SoMachine

Manage a Cyclic Task Interval
Toolbox_Advance Library Guide

04/2014

EIO0000000946.01

www.schneider-electric.com

Schneider Electric

The information provided in this documentation contains general descriptions and/or technical characteristics of the performance of the products contained herein. This documentation is not intended as a substitute for and is not to be used for determining suitability or reliability of these products for specific user applications. It is the duty of any such user or integrator to perform the appropriate and complete risk analysis, evaluation and testing of the products with respect to the relevant specific application or use thereof. Neither Schneider Electric nor any of its affiliates or subsidiaries shall be responsible or liable for misuse of the information contained herein. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us.

No part of this document may be reproduced in any form or by any means, electronic or mechanical, including photocopying, without express written permission of Schneider Electric.

All pertinent state, regional, and local safety regulations must be observed when installing and using this product. For reasons of safety and to help ensure compliance with documented system data, only the manufacturer should perform repairs to components.

When devices are used for applications with technical safety requirements, the relevant instructions must be followed.

Failure to use Schneider Electric software or approved software with our hardware products may result in injury, harm, or improper operating results.

Failure to observe this information can result in injury or equipment damage.

© 2014 Schneider Electric. All rights reserved.

Table of Contents

Safety Information

Important Information

NOTICE

Read these instructions carefully, and look at the equipment to become familiar with the device before trying to install, operate, or maintain it. The following special messages may appear throughout this documentation or on the equipment to warn of potential hazards or to call attention to information that clarifies or simplifies a procedure.

The addition of this symbol to a Danger safety label indicates that an electrical hazard exists, which will result in personal injury if the instructions are not followed.

This is the safety alert symbol. It is used to alert you to potential personal injury hazards. Obey all safety messages that follow this symbol to avoid possible injury or death.

⚠ DANGER

DANGER indicates an imminently hazardous situation which, if not avoided, **will result in** death or serious injury.

⚠ WARNING

WARNING indicates a potentially hazardous situation which, if not avoided, **can result in** death or serious injury.

⚠ CAUTION

CAUTION indicates a potentially hazardous situation which, if not avoided, **can result in** minor or moderate injury.

NOTICE

NOTICE is used to address practices not related to physical injury.

PLEASE NOTE

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel. No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this material.

A qualified person is one who has skills and knowledge related to the construction and operation of electrical equipment and its installation, and has received safety training to recognize and avoid the hazards involved.

BEFORE YOU BEGIN

Do not use this product on machinery lacking effective point-of-operation guarding. Lack of effective point-of-operation guarding on a machine can result in serious injury to the operator of that machine.

⚠ WARNING

UNGUARDED MACHINERY CAN CAUSE SERIOUS INJURY

- Do not use this software and related automation equipment on equipment which does not have point-of-operation protection.
- Do not reach into machinery during operation.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

This automation equipment and related software is used to control a variety of industrial processes. The type or model of automation equipment suitable for each application will vary depending on factors such as the control function required, degree of protection required, production methods, unusual conditions, government regulations, etc. In some applications, more than one processor may be required, as when backup redundancy is needed.

Only the user can be aware of all the conditions and factors present during setup, operation, and maintenance of the machine; therefore, only the user can determine the automation equipment and the related safeties and interlocks which can be properly used. When selecting automation and control equipment and related software for a particular application, the user should refer to the applicable local and national standards and regulations. The National Safety Council’s Accident Prevention Manual (nationally recognized in the United States of America) also provides much useful information.

In some applications, such as packaging machinery, additional operator protection such as point-of-operation guarding must be provided. This is necessary if the operator’s hands and other parts of the body are free to enter the pinch points or other hazardous areas and serious injury can occur. Software products alone cannot protect an operator from injury. For this reason the software cannot be substituted for or take the place of point-of-operation protection.

Ensure that appropriate safeties and mechanical/electrical interlocks related to point-of-operation protection have been installed and are operational before placing the equipment into service. All interlocks and safeties related to point-of-operation protection must be coordinated with the related automation equipment and software programming.

NOTE: Coordination of safeties and mechanical/electrical interlocks for point-of-operation protection is outside the scope of the Function Block Library, System User Guide, or other implementation referenced in this documentation.

START-UP AND TEST

Before using electrical control and automation equipment for regular operation after installation, the system should be given a start-up test by qualified personnel to verify correct operation of the equipment. It is important that arrangements for such a check be made and that enough time is allowed to perform complete and satisfactory testing.

⚠ CAUTION

EQUIPMENT OPERATION HAZARD

- Verify that all installation and set up procedures have been completed.
- Before operational tests are performed, remove all blocks or other temporary holding means used for shipment from all component devices.
- Remove tools, meters, and debris from equipment.

Failure to follow these instructions can result in injury or equipment damage.

Follow all start-up tests recommended in the equipment documentation. Store all equipment documentation for future references.

Software testing must be done in both simulated and real environments.

Verify that the completed system is free from all short circuits and grounds that are not installed according to local regulations (according to the National Electrical Code in the U.S.A, for instance). If high-potential voltage testing is necessary, follow recommendations in equipment documentation to prevent accidental equipment damage.

Before energizing equipment:
- Remove tools, meters, and debris from equipment.
- Close the equipment enclosure door.
- Remove all temporary grounds from incoming power lines.
- Perform all start-up tests recommended by the manufacturer.

OPERATION AND ADJUSTMENTS

The following precautions are from the NEMA Standards Publication ICS 7.1-1995 (English version prevails):

- Regardless of the care exercised in the design and manufacture of equipment or in the selection and ratings of components, there are hazards that can be encountered if such equipment is improperly operated.
- It is sometimes possible to misadjust the equipment and thus produce unsatisfactory or unsafe operation. Always use the manufacturer’s instructions as a guide for functional adjustments. Personnel who have access to these adjustments should be familiar with the equipment manufacturer’s instructions and the machinery used with the electrical equipment.
- Only those operational adjustments actually required by the operator should be accessible to the operator. Access to other controls should be restricted to prevent unauthorized changes in operating characteristics.

About the Book

At a Glance

Document Scope

This documentation describes the Toolbox_Advance library functions used to get and set the interval of a Cyclic Task.

Validity Note

This document has been updated with the release of SoMachine V4.0.

Related Documents

Title of the Document	Reference Number
Modicon M238 Logic Controller Programming Guide	EIO0000000384 (FRE) EIO0000000385 (FRE) EIO0000000386 (GER) EIO0000000388 (SPA) EIO0000000387 (ITA) EIO0000000389 (CHS)
Modicon M258 Logic Controller Programming Guide	EIO0000000402 (ENG) EIO0000000403 (FRE) EIO0000000404 (GER) EIO0000000405 (SPA) EIO0000000406 (ITA) EIO0000000407 (CHS)
Modicon LMC058 Logic Controller Programming Guide	EIO0000000408 (ENG) EIO0000000409 (FRE) EIO0000000410 (GER) EIO0000000411 (SPA) EIO0000000412 (ITA) EIO0000000413 (CHS)
Magelis XBTGC HMI Controller Programming Guide	EIO0000000650 (ENG) EIO0000000651 (FRE) EIO0000000652 (GER) EIO0000000653 (SPA) EIO0000000654 (ITA) EIO0000000655 (CHS)

Title of the Document	Reference Number
Magelis XBTGT, XBTGK HMI Controller Programming Guide	EIO0000000638 (ENG) EIO0000000639 (FRE) EIO0000000640 (GER) EIO0000000641 (SPA) EIO0000000642 (ITA) EIO0000000643 (CHS)
ATV IMC Drive Controller Programming Guide	EIO0000000390 (ENG) EIO0000000391 (FRE) EIO0000000392 (GER) EIO0000000393 (SPA) EIO0000000394 (ITA) EIO0000000395 (CHS)

Product Related Information

WARNING

LOSS OF CONTROL

- The designer of any control scheme must consider the potential failure modes of control paths and, for certain critical control functions, provide a means to achieve a safe state during and after a path failure. Examples of critical control functions are emergency stop and overtravel stop, power outage and restart.
- Separate or redundant control paths must be provided for critical control functions.
- System control paths may include communication links. Consideration must be given to the implications of unanticipated transmission delays or failures of the link.
- Observe all accident prevention regulations and local safety guidelines.[1]
- Each implementation of this equipment must be individually and thoroughly tested for proper operation before being placed into service.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

[1] For additional information, refer to NEMA ICS 1.1 (latest edition), "Safety Guidelines for the Application, Installation, and Maintenance of Solid State Control" and to NEMA ICS 7.1 (latest edition), "Safety Standards for Construction and Guide for Selection, Installation and Operation of Adjustable-Speed Drive Systems" or their equivalent governing your particular location.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- Only use software approved by Schneider Electric for use with this equipment.
- Update your application program every time you change the physical hardware configuration.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Chapter 1
Presentation

Presentation

Overview

The Toolbox_Advance library offers 2 functions that can be used in a Program Organisation Unit (POU) to get (`GetCurrentTaskCycle`) and set (`SetCurrentTaskCycle`) the interval of the attached Cyclic Task.

NOTE: These functions can be used in POUs attached to Cyclic Tasks only. For further information about the Tasks configuration and POUs attachment, refer to the Tasks chapter of your Controller Programming Guide.

Adding the Toolbox_Advance Library

To have access to the Toolbox_Advance functions, it is necessary to manually add the library:

Step	Action
1	Double-click the **Library Manager** node of the controller **Application** node in the **Devices** window.
2	Click **Add Library**. Result: The **Add Library** dialog box opens.
3	Browse to **Util**, visible when the **Company** filter is set to **Schneider Electric** or **All Company**. Result: The **Util** libraries list is displayed.
4	Select the **Toolbox_Advance library** and press **OK**. Result: The library is added to the **Library Manager** list.

Chapter 2
Functions Descriptions

Overview

This chapter describes the functions of the Toolbox_Advance library.

What Is in This Chapter?

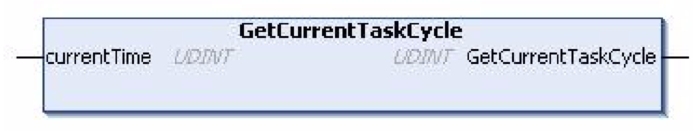
This chapter contains the following topics:

GetCurrentTaskCycle: Gets the Current Cyclic Task Interval

Function Description

This function returns an operation diagnostic and the current interval of the task attached Cyclic Task in microseconds (µs).

Graphical Representation



IL and ST Representation

To see the general representation in IL or ST language, refer to the chapter Function and Function Block Representation *(see page 23)*.

I/O Variables Description

The following table describes the output variable:

Output	**Type**	**Description**
`GetCurrentTaskCycle`	`UDINT`	Function operation diagnostic: 0 = No error detected or Non Cyclic Task 2 = Invalid parameter detected 12 = Function not implemented in the controller

The following table describes the input/output variable:

Input/Output	**Type**	**Description**
`currentTime`	`UDINT`	Current interval of the POU attached Task in microseconds (µs). **NOTE:** Returns 0 if an error has been detected.

NOTE: If this function is called in POU attached to a non Cyclic Task, the function returns 0 and `currentTime` is set to 0.

Example

The following example describes how to get the current interval of the Cyclic Task attached to a POU `GetTaskInterval` in ST language. If a function operation error is detected, a diagnostic flag is set to `TRUE` and the error code is stored.

If no error detected, the diagnostic flag is set to `FALSE` and the valid task interval is stored.

Variable declaration:

```
PROGRAM GetTaskInterval
VAR
   // Last Task interval
   TaskInterval: UDINT := 0;
   // Last valid Task interval
   TaskInterval_Memo: UDINT := 0;
   // GetCurrentTaskCycle function operation diagnostic
   GetTaskCycle_Diag: UDINT := 0;
   // Memorisation of last GetCurrentTaskCycle detected error
   code
   GetTaskCycle_Diag_Memo: UDINT := 0;
   // GetCurrentTaskCycle error detected flag
   GetTaskCycle_Err: BOOL := FALSE;
END_VAR
```

Program:

```
// Get the last Cyclic Task interval
GetTaskCycle_Diag:=GetCurrentTaskCycle(TaskInterval);
// Check diagnostic
IF TaskInterval=0 OR GetTaskCycle_Diag<>0
   THEN // Error detected
   GetTaskCycle_Diag_Memo:=GetTaskCycle_Diag;
   GetTaskCycle_Err:=TRUE;
   ELSE // Valid Task interval
   TaskInterval_Memo:=TaskInterval;
   GetTaskCycle_Err:=FALSE;
END_IF;
```

SetCurrentTaskCycle: Sets the Cyclic Task Interval

Function Description

This function sets the Cyclic Task interval with the input parameter value in microseconds (µs) and returns an operation diagnostic. If the operation has been executed successfully, the new interval is valid at the start of the next iteration of the Cyclic Task (no effect on the current execution).

NOTE: The value passed to the function must be a multiple of a thousand. If not, an error code will be returned.

If you define an interval for a Cyclic Task that is too short (shorter than the effective duration of the task and any other processing required by the application), it will repeat immediately after the write of the outputs without executing other lower priority tasks or any system processing. This will affect the execution of all tasks and cause the controller to exceed the system watchdog limits, generating a system watchdog exception and stopping your controller.

NOTICE

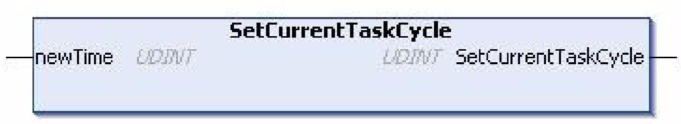
UNINTENDED INTERRUPTION OF PROGRAM EXECUTION

Test your new Cyclic Task interval time to validate that is appropriate to prevent a system watchdog exception before calling the `SetCurrentTaskCycle` function.

Failure to follow these instructions can result in equipment damage.

NOTE: You must test the possible range of values prior to commissioning your application, and be sure your code validates the new interval to those test conditions.

Graphical Representation



IL and ST Representation

To see the general representation in IL or ST language, refer to the chapter Function and Function Block Representation *(see page 23)*.

I/O Variables Description

The following table describes the input variable:

Input	Type	Description
`newTime`	`UDINT`	New value of the POU attached Task interval in microseconds (µs). **NOTE:** If valid, the new value is operational at next cycle.

The following table describes the output variable:

Output	Type	Description
`SetCurrentTaskCycle`	`UDINT`	Function operation diagnostic: 0 = No error detected 1 = General (internal) error detected 2 = Invalid parameter detected (out of range value) 12 = Function not implemented in the controller 24 = Functionality not supported (for example non Cyclic Task)

NOTE: This function changes the current Cyclic Task interval. It does not override the Cyclic Task interval parameter value set in the task configuration. Initial configuration parameters are restored on Reset, Reboot or Download command.

Refer to your controller programming guide for more information about your controller states and behaviors.

NOTE: Do not use the SetCurrentTaskCycle function in the same task defined as the CANopen managers bus cycle task. Changing the cycle time of this task will affect the heartbeat or the nodeguarding, possibly causing the CANopen devices to detect an error and pass into a fallback state.

⚠ CAUTION

UNINTENDED INTERRUPTION OF PROGRAM EXECUTION

Do not use the `SetCurrentTaskCycle` function in any task defined as a bus cycle task.

Failure to follow these instructions can result in injury or equipment damage.

Example

The following example describes how to set the current interval of the Cyclic Task attached to a POU `SetTaskInterval` in CFC, FBD or LD language.

The function `SetCurrentTaskCycle` is executed when the flag `SetTaskInterval_Flag` is `TRUE` (this flag is automatically reset to `FALSE` after execution) and the function operation diagnostic is stored in the `SetTaskInterval_Diag` variable.

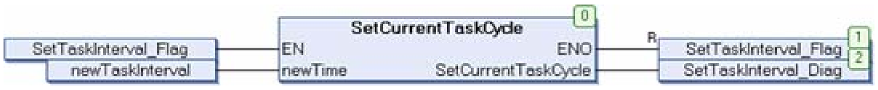
NOTE:

The function enabling input/output `(EN/ENO)` is activated:
- In CFC: right click on the function and select the **EN/ENO** command.
- In FBD: add the function with the command **Insert Box with EN/ENO** of the **FBD/LD/IL** menu.
- In LD: switch to FBD view with the **View** command of the **FBD/LD/IL** menu, insert the function as described above and switch back to LD view.

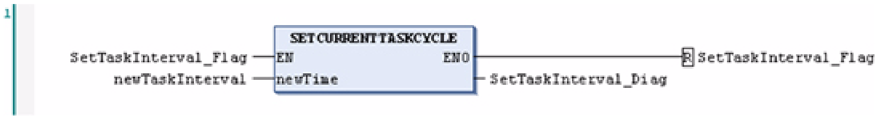
Variables declaration:

```
PROGRAM SetTaskInterval
VAR
   // Set a new Task Interval command flag
   SetTaskInterval_Flag: BOOL := FALSE;
   // Value of the new Cyclic Task interval value (µs)
   newTaskInterval: UDINT;
   //SetCurrentTaskCycle function operation diagnostic
   SetTaskInterval_Diag: UDINT := 0;
END_VAR
```

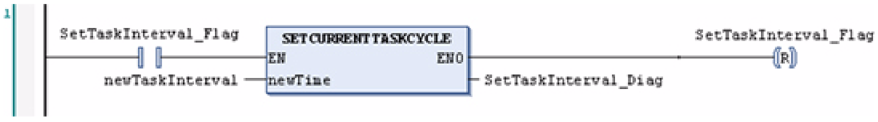
Program in CFC language:



Program in FBD language:



Program in LD language:



Appendices

Appendix A
Function and Function Block Representation

Overview

Each function can be represented in the following languages:
- IL: Instruction List
- ST: Structured Text
- LD: Ladder Diagram
- FBD: Function Block Diagram
- CFC: Continuous Function Chart

This chapter provides functions and function blocks representation examples and explains how to use them for IL and ST languages.

What Is in This Chapter?

This chapter contains the following topics:

Differences Between a Function and a Function Block

Function

A function:
- is a POU (Program Organization Unit) that returns one immediate result.
- is directly called with its name (not through an instance).
- has no persistent state from one call to the other.
- can be used as an operand in other expressions.

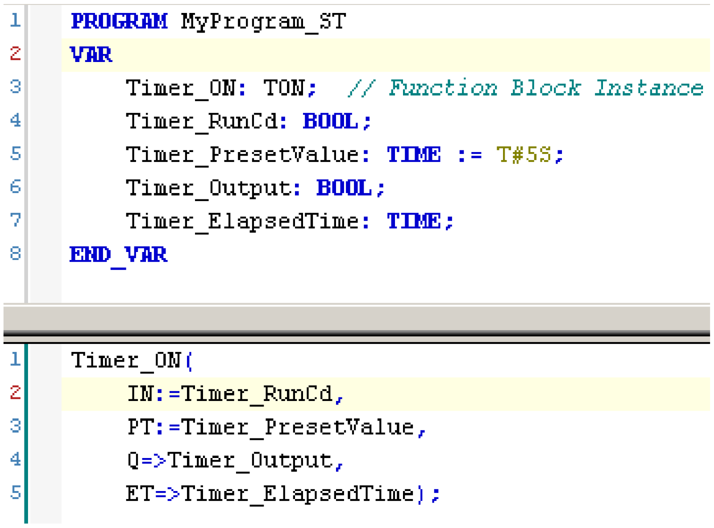
Examples: boolean operators (`AND`), calculations, conversion (`BYTE_TO_INT`)

Function Block

A function block:
- is a POU (Program Organization Unit) that returns one or more outputs.
- needs to be called by an instance (function block copy with dedicated name and variables).
- each instance has a persistent state (outputs and internal variables) from one call to the other from a function block or a program.

Examples: timers, counters

In the example, `Timer_ON` is an instance of the function block `TON`:



```
PROGRAM MyProgram_ST
VAR
    Timer_ON: TON;   // Function Block Instance
    Timer_RunCd: BOOL;
    Timer_PresetValue: TIME := T#5S;
    Timer_Output: BOOL;
    Timer_ElapsedTime: TIME;
END_VAR
```

```
Timer_ON(
    IN:=Timer_RunCd,
    PT:=Timer_PresetValue,
    Q=>Timer_Output,
    ET=>Timer_ElapsedTime);
```

How to Use a Function or a Function Block in IL Language

General Information

This part explains how to implement a function and a function block in IL language.

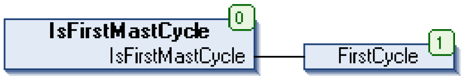
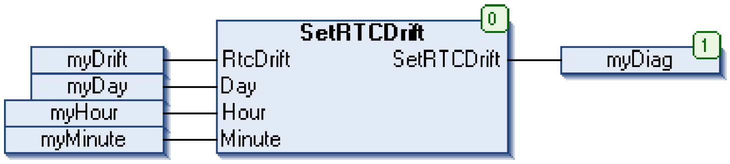
Functions `IsFirstMastCycle` and `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

Using a Function in IL Language

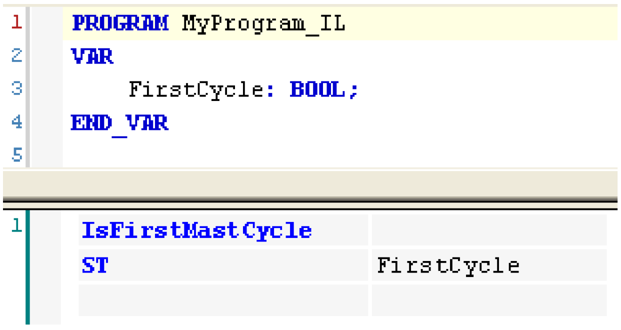
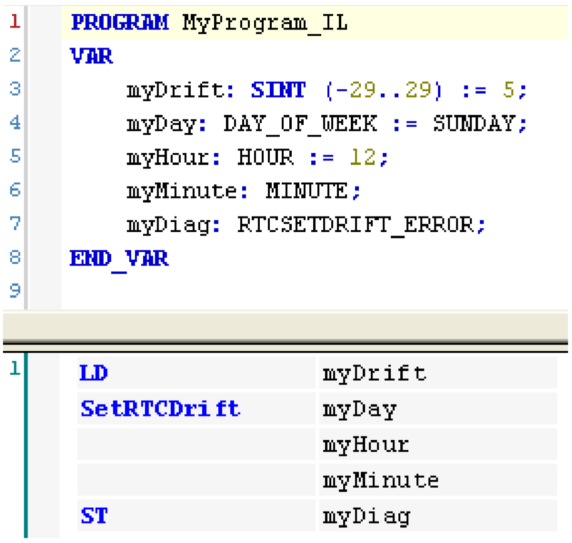
This procedure describes how to insert a function in IL language:

Step	Action
1	Open or create a new POU in Instruction List language. **NOTE:** The procedure to create a POU is not detailed here. For more information refer to Adding, Declaring an Calling POUs *(see SoMachine, Programming Guide)*.
2	Create the variables that the function requires.
3	If the function has 1 or more inputs, start loading the first input using LD instruction.
4	Insert a new line below and: ● type the name of the function in the operator column (left field), or ● use the **Input Assistant** to select the function (select **Insert Box** in the context menu).
5	If the function has more than 1 input and when Input Assistant is used, the necessary number of lines is automatically created with `???` in the fields on the right. Replace the `???` with the appropriate value or variable that corresponds to the order of inputs.
6	Insert a new line to store the result of the function into the appropriate variable: type ST instruction in the operator column (left field) and the variable name in the field on the right.

To illustrate the procedure, consider the Functions `IsFirstMastCycle` (without input parameter) and `SetRTCDrift` (with input parameters) graphically presented below:

Function	Graphical Representation
without input parameter: `IsFirstMastCycle`	 0 IsFirstMastCycle IsFirstMastCycle FirstCycle 1
with input parameters: `SetRTCDrift`	 0 SetRTCDrift myDrift myDay myHour myMinute RtcDrift Day Hour Minute SetRTCDrift myDiag 1

In IL language, the function name is used directly in the operator column:

Function	Representation in SoMachine POU IL Editor
IL example of a function without input parameter: `IsFirstMastCycle`	 `PROGRAM MyProgram_IL` `VAR` `FirstCycle: BOOL;` `END_VAR` — `IsFirstMastCycle` / `ST FirstCycle`
IL example of a function with input parameters: `SetRTCDrift`	 `PROGRAM MyProgram_IL` `VAR` `myDrift: SINT (-29..29) := 5;` `myDay: DAY_OF_WEEK := SUNDAY;` `myHour: HOUR := 12;` `myMinute: MINUTE;` `myDiag: RTCSETDRIFT_ERROR;` `END_VAR` — `LD myDrift` / `SetRTCDrift myDay` / `myHour` / `myMinute` / `ST myDiag`

```
PROGRAM MyProgram_IL
VAR
    FirstCycle: BOOL;
END_VAR

IsFirstMastCycle
ST                  FirstCycle
```

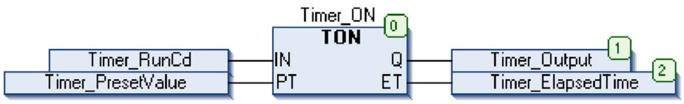
```
PROGRAM MyProgram_IL
VAR
    myDrift: SINT (-29..29) := 5;
    myDay: DAY_OF_WEEK := SUNDAY;
    myHour: HOUR := 12;
    myMinute: MINUTE;
    myDiag: RTCSETDRIFT_ERROR;
END_VAR

LD              myDrift
SetRTCDrift     myDay
                myHour
                myMinute
ST              myDiag
```

Using a Function Block in IL Language

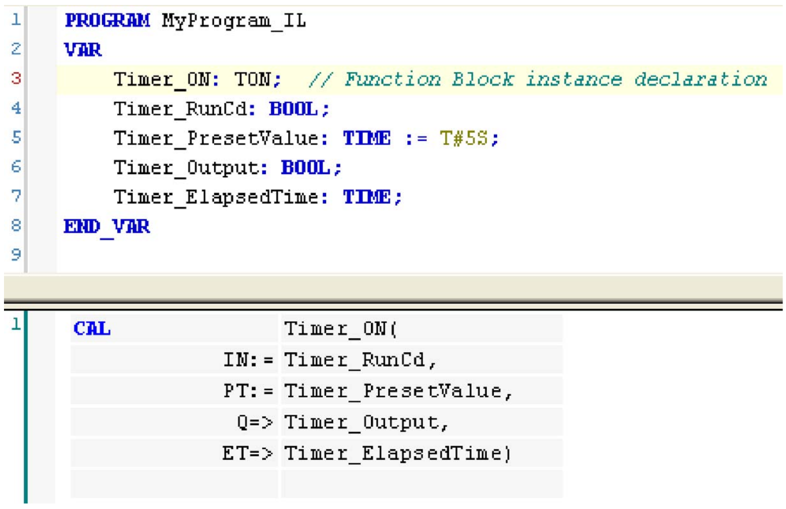
This procedure describes how to insert a function block in IL language:

Step	Action
1	Open or create a new POU in Instruction List language. **NOTE:** The procedure to create a POU is not detailed here. For more information refer to Adding, Declaring an Calling POUs *(see SoMachine, Programming Guide)*.
2	Create the variables that the function block requires, including the instance name.
3	Function Blocks are called using a `CAL` instruction: ● Use the **Input Assistant** to select the FB (right-click and select **Insert Box** in the context menu). ● Automatically, the `CAL` instruction and the necessary I/O are created. Each parameter (I/O) is an instruction: ● Values to inputs are set by "`:=`". ● Values to outputs are set by "`=>`".
4	In the `CAL` right-side field, replace `???` with the instance name.
5	Replace other `???` with an appropriate variable or immediate value.

To illustrate the procedure, consider this example with the `TON` Function Block graphically presented below:

Function Block	Graphical Representation
`TON`	 Timer_ON 0 TON Timer_RunCd Timer_PresetValue IN PT Q ET Timer_Output 1 Timer_ElapsedTime 2

In IL language, the function block name is used directly in the operator column:

Function Block	Representation in SoMachine POU IL Editor
TON	

```
PROGRAM MyProgram_IL
VAR
    Timer_ON: TON;   // Function Block instance declaration
    Timer_RunCd: BOOL;
    Timer_PresetValue: TIME := T#5S;
    Timer_Output: BOOL;
    Timer_ElapsedTime: TIME;
END_VAR
```

```
CAL          Timer_ON(
        IN:= Timer_RunCd,
        PT:= Timer_PresetValue,
         Q=> Timer_Output,
        ET=> Timer_ElapsedTime)
```

How to Use a Function or a Function Block in ST Language

General Information

This part explains how to implement a Function and a Function Block in ST language.

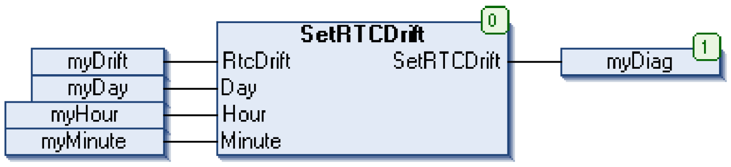
Function `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

Using a Function in ST Language

This procedure describes how to insert a function in ST language:

Step	Action
1	Open or create a new POU in Structured Text language. **NOTE:** The procedure to create a POU is not detailed here. For more information refer to Adding, Declaring and Calling POUs *(see SoMachine, Programming Guide)*.
2	Create the variables that the function requires.
3	Use the general syntax in the **POU ST Editor** for the ST language of a function. The general syntax is: `FunctionResult:= FunctionName(VarInput1, VarInput2,.. VarInputx);`

To illustrate the procedure, consider the function `SetRTCDrift` graphically presented below:

Function	Graphical Representation
SetRTCDrift	 SetRTCDrift 0 myDrift myDay myHour myMinute RtcDrift Day Hour Minute SetRTCDrift myDiag 1

The ST language of this function is the following:

Function	Representation in SoMachine POU ST Editor

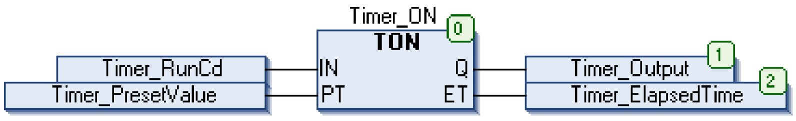
SetRTCDrift	`PROGRAM MyProgram_ST`
	`VAR myDrift: SINT(-29..29) := 5;`
	`myDay: DAY_OF_WEEK := SUNDAY;`
	`myHour: HOUR := 12;`
	`myMinute: MINUTE;`
	`myRTCAdjust: RTCDRIFT_ERROR;`
	`END_VAR`
	`myRTCAdjust:= SetRTCDrift(myDrift, myDay, myHour, myMinute);`

Using a Function Block in ST Language

This procedure describes how to insert a function block in ST language:

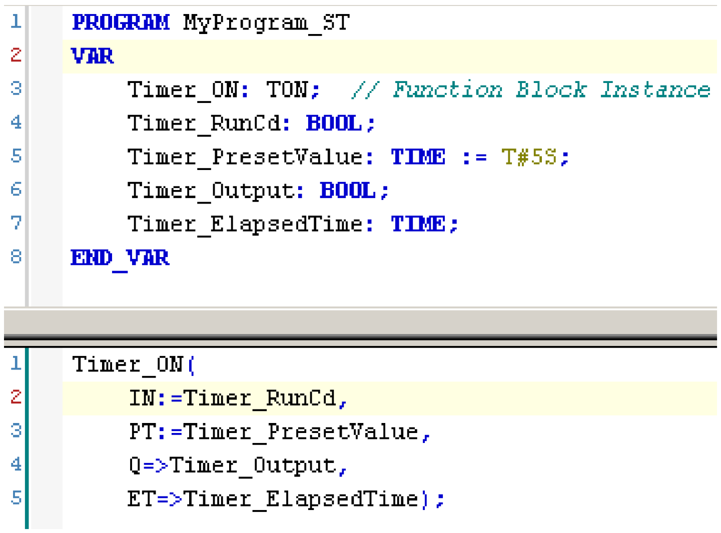
Step	Action
1	Open or create a new POU in Structured Text language. **NOTE:** The procedure to create a POU is not detailed here. For more information on adding, declaring and calling POUs, refer to the related documentation *(see SoMachine, Programming Guide)*.
2	Create the input and output variables and the instance required for the function block: ● Input variables are the input parameters required by the function block ● Output variables receive the value returned by the function block
3	Use the general syntax in the **POU ST Editor** for the ST language of a Function Block. The general syntax is: `FunctionBlock_InstanceName(Input1:=VarInput1, Input2:=VarInput2,... Ouput1=>VarOutput1, Ouput2=>VarOutput2,...);`

To illustrate the procedure, consider this example with the `TON` function block graphically presented below:

Function Block	Graphical Representation
`TON`	 Timer_ON 0 TON Timer_RunCd Timer_PresetValue IN PT Q ET 1 Timer_Output 2 Timer_ElapsedTime

This table shows examples of a function block call in ST language:

Function Block	Representation in SoMachine POU ST Editor

TON	
	```
	PROGRAM MyProgram_ST
	VAR
	    Timer_ON: TON;  // Function Block Instance
	    Timer_RunCd: BOOL;
	    Timer_PresetValue: TIME := T#5S;
	    Timer_Output: BOOL;
	    Timer_ElapsedTime: TIME;
	END_VAR

	Timer_ON(
	    IN:=Timer_RunCd,
	    PT:=Timer_PresetValue,
	    Q=>Timer_Output,
	    ET=>Timer_ElapsedTime);
	```

Glossary

B

byte

A type that is encoded in an 8-bit format, ranging from `16#00` to `16#FF` in hexadecimal representation.

C

CFC

(*continuous function chart*) A graphical programming language (an extension of the IEC 61131-3 standard) based on the function block diagram language that works like a flowchart. However, no networks are used and free positioning of graphic elements is possible, which allows feedback loops. For each block, the inputs are on the left and the outputs on the right. You can link the block outputs to the inputs of other blocks to create complex expressions.

F

FB

(*function block*) A convenient programming mechanism that consolidates a group of programming instructions to perform a specific and normalized action, such as speed control, interval control, or counting. A function block may comprise configuration data, a set of internal or external operating parameters and usually 1 or more data inputs and outputs.

FBD

(*function block diagram*) One of the 5 languages for logic or control supported by the standard IEC 61131-3 for control systems. Function block diagram is a graphically oriented programming language. It works with a list of networks where each network contains a graphical structure of boxes and connection lines representing either a logical or arithmetic expression, the call of a function block, a jump, or a return instruction.

I

IL

(*instruction list*) A program written in the language that is composed of a series of text-based instructions executed sequentially by the controller. Each instruction includes a line number, an instruction code, and an operand (refer to IEC 61131-3).

INT

(*integer*) A whole number encoded in 16 bits.

L

LD

(*ladder diagram*) A graphical representation of the instructions of a controller program with symbols for contacts, coils, and blocks in a series of rungs executed sequentially by a controller (refer to IEC 61131-3).

P

POU

(*program organization unit*) A variable declaration in source code and a corresponding instruction set. POUs facilitate the modular re-use of software programs, functions, and function blocks. Once declared, POUs are available to one another.

S

ST

(*structured text*) A language that includes complex statements and nested instructions (such as iteration loops, conditional executions, or functions). ST is compliant with IEC 61131-3.

V

variable

A memory unit that is addressed and modified by a program.

Index