;
ELSEIF Server.Tcpclotl_0.status = ERR_FUB_BUSY THEN ("Tcpclotl not finished -> redo")
  ("Busy")
ELSE ("Goto Error Step")
  Server.sStep := 100;
END_IF
```

Status evaluation of function blocks

If the status of the function (the program code in the red box) is not evaluated accurately, then the function itself or the subsequent program might not execute correctly.

5.3 Using variables in programs

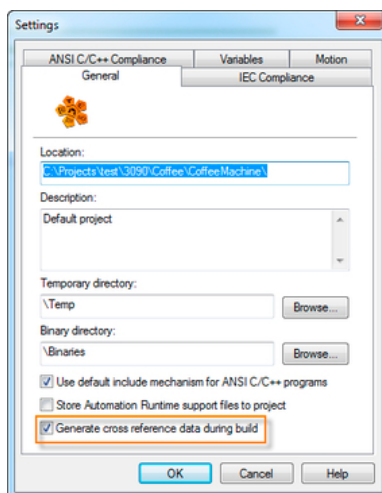
The proper usage of variables in the different programs given in the logical view can be checked by creating a cross-reference list or explicitly searching for a known variable name.

5.3.1 Cross-reference list

The cross-reference list indicates which process variables, functions and function blocks are actually being used 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.

The cross-reference list must be enabled via the <**Project**> / <**Settings**> menu item before it can be generated.



Enabling the cross-reference list during a project build

Generating a cross-reference list

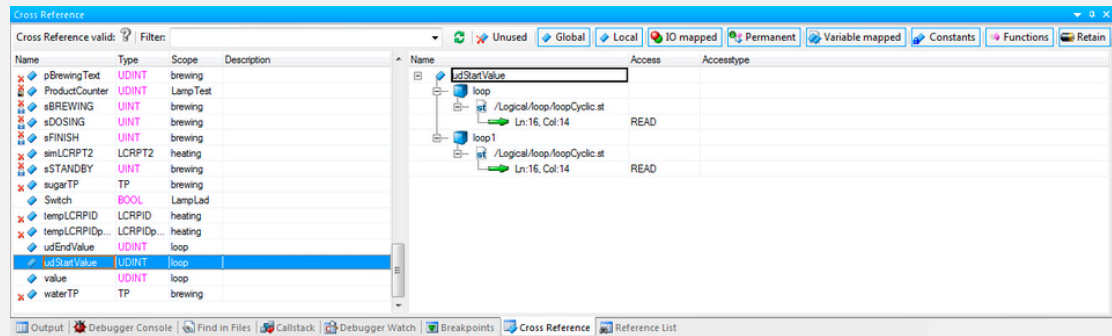
Create a cross-reference list in an open project (e.g. the main project from TM210).

- 1) Enable the cross-reference list in the project settings.
- 2) Build the project.



The cross-reference list of variables and their attributes can be analyzed in the output window (though results will vary depending on whether a project from TM210 or TM213 is being used).

If a variable is selected on the left side, its usage and the type of access will be displayed on the right side.



Result of cross-reference list generation



[Project Management / The Workspace / Output window / Cross-reference](#)

[Project Management / The Workspace / Menus / Project / Create cross-reference](#)

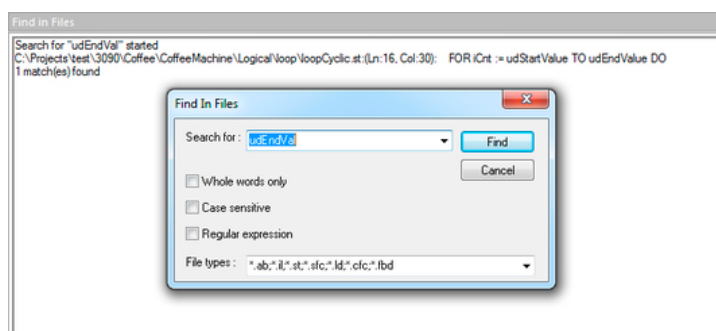
[Project Management / The Workspace / Menus / Project / Build cross reference](#)

5.3.2 Searching in files

If the name of a variable is known, it is possible to search for it and see how it is being used in the respective files.

This search is started with the **<Edit> / <Find and REplace - Find in Files>** menu item or the **<CTRL> + <Shift> + <F>** key combination.

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.



Searching in files

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

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 Using the System Diagnostics Manager

The 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.



Browser

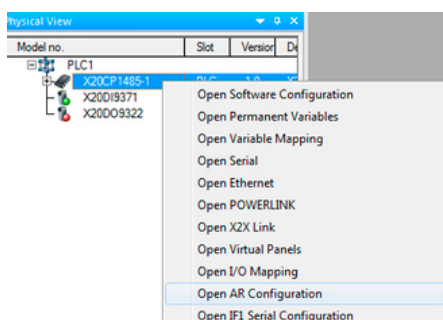


Diagnostics und Service / Diagnostics Tool / System Diagnostics Manager

6.1.1 Enabling the SDM

The SDM is enabled automatically whenever a new project is built with Automation Runtime version 3.0 or higher.

The SDM is configured from the physical view by opening up the AR configuration for the CPU node.



Opening the AR configuration



The System Diagnostics Manager requires services included in the Web Server, which is also an integral component of Automation Runtime.

Check the AR configuration for the SDM

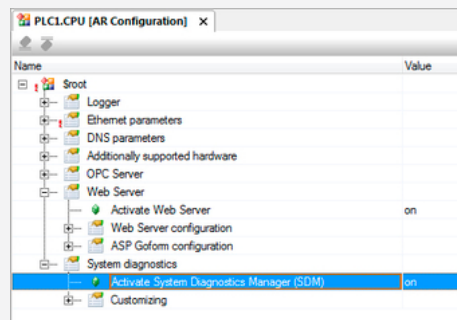
Look to see if the Web Server and the System Diagnostics Manager are enabled in the AR configuration.

Making preparations for servicing

- 1) Open the AR configuration from the CPU's node in the physical view.



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



Enabling the System Diagnostics Manager



In order for the SDM to function properly, there must be sufficient memory available on the target system.

On target systems with only 32 MB RAM, memory bottlenecks may occur when using Visual Components and/or motion applications. If this happens, the System Diagnostics Manager should be disabled in the AR configuration.

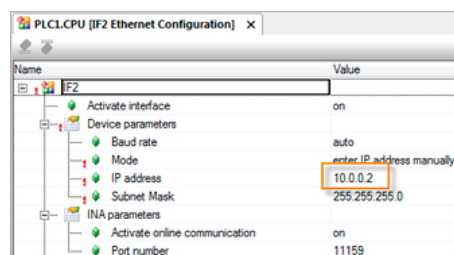
6.1.2 Accessing the SDM

The information in the **S**ystem **D**iagnostics **M**anager can be displayed using any web browser.

In order to gain access, the IP address of the target system must be known.



The target system IP address can be checked in the CPU's Ethernet configuration.



Opening the IFx Ethernet configuration

A plug-in is installed automatically for Internet Explorer (starting with version 7.x) to provide SVG (Scaleable Vector Graphics) support.



Diagnostics and Service / Diagnostics Tool / System Diagnostics Manager / SVG plug-in

Display the SDM with the URL "http://IP_Address/SDM"

Enter the URL for accessing the SDM.

E.g. http://10.0.0.2/SDM

- 1) Start Internet Explorer.
- 2) Enter the URL for accessing the SDM.



After pressing <Enter> to complete the URL, the SDM pages are loaded from the target system and shown in the browser. The left side of the SDM is used for navigation.

B&R System Diagnostics Manager

Perfection in Automation
www.br-automation.com

SDM

System - General

Operational Values

Node number:	204 / 0xCC
Current CPU mode:	RUN
Battery status:	OK
CPU temperature:	47 °C / 116.6 °F
Current CPU usage:	22% <input type="text"/> <input type="button" value="History"/>
Target time:	2011-06-27 / 15:17:32
Operating hours:	-
Power-on cycles:	-

Time Synchronisation

Time zone:	GMT
SNTP server:	disabled
SNTP client:	disabled
SNTP server 1:	-
SNTP server 2:	-
SNTP server 3:	-
SNTP server 4:	-
SNTP sync interval:	-
RTC sync interval:	-

Software Versions

Automation runtime:	M03.08
Visual components:	-
Motion control:	-
CNC software:	-

CPU Configuration

Host name:	brunnerhX20-p..
Default domain:	-
CPU mode switch:	0x04
Reboot mode:	
after reset:	service
after powerfall:	warm start
after change of CF/HD:	warm start
Preserve permanent PV memory:	
after change of CF/HD:	no
Profiling:	enabled
FTP:	enabled
USB remote install:	disabled
CF remote install:	disabled

Ethernet Network Devices

Default gateway:	172.16.0.1
IF2:	172.16.1.111
IF3.ETH:	192.168.101.240

© 2011 B&R , www.br-automation.com , Legal X20CP1485-1 | brunnerhX20-profiling | RUN | 2011-06-27 / 15:17:32 GMT

Displaying SDM pages in a web browser

Save the Logger entries in the System Diagnostics Manager and evaluate them in Automation Studio

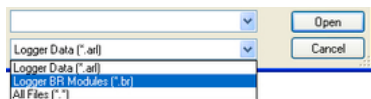
Situation:

The system has booted into service mode for no apparent reason. Unfortunately, Automation Studio is not available on site. Since the 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.

Making preparations for servicing

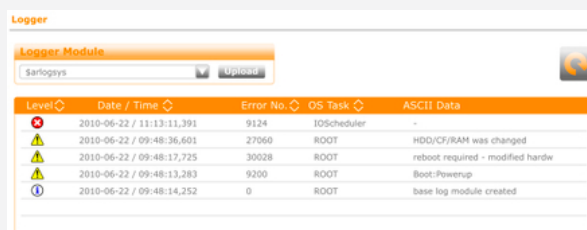
- 1) Establish a connection with the SDM and switch to the "Logger" page
- 2) Upload the "\$arlogsys" module.
- 3) Save the file using the .br file extension.
- 4) Open the Logger in Automation Studio.
- 5) Load the Logger file with the .br file extension.



File type with extension .br



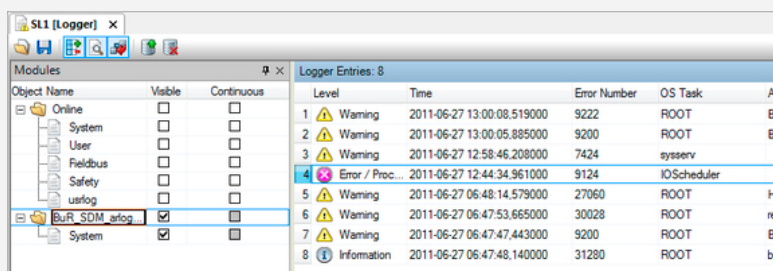
Automation Runtime events are shown in the SDM Logger without additional supplementary information. This can only be displayed in Automation Studio.



Level	Date / Time	Error No.	OS Task	ASCII Data
Warning	2010-06-22 / 11:13:11,391	9124	IOScheduler	-
Warning	2010-06-22 / 09:48:36,601	27060	ROOT	HDD/CF/RAM was changed
Warning	2010-06-22 / 09:48:17,725	30028	ROOT	reboot required - modified hardw
Warning	2010-06-22 / 09:48:13,283	9200	ROOT	Boot:Powerup
Information	2010-06-22 / 09:48:14,252	0	ROOT	base log module created

Logger entries in the System Diagnostics Manager

The online data must be deselected in Automation Studio's Logger; otherwise, both the online entries as well as the entries in the .BR module will be displayed.



Object Name	Visible	Continuous
Online	<input type="checkbox"/>	<input type="checkbox"/>
System	<input type="checkbox"/>	<input type="checkbox"/>
User	<input type="checkbox"/>	<input type="checkbox"/>
Fieldbus	<input type="checkbox"/>	<input type="checkbox"/>
Safety	<input type="checkbox"/>	<input type="checkbox"/>
usrlog	<input type="checkbox"/>	<input type="checkbox"/>
BuR_SDM_arlog	<input checked="" type="checkbox"/>	<input type="checkbox"/>
System	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Level	Time	Error Number	OS Task	A
1 Warning	2011-06-27 13:00:08.519000	9222	ROOT	B
2 Warning	2011-06-27 13:00:05.885000	9200	ROOT	B
3 Warning	2011-06-27 12:58:46.208000	7424	syserv	
4 Error / Proc...	2011-06-27 12:44:34.961000	9124	IOScheduler	
5 Warning	2011-06-27 06:48:14.579000	27060	ROOT	H
6 Warning	2011-06-27 06:47:53.665000	30028	ROOT	re
7 Warning	2011-06-27 06:47:47.443000	9200	ROOT	B
8 Information	2011-06-27 06:47:48.140000	31280	ROOT	bi

Disabling the online Logger entries

6.2 Querying and displaying the battery status

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.



Battery

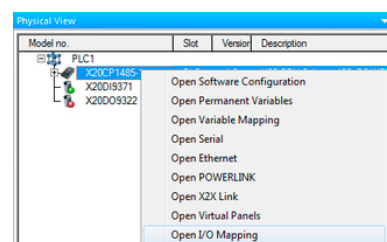


The battery must be changed according to the lifespan specified in the hardware documentation of the CPU while keeping the considerations of the system's service manual in mind.

The battery status can be queried in two ways:

- Using the ASHW library
- Via a variable in the I/O mapping window

The I/O mapping window can be opened using <Open I/O Mapping> from the CPU's shortcut menu in the physical view. The variable attached to the "BatteryStatusCPU" can be evaluated in the application program as needed.



Opening the CPU I/O mapping window

Channel Name	Data Type	Task Class	PV or Channel Name	Inverse	Simulate	Description [1]
SerialNumber	UDINT			<input type="checkbox"/>	<input type="checkbox"/>	Serial number
ModuleID	UINT			<input type="checkbox"/>	<input type="checkbox"/>	Module ID
ModeSwitch	USINT			<input type="checkbox"/>	<input type="checkbox"/>	Mode switch
BatteryStatusCPU	USINT			<input type="checkbox"/>	<input type="checkbox"/>	Battery status CPU (0 = battery is ok)
TemperatureCPU	UINT			<input type="checkbox"/>	<input type="checkbox"/>	Temperature CPU [1/10°C]
TemperatureENV	UINT			<input type="checkbox"/>	<input type="checkbox"/>	Temperature cooling plate [1/10°C]
SystemTime	DINT			<input type="checkbox"/>	<input type="checkbox"/>	System time at the start of the operation
StatusInput01	BOOL			<input type="checkbox"/>	<input type="checkbox"/>	I/O power supply warning (0 = OK)

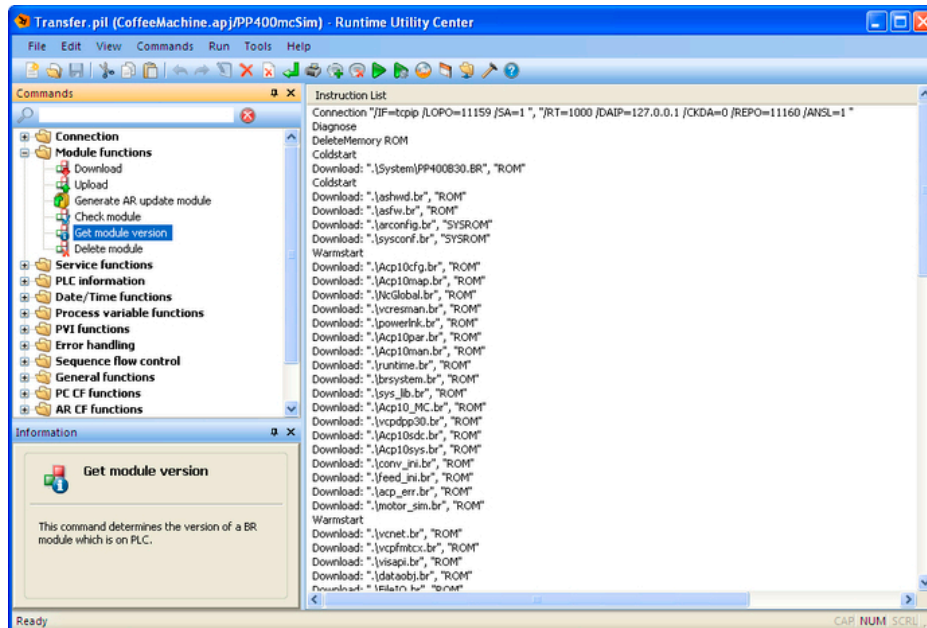
Querying the battery status in the I/O mapping window

6.3 Diagnostics Tool Runtime Utility Center

The **Runtime Utility Center** program is not just used to copy an executable application to CompactFlash from Automation Studio.

The Runtime Utility Center also includes functions useful for commissioning, maintenance, diagnostics and servicing.

The Runtime Utility Center started in Automation Studio with the **<Tools> / <Runtime Utility Center>** menu item.



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 \ Diagnostics Tool \ 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.

Back up variable values

Situation:

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 PVI Transfer list that does the following:

- Backs up the variable values in the "Loop" task

Procedure:

- Open Runtime Utility Center from Automation Studio.
- Create a new list using the **<File> / <New>** menu item.
- Insert the command for establishing a connection using the **<Command> / <Connection>**



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

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

- Insert the command for loading the variable list using the **<Command> / <Processvariable-Functions> / <Variable list>**



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

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

- Execute the functions with **<F5>**.



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



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

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 PVI Transfer list that does the following:

- Restores the variable values
- 1) Open the Runtime Utility Center from Automation Studio. Create a new list using the **<File> / <New>** menu item
 - 2) Insert the command for establishing a connection using the **<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 using the **<Command> / <Processvariable-Functions>**

tions> / <Write variable list to PLC>

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

- 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 undesired behavior in the process sequence may result if considerable care is not taken when the variables are rewritten.

If this situation becomes necessary, we recommend switching the controller to diagnostic mode first (CPU stop).

6.3.2 Passing on the project with the 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 online connection to the CPU.

Reinstall the controller

The controller should be reinstalled without using Automation Studio. Start the Runtime Utility Center in Automation Studio after the project has been built. In the Runtime Utility Center, the CD generation process is started with the <Tools> / <Create installation package> menu item.

Requirements:

- The controller must be outfitted with a CompactFlash card that includes an executable version of Automation Runtime.
- The IP address of the CPU must be known.

Step 1:

- Compile the project in Automation Studio.
- Start the Runtime Utility Center from Automation Studio using the <Tools> / <Runtime Utility Center>
- Generate an executable Runtime Utility Center installation image using the <Tools> / <Create installation package>
- After the target directory is specified, the generation process begins by pressing the <Start> button.



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



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

The Runtime Utility Center can be closed once the creation process has been completed

Step 2:

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

To pass on the Runtime Utility Center installation to others, the contents of the target directory specified during the CD creation process can be burned to CD or copied to a flash drive. .

The "**Start.bat**" file simply needs to be run on the PC at the new system.



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

7 SUMMARY

There are several different tools available to localize problems and errors.

They need to be used sensibly in combination with analytical thinking.



Diagnostics

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.

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.

TRAINING MODULES

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

