| Situation | IL | Ladder | Rung reversible |
|---|---|---|---|
| Instruction preceding *MRD* or *MPP* not either a conditional action or associated with stack instructions | Advisory | - | No |
| Unnested *OR* between *MPS* and *MPP* | Advisory | - | No |
| *OR* after an action instruction | Advisory | - | No |
| *OR* after *MPS*, *MRD* or *MPP* | Advisory | - | No |
| Subroutine call or *JMPC* not the last action instruction of the rung | Advisory | Error | No |
| Canonic rung exceeds 7x11 cells in Twido, 256 x 30 cells in EcoStruxure Machine Expert - Basic | Advisory | - | No |
| Unconditional action instruction between *BLK* and *END_BLK* | Error | - | No |
| *OUT_BLK* not followed by *LD* of a valid FB output or *END_BLK* | Error | - | No |
| FB cannot occupy first cell | - | - | Yes |
| FB on top of the rung, it replaces items occupying the cells | - | - | Yes |
| No logic above or below a FB | - | Error | No |
| *XOR* in first column | - | Error | No |
| Contacts and horizontal connectors in last column | - | Error | No |
| Down connectors in last row or last column | - | Error | No |
| Allow only valid subroutines 0 to 63 | - | Error | No |
| Allow only valid labels 0 to 63 | - | Error | No |
| Invalid operate expressions in operation block | - | Error | No |
| Invalid comparison expressions in comparison block | - | Error | No |
| Invalid address or symbol in contact and coil | - | Error | No |
| Invalid operand or expression with Ladder instruction | - | Error | No |
| Rung with no output action item | - | Error | No |
| Discontinuity between left and right power bars | - | Error | No |
| Dangling Ladder rung | - | Error | No |
| Ladder rung contains items that are short-circuited using connectors | - | Error | No |
| All divergences that contain only boolean logic items must converge in reverse order | - | Error | No |
| FB has no input associated | - | Error | No |
| FB output pins cannot be connected together | - | Error | No |
| *XOR* connected to power bar | - | Error | No |
| Subroutine call and jump not the last output action item | Advisory | Error | No |
| Canonic rung that contains a FB with part of the FB in the last column | - | - | No |
| Canonic rung exceeds 7x11 cells in Twido, 256 x 30 cells in EcoStruxure Machine Expert - Basic | Advisory | Error | No |
| *OPEN* and *SHORT* connected to the left node of subnetwork | - | Error | No |
| *XOR* connected to the left node of subnetwork | - | Error | No |
| There is not at least one existing *LIST* sentence that can represent the ladder rung | - | Error | No |

# Configuring Program Behavior and Tasks

## Application Behavior

### Overview

You can configure the following aspects of how the application interacts with the logic controller:

- **Startup**, page 52
- **Watchdog**, page 53
- **Fallback behavior**, page 54
- **Functional levels**, page 54
- **Options**, page 56

### Configuring Application Behavior

Follow these steps to configure the application behavior:

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Select the **Behavior** item.<br><br>**Result**: The **Behavior** properties appear in the lower central area of the **Programming** window. |
| 3 | Modify the properties as required. |
| 4 | Click **Apply** to save the changes. |

### Startup

Specify how the program behaves following a restart of the logic controller:

- **Start In Previous State**: The logic controller starts in the state that it was in before it was stopped.
- **Start In Stop**: The logic controller does not automatically start application execution.
- **Start In Run** (default): The logic controller automatically starts application execution given run criteria, such as the presence and charge of a battery, are met.
- **Unconditional Start In Run**: The logic controller automatically starts application execution even if the controller battery is absent or discharged.

When using the Start In Run feature, the controller will start executing program logic when power is applied to the equipment. It is essential to know in advance how automatic reactivation of the outputs will affect the process or machine being controlled. Configure the Run/Stop input to help control the Start In Run feature. In addition, the Run/Stop input is designed to give local control over remote RUN commands. If the possibility of a remote RUN command after the controller had been stopped locally by EcoStruxure Machine Expert - Basic would have unintended consequences, you must configure and wire the Run/Stop input to help control this situation.

**⚠WARNING**

**UNINTENDED MACHINE START-UP**

- Confirm that the automatic reactivation of the outputs does not produce unintended consequences before using the Start In Run feature.
- Use the Run/Stop input to help control the Start In Run feature and to help prevent the unintentional start-up from a remote location.
- Verify the state of security of your machine or process environment before applying power to the Run/Stop input or before issuing a Run command from a remote location.

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

**⚠WARNING**

**UNINTENDED MACHINE OR PROCESS START-UP**

- Verify the state of security of your machine or process environment before applying power to the Run/Stop input.
- Use the Run/Stop input to help prevent the unintentional start-up from a remote location.

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

When using the Unconditional Start In Run feature, the controller will attempt to start executing program logic when power is applied to the equipment, independent of the reason the controller had previously stopped. This occurs even if there is no charge in the battery, or if the battery is not present. Therefore, the controller will start with all memory values re-initialized to zero or other predetermined default values. It is conceivable that if the controller attempts to restart, for example, after a short power outage, the values in memory at the time of the outage would be lost, and restarting the machine may have unintended consequences as there was no battery to maintain memory values. It is essential to know in advance how an unconditional start will affect the process or machine being controlled. Configure the Run/Stop input to help control the Unconditional Start In Run feature.

**⚠WARNING**

**UNINTENDED MACHINE OPERATION**

- Conduct a thorough risk analysis to determine the effects, under all conditions, of configuring the controller with the Unconditional Start In Run feature.
- Use the Run/Stop input to help avoid an unwanted unconditional restart.

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

## Watchdog

A watchdog is a special timer used so that programs do not overrun their allocated scan time.

The watchdog timer has a default value of 250 ms. Specify the duration of the watchdog scan task. The possible range is 10...500 ms.

## Fallback Behavior

Specify the fallback behavior to use when the logic controller enters the *STOPPED* or an exception state for any reason.

Two fallback behaviors exist:

- Select **Fallback values** to set outputs to the fallback values defined in the configuration properties of embedded logic controller and expansion module outputs. This is the default.

  Refer to the *Programming Guide* of the logic controller or expansion module for information on configuring fallback values for outputs.

  Individual fallback values cannot be defined for configured Status Alarm, PTO, and FREQGEN outputs. The fallback value for these objects is 0 and cannot be modified.
- Select **Maintain values** to keep each output in the state it was when the logic controller entered the *STOPPED* or an exception state. In this mode, the fallback values configured for logic controller and expansion module outputs are ignored and, instead, are set to the last value assumed by the output.

  **Maintain values** behavior is not applied to fast outputs (HSC reflex outputs, PLS, PWM, PTO, and FREQGEN); the fallback value for these objects is 0.

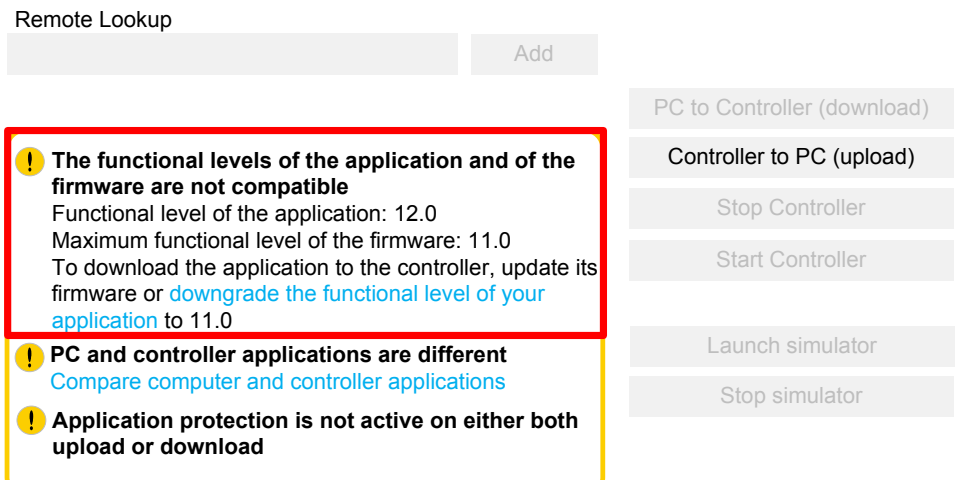
## Functional Levels

Your system could include logic controllers with different firmware versions, and therefore with different capability levels. EcoStruxure Machine Expert - Basic supports functional level management to allow you to control the functional level of your application.

When EcoStruxure Machine Expert - Basic connects to the logic controller, it reads the functional level of the:

- Logic controller firmware to authorize download of the EcoStruxure Machine Expert - Basic application to the logic controller. The functional level selected for the application must be less than or equal to the maximum functional level supported by the logic controller. If this is not the case, a message tells you to either update the firmware, or to manually downgrade the functional level of the application (by selecting a level from the Functional Levels list, see below).
- Application in the logic controller, to determine whether to authorize upload of the logic controller application to the PC running EcoStruxure Machine Expert - Basic. To authorize application upload, the functional level of the logic controller application must be less than or equal to the maximum functional level supported by the installed version of EcoStruxure Machine Expert - Basic. If this is not the case, you must upgrade EcoStruxure Machine Expert - Basic to the latest version before uploading.

Any incompatibilities between the functional levels of the EcoStruxure Machine Expert - Basic application and the application embedded in the connected logic controller are displayed in the **Commissioning** window. For example:




Under **Programming > Behavior**, select a level for the EcoStruxure Machine Expert - Basic application from the **Functional levels** list:

- **Level 14.0**: Authorizes support for User-Defined Function Blocks (UDFB) / User-Defined Functions (UDF) password protection, and UDFB configuration tooltip.
- **Level 13.0**: Authorizes security strategy improvement.
- **Level 12.0**: Authorizes support for a new Dynamic Preset mode in the Timer function block.
- **Level 11.0**: Authorizes configuring the end character in strings, support of the TM3 Bus Coupler TM3BCEIP and TM3BCSL on IO Scanner, filtering and latch on TM3 digital inputs, cybersecurity enhancements, local function block instances in a UDFB, automatic saving and recovering of a project.
- **Level 10.1**: Authorizes application write protection, multiple function blocks in a rung, up to 255 rising/falling functions each, up to 32 instances of *READ_VAR*, *WRITE_VAR* and *WRITE_READ_VAR* function blocks.
- **Level 10.0**: Authorizes indexed memory bits, 200 Grafcet step objects, symbols for variables and parameters in UDFB.
- **Level 6.2**: Authorizes improvement of the password strategy.
- **Level 6.1**: Authorizes strings support in SMS function block.
- **Level 6.0**: Authorizes Modbus TCP IOScanner, user-defined functions, user-defined function blocks, data logging on SD card, string management, structure ladder block elements, rising and falling edge functions.
- **Level 5.1**: Authorizes security strategy modification.
- **Level 5.0**: Authorizes Modbus Serial IOScanner, Drive and RTC function blocks, multi-operand instructions.
- **Level 4.1**: Authorizes online mode enhancements, support for a modem on SL2.
- **Level 4.0**: Authorizes support for sink transistor output controllers, Grafcet (SFC), Frequency Generator, Retentive Timer, Memory Management, Remote Graphic display evolution.
- **Level 3.3**: Authorizes enhancements (PTO Motion Task, HSC evolution).
- **Level 3.2**: Authorizes enhancements to support **Optional module** feature, EtherNet/IP adapter, and *%SEND_RECV_SMS* function block.
- **Level 3.1**: Authorizes enhancements (**Unconditional Start In Run** feature).
- **Level 3.0**: Authorizes enhancements (communications, modem, Remote Graphic Display) to the previous level of software and hardware.

- **Level 2.0**: Authorizes any enhancements and corrections over the previous level software and firmware. For example, for Pulse Train Output (PTO) support, it would be necessary to select this functional level or greater.
- **Level 1.0**: First release of the combination of the EcoStruxure Machine Expert - Basic software and the compatible firmware version(s).

  **NOTE:** Each successive level includes all functionality of the preceding levels.

## Strings

Select the end character used to mark the end of strings:

- **CR (Carriage Return)**. The default value.
- **LF (Line feed)**
- **Null (Null)**

The selected character is automatically added to the end of:

- Strings created using the INT_TO_ASCII (see EcoStruxure Machine Expert - Basic, Generic Functions Library Guide), DINT_TO_ASCII (see EcoStruxure Machine Expert - Basic, Generic Functions Library Guide), and FLOAT_TO_ASCII (see EcoStruxure Machine Expert - Basic, Generic Functions Library Guide) instructions.
- Strings assigned to memory words, page 83. For example, `%MW0:5 := "Hello"` is stored as a sequence of 2 ASCII character codes with the selected end of string character appended as the last character in the string.
- Strings created using the Constant String Assistant window, page 83.

  **NOTE:** The application must be configured with a Functional Level, page 54 of a least **Level 11.0** to support configuring the end of string character. Otherwise, the end of string character is set to **Carriage Return** (CR) and cannot be changed.

# Tasks and Scan Modes

## Overview

EcoStruxure Machine Expert - Basic has the following scan modes for Master task:

- **Normal mode**

  Continuous cyclic scanning mode (Freewheeling mode); a new scan starts immediately after the previous scan has completed.
- **Periodic mode**

  Periodic cyclic scanning mode; a new scan starts only after the configured scan time of the previous scan has elapsed. Every scan is therefore the same duration.

EcoStruxure Machine Expert - Basic offers the following task types:

- **Master task**: Main task of the application.

  Master task is controlled by continuous cyclic scanning (in normal scan mode) or by specifying the scan period of 1...150 ms (default 100 ms) in periodic scan mode.

- **Periodic task**: A short duration subroutine processed periodically.

  Periodic tasks are configured by specifying the scan period of 1...255 ms (default 255 ms).

- **Event task**: A very short duration subroutine to reduce the response time of the application.

  Event tasks are triggered by the physical inputs or the *HSC* function blocks. These events are associated with embedded digital inputs (%I0.2...%I0.5) (rising, falling or both edges) or with the high speed counters (*%HSC0* and *%HSC1*) (when the count reaches the high speed counter threshold). You can configure 2 events for each HSC function block.

## Tasks Priorities

This table summarizes the task types and their priorities:

| Task Type | Scan Mode | Triggering Condition | Configurable Range | Maximum Number of Tasks | Priority |
|---|---|---|---|---|---|
| Master | Normal | Normal | Not applicable | 1 | Lowest |
| | Periodic | Software timer | 1...150 ms[1] | | |
| Periodic | Periodic | Software timer | 1...255 ms[1] | 1 | Higher than master task and lower than event tasks |
| Event | Periodic | Physical inputs | %I0.2...%I0.5 | 4 | Highest |
| | | *%HSC* function blocks | 2 events per % HSC object | | |

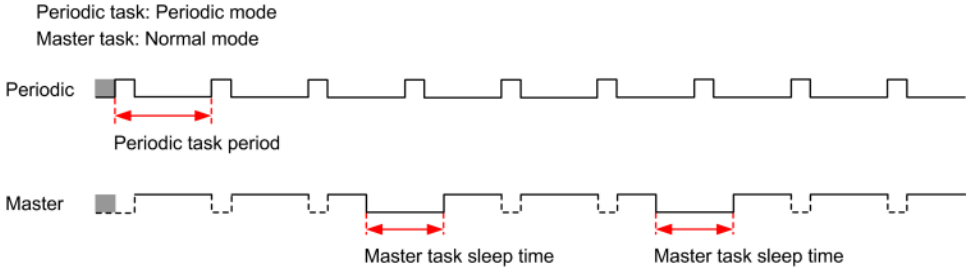
[1] The application must be configured with a functional level, page 54 of at least Level 5.0 to be able to configure a minimum value of 1 ms. Otherwise, the minimum value is 2 ms.

## Events Priorities

Refer to Event Priorities and Queues, page 89.

## Master Task in Normal Scan Mode

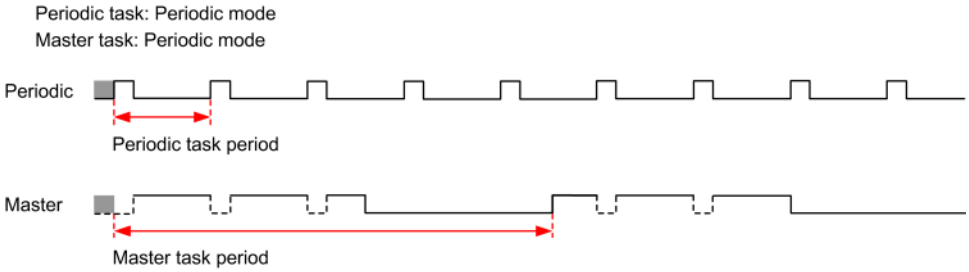
This graphic shows the relationship between master task and periodic task execution when the master task is configured in normal scan mode:




**NOTE:** The master task sleep time is at least 30% of the total cycle time with a minimum of 1 millisecond.

## Master Task in Periodic Scan Mode

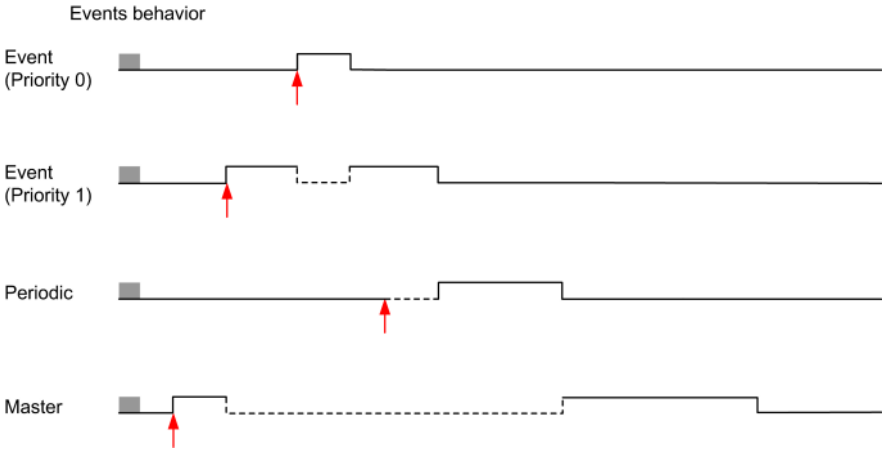
This graphic shows the relationship between master task and periodic task when the master task is configured in periodic scan mode:




## Event Priority Over Master and Periodic Tasks

Event priorities control the relationship between the event tasks, master task, and periodic task. The event task interrupts the master task and periodic task execution.

This figure shows the relationship between event tasks, master tasks, and periodic tasks in the periodic mode:




The event tasks are triggered by a hardware interruption that sends a task event to the event task.

# Managing POUs

## POUs

### Overview

A Program Organization Unit (POU) is a reusable object used in a program. Each POU consists of a variable declaration and a set of instructions in the source code of a supported programming language.

One POU always exists and is linked to the master task of the program. This POU is then called automatically whenever the program starts.

You can create additional POUs containing other objects, for example, functions or function blocks.

When first created, a POU can be either:

- associated with a task, page 59, or
- a Free POU, page 64. A Free POU is not associated with a specific task or event. A Free POU can, for example, contain library functions that are maintained independently of the main program. Free POUs are called from within programs as either subroutines or jumps. A periodic task, page 86 is a subroutine that is implemented as a Free POU.

# Managing POUs with Tasks

## Adding a New POU Associated with a Task

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Add a new POU by one of the following methods:<br>• Right-click the **Master Task** and choose **Add POU** from the contextual menu that appears.<br>• Select the **Master Task** and click  (**Add POU**) on the toolbar at the top of the **Tasks** tab.<br>**Result**: A new POU is added to the program structure immediately below the default/last POU in the **Master Task**. The default name is *n* - **New POU**, where *n* is an integer incremented each time a POU is created. |
| 3 | To reposition a POU in the **Master Task**, select a POU and click the UP or DOWN button on the toolbar at the top of the **Tasks** tab to move the selected POU up or down in the program structure. |

## Inserting a New POU

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Select an existing POU above which to insert the POU. |
| 3 | Right-click the selected POU and choose **Insert POU** from the contextual menu that appears. |
| 4 | To reposition a POU in the **Master Task**, select a POU and click the UP or DOWN button on the toolbar at the top of the **Tasks** tab to move the selected POU up or down in the program structure. |

## Copying Existing POUs Associated with a Task

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Select one or multiple POUs:<br>• Select an existing POU in the **Master Task**.<br>• Press and hold the CTRL key and select each POU in the **Master Task**. |
| 3 | Right-click one of the selected POU in the **Master Task** and choose **Copy POU** from the contextual menu that appears. |
| 4 | Right-click the **Master Task** and choose **Paste POU** from the contextual menu that appears.<br>**Result**: One or multiple POUs are added to the program structure immediately below the selected POU in the **Master Task** with the same name as the copied POU. |

## Exporting of POU or Free POUs

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Select one or multiple existing POUs or Free POUs in the **Master Task**. |
| 3 | Right-click on the selected POUs or Free POUs in the **Master Task** and choose **Export POU** from the contextual menu that appears. |
| 4 | Save the POU files (*.smbf) in the **Export** folder that appears. |

## Importing of POU or Free POUs

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Select one or multiple existing POUs or Free POUs in the **Master Task**. |
| 3 | Right-click on the selected POUs or Free POUs in the **Master Task** and choose **Import POU** from the contextual menu that appears. |
| 4 | Select the POU files (*.smbf) from the folder that appears.<br>**NOTE:** If a maximum number of Free POUs are reached or the file is corrupted (invalid format), an error message appears and the Free POUs are not imported. |

## Renaming a POU

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Edit the POU name by one of the following methods:<br>• Right-click a POU and choose **Rename POU** from the contextual menu that appears.<br>• Double-click a POU.<br>• Select a POU and double-click the POU name in the programming workspace.<br>• Select a POU and press the F2 key. |
| 3 | Type the new name for the POU and press ENTER. |

## Removing POUs

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Select one or multiple POUs:<br>• Select an existing POU in the **Master Task**.<br>• Press and hold the CTRL key and select each POU in the **Master Task**. |
| 3 | Delete the selected POUs:<br>• Right-click a selected POU in the **Master Task** and choose **Delete POU** from the contextual menu that appears.<br>• Press the DELETE key. |

# Managing Rungs

## Creating a Rung

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Add a rung in a POU by any of the following methods:<br>• Right-click a POU and choose **Add Rung** from the contextual menu that appears.<br>• Select a POU and click  (**Add Rung** button) on the toolbar at the top of the **Tasks** tab.<br>• Select a POU and click  (**Create a new Rung** button) on the toolbar at the top of the programming workspace.<br>**Result**: A new rung is added to the program structure immediately below the last rung. |
| 3 | To reposition a rung in a POU, select a rung and click the UP or DOWN button on the toolbar at the top of the **Tasks** tab to move the selected rung up or down in the program structure. |
| 4 | The rung is given sequence identifier, such as `Rung0`. You may additionally add a rung comment to identify the rung by clicking the rung header. |
| 5 | The default programming language is **LD** (ladder). To select a different programming language for this rung, click **LD** and choose a different programming language. |
| 6 | If this rung is to be called with a *JUMP* instruction, assign a label to the rung by clicking the drop-down button below the rung sequence identifier **Rung***x*, where *x* is the rung number in a POU, and choose **%L** from the list.<br>**Result**: The rung is labeled as **%L***y*, where y is the label number. **%L** appears on the button and the label number *y* appears in suffix with the button.<br>**NOTE:** The label number is incremented by 1 as you define the next label.<br>To modify the label number, double-click the label number in a rung and enter the new number and then press ENTER. |

## Inserting a Rung Above an Existing Rung

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Select an existing rung in the **Programming** workspace. |
| 3 | Click  (**Insert a new Rung** button) on the toolbar at the top of the programming workspace.<br>**Result**: A new rung appears above the selected rung. |
| 4 | The rung is given sequence identifier, such as `Rung0`. You may additionally add a rung comment to identify the rung by clicking the rung header. |
| 5 | The default programming language is **LD** (ladder). To select a different programming language for this rung, click **LD** and choose a different language. |
| 6 | If this rung is to be called with a *JUMP* instruction, assign a label to the rung by clicking the drop-down button below the rung sequence identifier **Rung***x*, where *x* is the rung number in a POU, and choose **%L** from the list.<br>**Result**: The rung is labeled as **%L***y*, where y is the label number. **%L** appears on the button and the label number *y* appears in suffix with the button.<br>**NOTE:** The label number is incremented by 1 as you define the next label.<br>To modify the label number, double-click the label number in a rung and enter the new number and then press ENTER. |

## Copying Rungs

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Select one or multiple rungs:<br>• Select an existing rung.<br>• Press and hold the CTRL key and select each rung. |
| 3 | Right-click one of the selected rungs to copy and do one of the following methods:<br>• Choose **Copy selected rung** from the contextual menu that appears.<br>• Press CTRL + C. |
| 4 | Right-click a rung and do one the following methods:<br>• Choose **Paste Rung** from the contextual menu that appears.<br>• Press CTRL + V.<br>**Result**: A copy of the rung is inserted with the same label as the original rung. Edit the label as required. |

**NOTE:** You can also copy and paste rungs in the **Programming** window:

| Step | Action |
|---|---|
| 1 | Right-click the rung to copy and choose **Copy selected rung**. |
| 2 | Right-click in the programming workspace and choose **Paste Rung**. |

## Renaming a Rung

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Edit the rung name by one of the following methods:<br>• Right-click a rung and choose **Rename Rung** from the contextual menu that appears.<br>• Double-click a rung.<br>• Select a rung and double-click the rung name or the text ***name*** in the programming workspace.<br>• Select a rung and press the F2 key. |
| 3 | Type the new name for the rung and press ENTER. |

## Removing Rungs

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Delete a rung by one of the following methods:<br>• Right-click a rung and choose **Delete Rung** from the contextual menu that appears.<br>• Select a rung and click  (**Delete Rung** button) on the toolbar at the top of the **Tasks** tab.<br>• Select a rung and click  (**Delete the Rung** button) on the toolbar at the top of the programming workspace.<br>• Right-click a rung in the programming workspace and choose **Delete the selected rung** from the contextual menu that appears.<br>• Select a rung and press the DELETE key. |
| 3 | If the rung is not empty, you are prompted to confirm deleting the rung. |

# Managing Grafcet (SFC) POUs

## Creating a Grafcet POU

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming window**. |
| 2 | Add a new Grafcet POU by one of the following methods:<br>• Right-click on **Master task** and choose **Add Grafcet POU** from the contextual menu that appears.<br>• Click the  (**Add Grafcet POU**) button on the toolbar at the top of the **Tasks** tab.<br>**Result:** A *n* - **Grafcet** node appears below the **Master task** node, where *n* is an integer incremented each time a Grafcet POU is created. |

## Inserting a New Grafcet POU

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming window**. |
| 2 | Select an existing Grafcet POU above which to insert the new Grafcet POU. |
| 3 | Right-click the selected POU and choose **Insert Grafcet POU** from the contextual menu that appears. |
| 4 | To reposition a Grafcet POU in the **Master Task**, select a Grafcet POU and click the UP or DOWN button on the toolbar at the top of the **Tasks** tab to move the selected Grafcet POU up or down in the program structure. |

## Copying Grafcet POUs

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Select one or multiple Grafcet POUs:<br>• Select an existing Grafcet POU in the **Master Task**.<br>• Press and hold the CTRL key and select each Grafcet POU in the **Master Task**. |
| 3 | Right-click one of the selected Grafcet POU in the **Master Task** and choose **Copy POU** from the contextual menu that appears. |
| 4 | Right-click the **Master Task** and choose **Paste POU** from the contextual menu that appears.<br>**Result**: One or multiple Grafcet POUs are added to the program structure immediately below the selected Grafcet POU in the **Master Task** with the same name as the copied Grafcet POU. |

## Renaming a Grafcet POU

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Edit the Grafcet POU name by one of the following methods:<br>• Right-click on a Grafcet POU and choose **Rename POU** from the contextual menu that appears.<br>• Double-click a Grafcet POU.<br>• Select a Grafcet POU and press the F2 key. |
| 3 | Type the new name for the Grafcet POU node and press ENTER. |

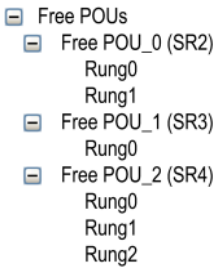
## Removing Grafcet POUs

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Select one or multiple Grafcet POUs:<br>• Select an existing Grafcet POU in the **Master Task**.<br>• Press and hold the CTRL key and select each Grafcet POU in the **Master Task**. |
| 3 | Delete the selected Grafcet POUs:<br>• Right-click a selected Grafcet POU in the **Master Task** and choose **Delete POU** from the contextual menu that appears.<br>• Press the DELETE key. |

# Free POUs

## Introduction

In EcoStruxure Machine Expert - Basic, a Free POU is a special type of POU that is not explicitly associated with a task:




Each Free POU is implemented as a subroutine and is made up of 1 or more rungs written in the Ladder or IL programming languages.

**NOTE:** Grafcet POUs cannot be Free POUs.

Free POUs are consumed when:

- Called using a subroutine call (SRi) from within a program rung
- Configured as the periodic task
- Configured as an event task, for example, the subroutine for threshold 0 of a High Speed Counter (HSC) function block (%HSCi.TH0)

When consumed as periodic or event tasks, the Free POU subroutine is automatically moved from the **Free POUs** area of the **Tasks** window to the **Periodic Task** or **Events** area of the window, respectively.

When no longer consumed as a periodic task of event task, the subroutine moves back to the **Free POUs** area and available to be consumed by other tasks or events.

## Creating a New Free POU

Proceed as follows to create a new Free POU:

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Right-click on **Free POUs** and choose **Add Free POU** from the contextual menu that appears.<br><br>**Result**: A new POU with the default name "`Free POU_0`" and default subroutine number "`SR0`" appears below the **Free POUs** branch and a new rung appears in the **Programming** workspace. |
| 3 | Optionally, right-click on the new POU and choose **Rename POU**, then type a new name for the POU and press Enter.<br><br>The name of the Free POU is also updated in the rung that appears in the **Programming** workspace. |
| 4 | Optionally, type a comment, page 125 to associate with the Free POU. |
| 5 | Select **Subroutine number** to the right of the comment box and choose a subroutine number from the list.<br><br>**Result**: The POU description in the **Free POUs** list is updated with the subroutine number chosen, for example "SR11". |
| 6 | Create the rungs/steps and source code for the Free POU/Free Grafcet POU, in the Ladder or IL programming language. |

## Copying Existing Free POUs

Proceed as follows to copy and paste existing POUs associated with a task to create a Free POU:

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Select one or multiple Free POUs:<br>• Select an existing Free POU.<br>• Press and hold the CTRL key and select each Free POU. |
| 3 | Right-click one of the selected Free POU and choose **Copy POU** from the contextual menu that appears. |
| 4 | Right-click and choose **Paste POU** from the contextual menu that appears.<br><br>**Result**: One or multiple new Free POUs with the name **Free POU_x**, where x is the next available Free POU number, and default subroutine number **SRx** , where x is the next available subroutine number, appear below **Free POU**s. All rungs of the POU are automatically associated with the new Free POU subroutine number. |

## Copying Existing POUs Associated with a Task

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Select one or multiple POUs:<br>• Select an existing POU in the **Master Task**.<br>• Press and hold the CTRL key and select each POU in the **Master Task**. |
| 3 | Right-click one of the selected POU in the **Master Task** and choose **Copy POU** from the contextual menu that appears. |
| 4 | Right-click the **Master Task** and choose **Paste POU** from the contextual menu that appears.<br><br>**Result**: One or multiple POUs are added to the program structure immediately below the selected POU in the **Master Task** with the same name as the copied POU. |

## Exporting of Free POUs

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the upper left-hand area of the **Programming** window. |
| 2 | Select one or multiple existing Free POUs in the **Master Task**. |
| 3 | Right-click on the selected Free POUs in the **Master Task** and choose **Export POU** from the contextual menu that appears. |
| 4 | Save the Export Free POU files (*.smbf) in the Export folder that appears. |

## Importing of Free POUs

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the upper left-hand area of the **Programming** window. |
| 2 | Select one or multiple existing Free POUs in the **Master Task**. |
| 3 | Right-click on the selected Free POUs in the **Master Task** and choose **Import POU** from the contextual menu that appears. |
| 4 | Select the Free POU files (*.smbf) from the folder that appears, and click **Open**.<br>**NOTE:** If a maximum number of Free POUs is reached or the file is corrupted (invalid format), an error message appears and the Free POUs are not imported. |

## Removing Free POUs

Proceed as follows to remove Free POUs:

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Select one or multiple Free POUs:<br>• Select an existing Free POU.<br>• Press and hold the CTRL key and select each Free POU. |
| 3 | Delete the selected Free POUs:<br>• Right-click a selected Free POU and choose **Delete POU** from the contextual menu that appears.<br>• Press the DELETE key. |

**NOTE:** Unassign, page 87 a Free POU from a task before removing it.

## Assigning Free POUs to Events or Periodic Tasks

By default, Free POUs and subroutines are not associated with any events or tasks.

Refer to Creating Periodic Task, page 86 for information on how to associate a Free POU with a periodic task.

Refer to Creating Event Task, page 90 for information on how to associate a Free POU with an event.

# User-Defined Functions

## Overview

A user-defined function allows you to create new functions with input parameters, local variables and a return value. User-defined functions are stored as part of the EcoStruxure Machine Expert - Basic project.

You can call user-defined functions in:

- The Master task
- Periodic tasks
- Events
- Free POUs

**NOTE:** The application must be configured with a functional level, page 54 of at least **Level 6.0** to support user-defined functions.

## Creating a User-Defined Function

### Adding a New User-Defined Function

You can have up to 64 user-defined functions in a project.

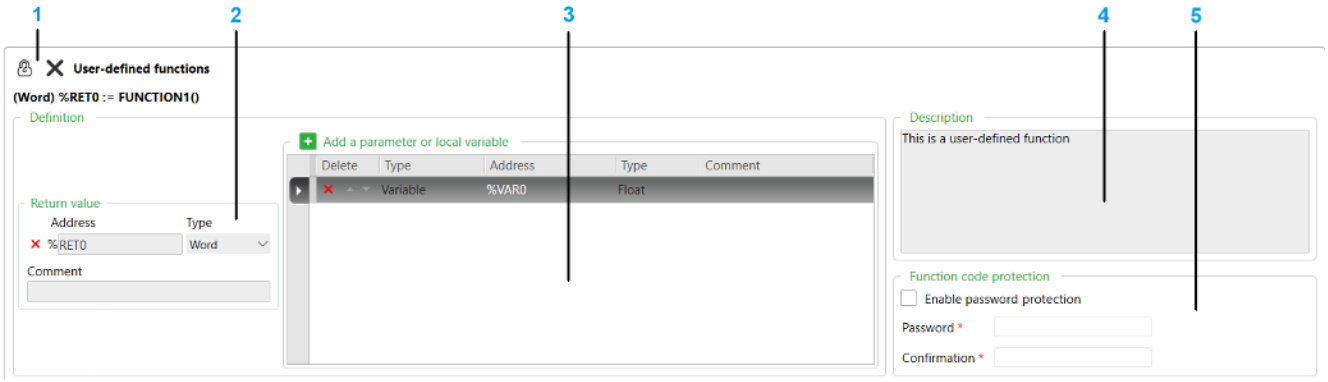
| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Create a user-defined function using one of the following methods:<br>• Right-click **User-defined functions** and choose **Add user-defined function** from the contextual menu that appears.<br>• Select **User-defined functions** and click  (**Add user-defined-function**) on the toolbar at the top of the **Tasks** tab.<br>• Select an existing user-defined function, right click and choose **Insert user-defined function**.<br>**Result**: A new user-defined function is added to the program structure at the bottom of the list. If you inserted a user-defined function, the new user-defined function is above the selected one. The default name is **FUNCTIONn**, where n is an integer incremented each time a user-defined function is created. |
| 3 | Optionally, rename the user-defined function. Refer to Renaming a User-Defined Function, page 73. |
| 4 | Define the user-defined function. Refer to Defining a User-Defined Function, page 68. |

You create and manage rungs in a user-defined function in the same way as rungs in a POU. Refer to Managing Rungs, page 61.

# Defining a User-Defined Function

## Presentation

The following illustration shows the actions that are available in the **Properties** view of the user-defined function:




**1 Toolbar**

- Detach the properties view.
- Close the properties view.

**2 Return value area**

- Add a return value
- Delete a return value
- Select the return value type from the menu.

**3 Parameter/local variable area**

- Add a parameter or a local variable
- Delete a parameter or a local variable

**4 Description area**

Optionally, you can write a description of the purpose of the user-defined function. This description appears in a tooltip when you use the user-defined function in an **operation block** or **comparison block**.

**5 Function code protection area**

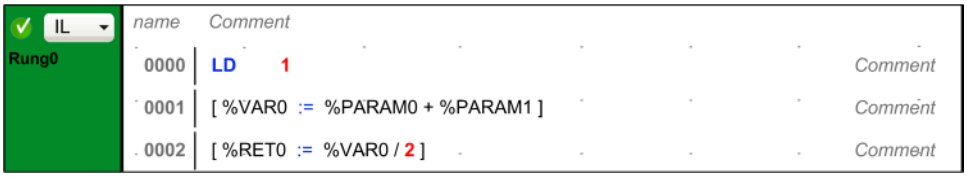
Optionally, you can define a password to help protect the content of the user-defined function. If you enable the password encryption, the password is requested to view/modify the content of the user-defined function. However, the password is not requested to use the user-defined function in an application.

# Programming a User-Defined Function

To program a user-defined function:

| Step | Action |
|---|---|
| 1 | Add a new user-defined function. Refer to Adding a User-Defined Function, page 67. |
| 2 | Define the interface of the user-defined function by defining the **Return value**, the input **Parameters** and the **Local variables**. Refer to Defining the Interface of a User-Defined Function, page 70. |
| 3 | Click **Apply**. |
| 4 | Define the functionality of the user-defined function in one or more Ladder/IL rungs, page 61:<br>1. Insert a Ladder structure element.<br>2. Program the user-defined function.<br>For example:<br>  |

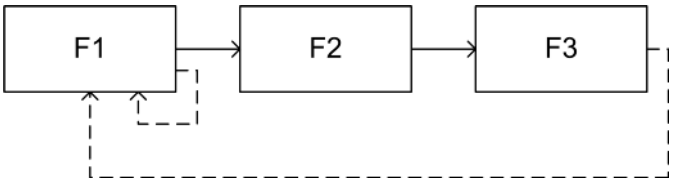
You can also directly program the user-defined function in the IL editor:




You can call other user-defined functions in the rungs that implement a user-defined function.

**NOTE:** User-defined functions cannot be recursive: a user-defined function cannot call itself directly or indirectly.

Example:




**- - -** Not allowed

A user-defined function cannot call a subroutine, but a subroutine can call a user-defined function.

## Defining the Interface of a User-Defined Function

To use a user-defined function, you must define the object types and their data types. You can modify the default name.

| Object Type | Default Name | Data Type | Description |
|---|---|---|---|
| **Return value** | *%RET0* | **Word**<br>**Double** | Value returned by the user-defined function.<br>Can only be used in a rung of a user-defined function. |
| **Parameters** | *%PARAMn*(1) | **Float** | Parameter of a user-defined function.<br>Can only be used in a rung of a user-defined function.<br>You can add parameters (including function blocks instances) to animation tables.<br>In online mode, the current values of parameters are not displayed on IL/Ladder editor. |
| **Local variables** | *%VARn*(1) | | Variables used to store data values within the user-defined function.<br>Can only be used in a rung of a user-defined function.<br>You can add local variables to animation tables.<br>In online mode, the current values of local variables are not displayed on IL/Ladder editor. |
| (1) *n* is an integer incremented each time a parameter or a local variable is created. | | | |

These objects are optional.

## Interfaces Variables and Global Variables

The three following variables can only be used in the rungs that implement the user-defined function:

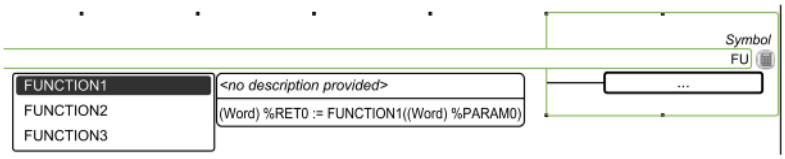
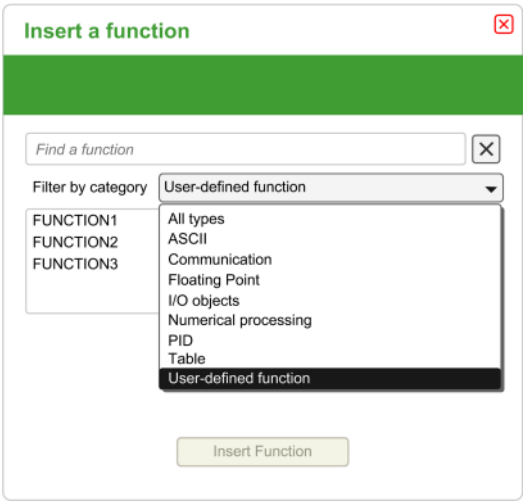
- *%RET0*
- *%PARAMn*
- *%VARn*

Global variables are the other variables that you can use in an EcoStruxure Machine Expert - Basic program, including the rungs of a user-defined function.

## Using User-Defined Functions

Once defined, user-defined functions can be used anywhere in the program using an **Operation Block** or **Comparison Block** in the same way as any other function.

In the Ladder editor:

| Step | Action |
|---|---|
| 1 | Click the **Operation Block** or **Comparison Block** button on the toolbar. |
| 2 | Click in the Action zone, page 114 of the rung to insert the block. |
| 3 | Click the **Selection mode** button on the toolbar. |
| 4 | Double-click the **Operation expression** line.<br>You can either:<br>• Type the name of the user-defined function. For example, for the name “FUNCTION1”, type “FU” and the names of all user-defined functions that begin with “FU” appear:<br>  <br>• Use Smart Coding, page 123:<br>1. Click the Smart Coding button.<br>2. Select **Filter by category** then **User-defined function**.<br>3. Select the user-defined function.<br>  |
| 5 | Click **Insert Function**. |
| 6 | Complete the definition of the user-defined function by typing the return value and the parameters as defined in Defining the Interface of a User-Defined Function, page 70. |

# Managing User-Defined Functions

## User-Defined Functions in Offline and Online Modes

You can manage user-defined functions in offline mode.

In online mode, you can:

- add a rung to an existing user-defined function
- modify a rung calling a user-defined function

## Copying/Cutting Existing User-Defined Functions

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Select one or more user-defined functions:<br>• Click to select a user-defined function.<br>• Select multiple user-defined functions by pressing and holding the CTRL key. |
| 3 | Right-click and choose **Copy user-defined functions** or **Cut user-defined functions** from the contextual menu that appears. |
| 4 | Right-click **User-defined functions** and choose **Paste user-defined function** from the contextual menu that appears.<br>**Result**: One or more user-defined functions are added at the end of the program structure in **User-defined functions**. EcoStruxure Machine Expert - Basic automatically assigns new name to the copied user-defined function. |

## Exporting User-Defined Functions

User-defined functions are stored as part of the project. If you want to use a user-defined function in another project, you have to export it, then import it to the other project.

You can copy/paste between EcoStruxure Machine Expert - Basic instances.

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Select one or more user-defined functions:<br>• Click to select a user-defined function.<br>• Select multiple user-defined functions by pressing and holding the CTRL key. |
| 3 | Right-click the selected user-defined functions in **User-defined functions** and choose **Export user-defined function** from the contextual menu that appears. |
| 4 | Save the user-defined function file (*.smbf) in the **Export** folder that appears. |

## Importing a User-Defined Function

User-defined functions are stored as part of the project. If you want to use a user-defined function in another project, you have to export it, then import it to the other project.

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Select a user-defined function in **User-defined functions**. |
| 3 | Right-click the selected user-defined function in the **User-defined functions** and choose **Import user-defined function** from the contextual menu that appears. |
| 4 | Navigate to the folder containing the user-defined function file (*.smbf) and select the user-defined function. |
| 5 | Confirm with **OK**. |

## Renaming a User-Defined Function

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Rename using one of the following methods:<br>• Right-click a user-defined function and choose **Rename user-defined function** from the contextual menu that appears.<br>• Double-click the user-defined function in the programming workspace.<br>• Select a user-defined function and press the F2 key. |
| 3 | Enter the new name for the user-defined function and press ENTER.<br>The accepted characters are A...Z, 0...9, _. The name must be unique. Otherwise, the name remains unchanged. |

## Deleting User-Defined Functions

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Select one or more user-defined functions by pressing and holding the CTRL key. |
| 3 | Delete the selected user-defined functions using one of the following methods:<br>• Right-click a selected user-defined function in **User-defined functions** and choose **Delete user-defined function** from the contextual menu that appears.<br>• Press the DELETE key.<br>• Click  on the toolbar at the top of the **Tasks** tab. |

# User-Defined Function Blocks

## Overview

A user-defined function block allows you to create new function blocks with one or more input and output parameters, and local variables. User-defined function blocks are stored as part of the EcoStruxure Machine Expert - Basic project.

You can call user-defined function blocks in:

- The Master task
- Periodic tasks
- Events
- Free POUs

**NOTE:** The application must be configured with a functional level, page 54 of at least **Level 6.0** to support user-defined function blocks.

## Creating a User-Defined Function Block

### Adding a New User-Defined Function Block

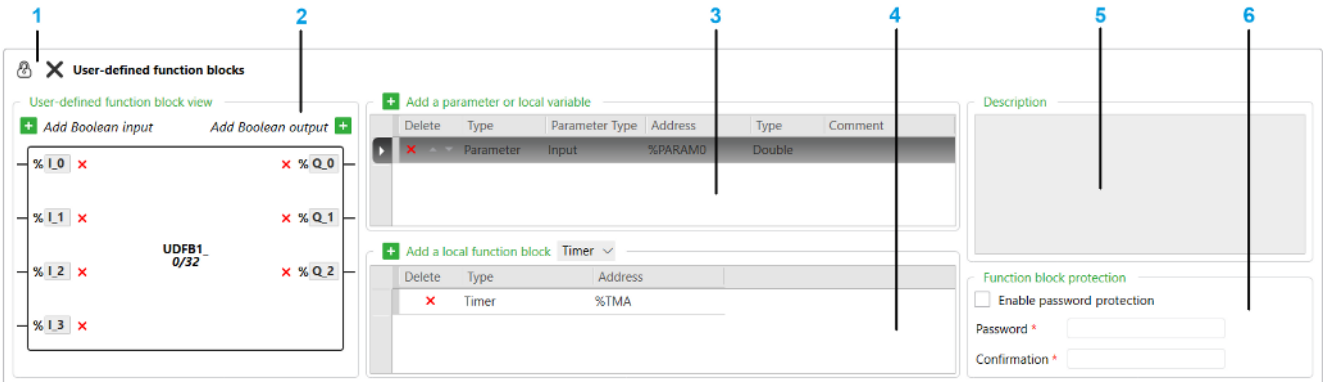
| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Create a user-defined function block using one of the following methods:<br>• Right-click **User-defined function blocks** and choose **Add user-defined function block** from the contextual menu that appears.<br>• Select **User-defined function blocks** and click  (**Add user-defined function block**) on the toolbar at the top of the **Tasks** tab.<br>• Select an existing user-defined function block, right click and choose **Insert user-defined function block**.<br>**Result**: A new user-defined function block is added to the program structure immediately at the bottom of the list. If you inserted a user-defined function block, the new user-defined function block is above the selected one. The default name is **UDFB**n, where n is an integer incremented each time a user-defined function block is created. |
| 3 | Optionally, rename the user-defined function block. Refer to Renaming a User-Defined Function Block, page 80. |
| 4 | Define the user-defined function block. Refer to Defining a User-Defined Function Block, page 75. |

You create and manage a rung in a user-defined function block in the same way as a rung in a POU. Refer to Managing Rungs, page 61.

# Defining a User-Defined Function Block

## Presentation

The following illustration shows the actions that are available in the **Properties** view of the user-defined function:




**1 Toolbar**

- Detach the properties view.
- Close the properties view.

**2 User-defined function block area**

- Add an input or an output
- Delete an input or an output
- The number of instances in the user logic for a function block definition is displayed under the function block name (in the above example: 32).

**3 Parameter/local variable area**

- Add a parameter or a local variable
- Delete a parameter or a local variable
- To modify the values of input/output parameters, refer to Modifying Input/Output Parameter Values of a User-Defined Function Block, page 78.

**4 Local function block area**

- Add a local function block
- Delete a local function block
- Select the function block type from the menu.

**5 Description area**

Optionally, you can write a description of the purpose of the user-defined function block. This description appears in a tooltip when you use the user-defined function block in an **operation block**.

**6 Function block protection area**

Optionally, you can define a password to help protect the content of the user-defined function block. If you enable the password encryption, the password is requested to view/modify the content of the user-defined function block. However, the password is not requested to use the user-defined function block in an application.

# Programming a User-Defined Function Block

To program a user-defined function block:

| Step | Action |
|---|---|
| 1 | Add a new user-defined function block. Refer to Adding a User-Defined Function Block, page 67. |
| 2 | Define the interface of the user-defined function block by defining the input **Parameters** and the **Local variables**. Refer to Defining the Interface of a User-Defined Function Block, page 76. |
| 3 | Click **Apply**. |
| 4 | Specify the functionality of the user-defined function block in one or more Ladder/IL rungs, page 61:<br>1. Click the **Function blocks**  button on the toolbar.<br>2. Select UDFB → the user-defined function block you want to insert.<br>3. Click in the Action zone, page 114 of the rung.<br>4. Program the user-defined function block. |

You cannot program a user-defined function block in the IL editor.

# Defining the Interface of a User-Defined Function Block

To use a user-defined function block you have to define the inputs, outputs, object types and their data types. You can change the default name.

| Object type | Default Name | Data Type | Description |
|---|---|---|---|
| **Inputs** | *%I_n*(1) | **Boolean** | Input pins for the function block. |
| **Outputs** | *%Q_n*(1) | **Boolean** | Output pins for the function block. |
| **Parameters** | *%PARAMn*(1) | **Word**<br>**Double**<br>**Float** | Can only be used in a rung of a user-defined function and user-defined function block.<br>You can add parameters of user-defined function block instances to animation tables. |
| **Local variables** | *%VARn*(1) | | Can only be used in a rung of a user-defined function and user-defined function block.<br>You can add local variables of user-defined function block instances to animation tables. |
| **Local Function Blocks** | **Type** and **Address**(2) cannot be modified | List of data types | Can only be used in a rung of a user-defined function block. |

(1) *n* is an integer incremented each time an object is created.

(2) A new instance is allocated to the next index after the last global instance used. For example, if %TM11 was the last timer function block used in the programming area, %TMA will be allocated to %TM12.

**NOTE:** You cannot allocate a new instance if the last timer function block used was %TM254.

## Interface Variables

The interface variables are the variables and parameters defined in the interface of user-defined function block, that can only be used in the rungs that implement the user-defined function block:

- *%I_n*
- *%Q_n*
- *%PARAMn*
- *%VARn*

  (where n is an integer)

## Global Variables

All objects not declared as interface variables are considered as global variables. This includes any function blocks and Ladder items added to the user-defined function block from the toolbar. As these global variables are shared between all instances of the user-defined function block and the entire program, access the objects sequentially in the application. Some examples of global variables:

- %MW*n* word
- %TM*n* function block
- RISING*n* function

(where *n* is an integer)

## Using a User-Defined Function Block

To insert a user-defined function block into a rung:

| Step | Action |
|---|---|
| 1 | Click the **Function blocks**  button on the toolbar. |
| 2 | Select UDFB → the user-defined function block you want to insert. |
| 3 | Click in the Action zone, page 114 of the rung. |
| 4 | Optionally use operation blocks to read or write function block parameters.<br>The syntax is *%<UDFB name><instance number>.PARAM*n, where n is an integer corresponding to the parameter number.<br>Example:<br>• You defined a user-defined function block named *MY_FB* with a parameter *%PARAM*0.<br>• An instance of this user-defined function block is placed in the Master task and the instance number 0 is assigned to it.<br>**Result:** The object *%MY_FB0.PARAM0* is available in any tasks. |

## Modifying an Existing User-Defined Function Block

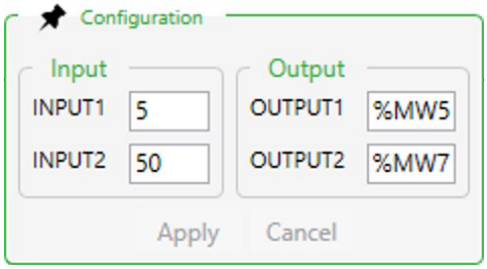
A user-defined function block is considered existing once it is used in your logic. This is indicated in the properties view of the user-defined function block definition, page 75.

You can edit the inputs, outputs, parameters and variables of the function block in the same fashion that they were created without additional restriction.

| ⚠WARNING |
| --- |
| **UNINTENDED EQUIPMENT OPERATION**<br>Verify all existing instances of the user-defined function block after editing any input or output definitions such that the logic of their use functions as originally intended.<br>**Failure to follow these instructions can result in death, serious injury, or equipment damage.** |

## Modifying Input/Output Parameter Values of a User-Defined Function Block

To modify a user-defined function block input or output parameter in offline mode:

| Step | Action |
| --- | --- |
| 1 | In the **Programming** window, move the cursor over the user-defined function block in the Ladder editor.<br>**Result:** The **Configuration** tooltip appears.<br>The following illustration shows an example of the **Configuration** tooltip:<br>  |
| 2 | Optionally, click the pin icon in the top left corner to retain the **Configuration** tooltip on the screen. |
| 3 | Click a field to modify and enter the value.<br>• For input parameters, immediate values or values from objects compatible with the parameter type are accepted.<br>• For output parameters, values are assigned to objects compatible with the parameter type. |
| 4 | To validate the modifications, you can use one of the following methods:<br>• Click **Apply**<br>• Click outside of the **Configuration** tooltip. When prompted, click **OK** to apply the modifications.<br>**Result:** The parameters are modified. |

# Managing User-Defined Function Blocks

## User-Defined Function Blocks in Offline and Online Modes

You can manage the user-defined function blocks in offline mode.

In online mode, you can:

- add a rung to an existing user-defined function block
- modify a rung calling a user-defined function block

## Copying/Cutting Existing User-Defined Function Blocks

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Select one or more user-defined function blocks:<br>• Click to select a user-defined function block.<br>• Select multiple user-defined function blocks by pressing and holding the CTRL key. |
| 3 | Right-click and choose **Copy user-defined function blocks** or **Cut user-defined function blocks** from the contextual menu that appears. |
| 4 | Right-click **User-defined function blocks** and choose **Paste user-defined function block** from the contextual menu that appears.<br>**Result**: One or more user-defined function blocks are added at the end of the program structure in **User-defined function blocks**. EcoStruxure Machine Expert - Basic automatically assigns new name to the copied user-defined function block. |

## Exporting User-Defined Function Blocks

User-defined function blocks are stored as part of the project. If you want to use a user-defined function block in another project, you have to export it, then import it to the other project.

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Select one or more user-defined function blocks:<br>• Click to select a user-defined function block.<br>• Select multiple user-defined function blocks by pressing and holding the CTRL key. |
| 3 | Right-click the selected user-defined function blocks in **User-defined function blocks** and choose **Export user-defined function block** from the contextual menu that appears. |
| 4 | Save the user-defined function block file (*.smbf) in the **Export** folder that appears. |

## Importing a User-Defined Function Block

User-defined function blocks are stored as part of the project. If you want to use a user-defined function block in another project, you have to export it, then import it to the other project.

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Select a user-defined function in **User-defined function blocks**. |
| 3 | Right-click the selected user-defined function in the **User-defined functions blocks** and choose **Import user-defined function block** from the contextual menu that appears. |
| 4 | Navigate to the folder containing the user-defined function block file (*.smbf) and select the user-defined function block. |
| 5 | Confirm with **OK**. |

## Renaming a User-Defined Function Block

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Rename using one of the following methods:<br>• Right-click a user-defined function block and choose **Rename user-defined function block** from the contextual menu that appears.<br>• Double-click the user-defined function block's name in the programming workspace.<br>• Select a user-defined function block and press the F2 key. |
| 3 | Enter the new name for the user-defined function block and press ENTER.<br>The accepted characters are A...Z, 0...9, _. The name must be unique. Otherwise, the name remains unchanged. |

## Deleting User-Defined Function Blocks

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Select one or more user-defined function blocks by pressing and holding the CTRL key. |
| 3 | Delete the selected user-defined function blocks using one of the following methods:<br>• Right-click a selected user-defined function block in **User-defined function blocks** and choose **Delete user-defined function block** from the contextual menu that appears.<br>• Press the DELETE key.<br>• Click  on the toolbar at the top of the **Tasks** tab. |

# Master Task

## Master Task Description

### Overview

The master task represents the main task of the application program. It is obligatory and is created by default. The master task is made up of sections and subroutines represented within Program Organizational Units (POUs). Each POU of the master task can be programmed in any of the supported programming languages.

### Procedure

| For | Refer To |
|---|---|
| Creating a new POU in the master task | Creating a New POU Associated with a Task, page 59 |
| Renaming a POU in the master task | Renaming a POU, page 60 |
| Removing a POU from the master task | Removing a POU, page 60 |

# Configuring Master Task

## Procedure

Follow these steps to configure the master task:

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Select the **Master Task** item.<br><br>**Result**: The **Master Task** properties appear in the lower central area of the EcoStruxure Machine Expert - Basic window. |
| 3 | Modify the properties as required. |
| 4 | Click **Apply** to save the changes. |

## Master Task Properties

**Scan Mode**

Choose the scan mode to use for the program:

- **Normal**: When a logic controller is in normal (freewheeling) scan mode, a new scan starts immediately after the previous scan has completed.
- **Periodic**: In periodic scan mode, the logic controller waits until the configured scan time has elapsed before starting a new scan. Every scan is therefore the same duration.

  Specify the scan **Period** for the periodic scan mode of 1...150 ms.

## System Bits and Words Controlling the Master Task

The master task can be controlled by system bits (%S) and system words (%SW).

This table lists the system bits:

| System Bits | Description |
|---|---|
| %S11 | Watchdog overflow |
| %S19 | Scan period overrun (periodic scan mode) |

This table lists the system words:

| System Words | Description |
|---|---|
| %SW0 | Logic controller scan period (periodic scan mode) |
| %SW30, %SW70 | Last scan time. Indicates the execution time of the last controller scan cycle, that is, the time elapsed between the start (acquisition of inputs) and the end (update of outputs) of a master task scan cycle. %SW30 provides the millisecond part, and %SW70 provides the microseconds part.<br><br>For example, if the scan time is 2.250 ms, %SW30 = 2 and %SW70 = 250. |
| %SW31, %SW71 | Maximum scan time. Indicates the execution time of the longest controller scan time since the last cold start of the logic controller. %SW31 provides the millisecond part, and %SW71 provides the microseconds part.<br><br>For example, if the scan time is 2.250 ms, %SW31 = 2 and %SW71 = 250. |
| %SW32, %SW72 | Minimum scan time. Indicates the execution time of the shortest controller scan time since the last cold restart of the logic controller. %SW32 provides the millisecond part, and %SW72 provides the microseconds part.<br><br>For example, if the scan time is 2.250 ms, %SW32 = 2 and %SW72 = 250. |

Refer to the *Programming Guide* for your hardware platform for a complete list of system bits and words and their meaning.

# Strings

## Overview

Strings are a sequence of bytes containing ASCII characters that you can store in the following memory objects:

- Memory words `%MW`
- Constant words `%KW`

There are two bytes in one word.

Syntax to program a string:

`%MWx:L`

**x** Index of the memory object

**L** Number of words used by the string and must be between 1...255.

The supported controllers have a little-endian architecture; the bytes are stored from the lowest order byte to the highest.

The following table shows you an example of the storage of the bytes for the string *Basic*:

| Memory objects | Hexadecimal | ASCII |
|---|---|---|
| `%MW0` or `%KW0` | 6142 | aB |
| `%MW1` or `%KW1` | 6973 | is |
| `%MW2` or `%KW2` | 0D63 | \rc (1) |
| (1) \r is the end of character. This character is mandatory and taken into account when processing the strings. It is configurable in the Application Behaviour window, page 56. The default value is CR (ASCII 13). | | |

**NOTE:** Starting functional level 11.0, it is possible to change the end character. Refer to Functional Levels, page 54.

You can write up to 509 characters.

**NOTE:** The memory objects are used as a variable or for a string. If you have configured a memory object for a string, do not configure any of the memory words it contains as a variable.

# Configuring Strings in Constant words

## Entering a String

| Step | Action |
|---|---|
| 1 | In the **Programming** window, click **Tools** > **Memory objects** > **Constant words**. |
| 2 | In **Constant word properties**, click **%KW**. |
| 3 | Click the ... button in the **Configuration** column for the constant word that you want to configure.<br><br>If the constant word is already configured the **Confirmation** window appears. Click **Ok** to overwrite the value. Otherwise, click **Cancel**.<br><br>**Result:** The **Constant string Assistant** window appears. |
| 4 | Enter the string.<br><br>**Result: Range of constants required**, including the termination marker, is displayed below the input text box for the string. |
| 5 | Click **Apply**. |

**Result:** The entered characters are applied to the corresponding and required constant variables. The end character selected in the Application Behavior window, page 56 is added to the string. The characters are inverted. Refer to the overview of this section, page 82.

# Assigning Strings in Memory Words

## Syntax

The following describes Instruction List syntax. You can insert Instruction List operations and assignment instructions, page 122 in Ladder Diagram rungs using an **Operation Block** graphical element.

For assigning a string in a memory word, use this syntax: `Op1 := "Your string".`

For example:
`%MW10:20 := "This is a Basic string."`

If you want the software to calculate the memory space needed, type `%MWx:? := "Your string".`

## Rules of Use

When you assign a string:

- Make sure there is no overlapping. You can overwrite a string by another string.
- The use of the quotation character is not supported for immediate string assignments.

# Managing Strings

## Introduction

The following functions allow you to:

- Copy a string.
- Get the length of a string.
- Concatenate two strings.
- Compare two strings.

## Syntax

The following describes Instruction List syntax. You can insert Instruction List operations and assignment instructions, page 122 in Ladder Diagram rungs using an **Operation Block** graphical element.

## Copying a String

For copying a string, use this syntax: `Op1 := Op2`.

The following table presents the authorized memory objects for Op1 and Op2:

| Parameters | Description |
|---|---|
| Op1 | `%MWx:L` |
| Op2 | `%MWy:L` or `%KWy:L` |
| **x, y** Indexes of the memory object<br>**L** must be the same for both `Op1` and `Op2` | |

## Getting the Length of a String

For getting the length of a string, use this syntax: `Op1 := LENGTH(Op2)`.

The following table presents the authorized memory objects for Op1 and Op2:

| Parameters | Description |
|---|---|
| Op1 | `%MWx` |
| Op2 | `%MWy:L` or `%KWy:L` |
| **x, y** Indexes of the memory object | |

Immediate strings are not accepted.

## Concatenating Two Strings

For concatenating two strings, use this syntax: `Op1 := CONCAT(Op2, Op3).`

The following table presents the authorized memory objects for Op1, Op2 and Op3:

| Parameters | Description |
|---|---|
| Op1 | `%MWx:L` |
| Op2 | `%MWy:A` or `%KWy:A` |
| Op3 | `%MWz:B` or `%KWz:B` |
| **x, y, z** Indexes of the memory object<br>EcoStruxure Machine Expert - Basic does not validate that L is sufficiently sized for the concatenation. Be sure that Op1 is of an adequate, minimum length for the operation. | |

Immediate strings are not accepted.

The following table presents the process of concatenation:

| Stage | Description |
|---|---|
| 1 | The application copies Op2 into Op1. |
| 2 | The copy stops if:<br>• The end character of Op2 is reached.<br>**NOTE:** The end character is configurable in the Application Behavior window, page 56. The default value is 'CR' (carriage return).<br>• The memory space assigned to Op2 is copied. *%S28* is set to TRUE. Refer to System Bits Description.<br>• The entire memory space of Op1 is filled. *%S28* is set to TRUE. |
| 3 | If the memory space of Op1 is not filled, the application continues by copying Op3 into Op1. |
| 4 | The copy stops if:<br>• The configured end character of Op3 is reached.<br>**NOTE:** The end character is configurable in the Application Behavior window, page 56. The default value is 'CR' (carriage return).<br>• The memory space assigned to Op3 is copied. *%S28* is set to TRUE.<br>• The entire memory space of Op1 is filled. *%S28* is set to TRUE. |
| 5 | The application ensures that Op1 ends with the configured end character. The last character of Op1 may be replaced by the configured end character if the memory space is filled. |

## Comparing Two Strings

For comparing two strings, use this syntax: `Op1 := EQUAL_STR(Op2, Op3).`

The following table presents the authorized memory objects for Op1, Op2 and Op3:

| Parameters | Description |
|---|---|
| Op1 | `%MWx` |
| Op2 | `%MWy:A` or `%KWy:A` |
| Op3 | `%MWz:B` or `%KWz:B` |
| **x, y, z** Indexes of the memory object | |

When the application detects a different character, Op1 equals the index position of the first different character encountered left to right.

The following table presents examples of the result of string comparison:

| Op2 | Op3 | Different character | Op1 |
|---|---|---|---|
| azerty | qwerty | First one | 0 |
| 123456 | 124356 | Third one | 2 |
| EcoStruxure Machine Expert - Basic | EcoStruxure Machine Expert - Basic | – | -1 |

The following table presents the process of a string comparison:

| If | And if | Then |
|---|---|---|
| The application reaches the configured end character of Op2<br>**NOTE:** The end character is configurable in the Application Behavior window, page 56. The default value is 'CR' (carriage return). | Op2 = Op3 | `Op1 := -1` |
| | Op2 ≠Op3 | Op1 equals the different character position. |
| The application finds a different character before reaching the end of Op2 or Op3. | – | Op1 equals the different character position. |
| The end of the memory space assigned to Op2 or Op3 is reached | A ≠ B | Op1 equals the different character position and *%S28* is set to TRUE. Refer to System Bits Description. |
| | A = B | `Op1 := -1` and *%S28* is raised. |

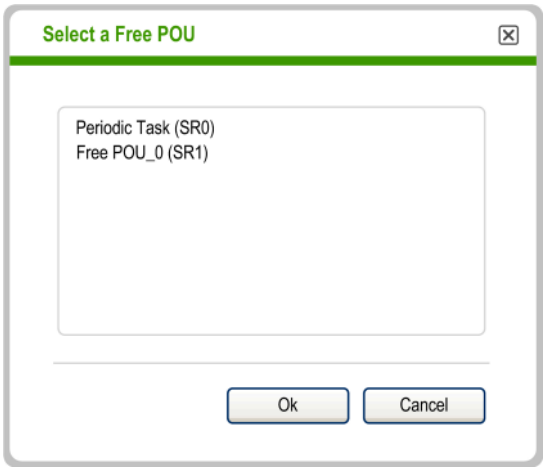
# Periodic Task

## Creating a Periodic Task

### Overview

A periodic task is a subroutine, usually of short duration, that is processed periodically. In EcoStruxure Machine Expert - Basic, this subroutine is implemented as a Free POU, page 64. The subroutine can be written in any of the programming languages supported by EcoStruxure Machine Expert - Basic.

## Assigning a Subroutine to a Periodic Task

| Step | Action |
|---|---|
| 1 | Create a new Free POU, page 65 containing the periodic task subroutine. |
| 2 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 3 | Assign a subroutine to the periodic task by one of the following methods:<br>• Select the **Periodic Task** and click  (**Assign Free POU** button) on the toolbar at the top of the **Tasks** tab.<br>• Right-click the **Periodic Task** and choose **Assign Free POU** from the contextual menu that appears.<br>**Result**: The **Select a Free POU** window is displayed:<br>  <br>**NOTE:** You can directly add a Free POU to the periodic task. Right-click the **Periodic Task** and choose **Add Free POU** from the contextual menu that appears. In this case, a Free POU is created and assigned to the periodic task. |
| 4 | Select a Free POU to assign to the periodic task and click **OK**.<br>**Result**: The selected subroutine is assigned to the **Periodic Task** and no longer available in the **Free POU** branch of the **Tasks** tab.<br>For example, if the Free POU “Free POU_0” containing the subroutine `SR4` is assigned to the periodic task, the **Free POU_0 (%SR4)** subroutine moves from the **Free POU** branch to the **Periodic Task** branch of the **Tasks** tab. |

## Removing a Subroutine from a Periodic Task

| Step | Action |
|---|---|
| 1 | Click the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Remove the subroutine from the **Periodic Task** by one of the following methods:<br>• Select the **Periodic Task** and click  (**Unassign Free POU** button) on the toolbar at the top of the **Tasks** tab.<br>• Right-click the **Periodic Task** and choose **Unassign Free POU** from the contextual menu that appears.<br>**Result**: The selected subroutine is removed from the **Periodic Task** and available as a Free POU in the **Free POUs** branch of the **Tasks** tab. |

# Configuring Periodic Task Scan Duration

## Procedure

Follow these steps to configure scan duration for periodic task:

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Select the **Periodic Task** item.<br><br>**Result**: The **Periodic Task** properties appear in the lower central area of the EcoStruxure Machine Expert - Basic window. |
| 3 | Modify the properties as required. |
| 4 | Click **Apply** to save the changes. |

## Periodic Task Properties

Specify the scan **Period** for the periodic task from 1...255 ms. The default value is 255 ms.

# Event Task

# Overview of Event Tasks

## Introduction

An event task:

- Is a part of a program executed when a given condition is met (event source)
- Has a higher priority than the main program
- Produces a rapid response time, enabling the overall response time of the system to be reduced.

## Description of an Event

An event is composed of:

- An *event source*: a software or hardware condition that interrupts the program when the event is triggered.
- A *POU*: an independent program entity (subroutine) associated with an event.
- A *priority level*: a priority assigned to events to determine the order in which they are executed.

# Event Sources

## Overview

8 event sources are available:

- 4 linked to selected physical inputs of the logic controller
- 4 linked to the *HSC* function block thresholds

An event source is always attached to a single event. When an event is triggered, it interrupts the controller, which then executes the subroutine associated with the event.

## Physical Input Events of a Logic Controller

The embedded digital inputs %I0.2, %I0.3, %I0.4 and %I0.5 of a logic controller can be configured as event sources (filtering must be disabled).

For each of these event sources, you can choose to:

- Trigger events on detection of a rising edge, falling edge, or both rising and falling edges
- Assign a priority to the event
- Identify the subroutine associated with the event.

For further details on configuring input events, refer to the *Programming Guide* of the logic controller.

## Threshold Output Event of an %HSC Function Block

The threshold outputs TH0 and TH1 of the *%HSC* function block can be used as event sources. Outputs TH0 and TH1 are set as follows:

- TH0 = 0 and TH1 = 0 when the value is less than threshold S0 and threshold S1
- TH0 = 1 and TH1 = 0 when the value is greater than threshold S0 and less than threshold S1
- TH0 = 1 and TH1 = 1 when the value is greater than threshold S0 and threshold S1

For each of these event sources, you can choose to:

- Trigger events on detection of a rising edge, falling edge, or both rising and falling edges.
- Assign a priority to the event.
- Identify the subroutine associated with the event.

A rising or falling edge of these outputs can activate an event process.

For further details on configuring output event, refer to the *Programming Guide* of the logic controller.

# Event Priorities

## Overview

Events have one of 8 possible priorities, from 7 (the lowest) to 0 (the highest).

Assign a priority to each event source. Two events cannot have the same priority. Thus, the order of execution depends on their relative priorities and the order in which they are detected.

Event tasks interrupt both master and periodic task execution. For more information, refer to Event Priority Over Master and Periodic Tasks, page 58.

**NOTE:** Care must be exercised when writing to global areas of memory or affecting I/O values when event tasks are called during the execution of other tasks. Modifying values that are otherwise used in the other tasks could affect logical outcomes of those tasks adversely.

**⚠WARNING**

**UNINTENDED EQUIPMENT OPERATION**

Thoroughly test and validate all tasks (Master, Periodic and any Event tasks) and the interactive affect they have on one another before putting your application into service.

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

For configuring the event task priorities, refer to the *Programming Guide* of your controller.

## Event Management

Each time an interrupt linked to an event source appears, the following sequence is launched:

| Stage | Description |
|---|---|
| 1 | Interrupt event occurs. |
| 2 | Save the context. |
| 3 | Execution of the programming section (subroutine labeled SRi:) linked to the event. |
| 4 | Update the embedded outputs. |
| 5 | Restore the context. |

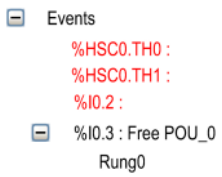
# Viewing Event Tasks

## Overview

Event tasks are displayed in the **Configuration** tab. Refer to Configuring Digital Inputs.

You can display configured event sources, subroutines attached to events, and verify the status of events using system bits and words.

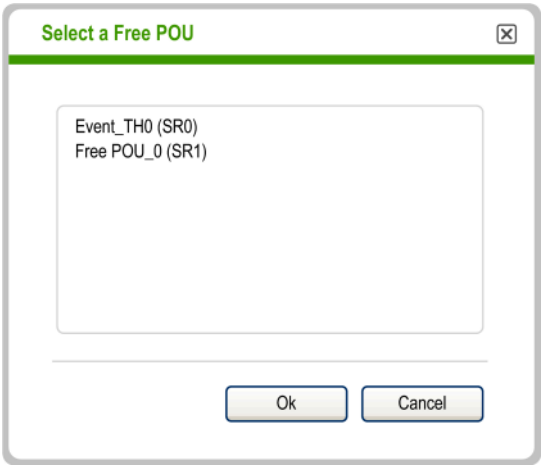
To display the event sources and subroutines (Free POUs) assigned to events:

| Step | Action |
|---|---|
| 1 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Select **Events**:<br>  <br>**NOTE:** Configured event sources that have not yet been assigned a subroutine appear in red. |

**NOTE:** Only embedded controller inputs/outputs can be used in an event subroutine.

## Assigning a Free POU to an Event Source

Proceed as follows to assign a Free POU to a configured event source:

| Step | Action |
|---|---|
| 1 | Create a new Free POU, page 65 containing the subroutine to use for the event. |
| 2 | Select the **Tasks** tab in the left-hand area of the **Programming** window. |
| 3 | Assign a subroutine to the event source by one of the following methods:<br>• Select the event source in the **Events** list and click  (**Assign Free POU** button) on the toolbar at the top of the **Tasks** tab.<br>• Right-click the event source in the **Events** list and choose **Assign Free POU** from the contextual menu that appears.<br>**Result**: The **Select a Free POU** window is displayed:<br>  <br>**NOTE:** You can directly add a Free POU to the event source. Right-click the event source in the **Events** list and choose **Add Free POU** from the contextual menu that appears. In this case, a Free POU is created and assigned to the event source. |
| 4 | Select a Free POU to assign to the event source and click **OK**.<br>**Result**: The selected subroutine is assigned to the event source and no longer avaialble in the **Free POU** branch of the **Tasks** tab.<br>For example, if the Free POU “Free POU_0” containing the subroutine `SR1` is assigned to the event source, the **Free POU_0 (%SR1)** subroutine moves from the **Free POU** branch to the event source branch of the **Tasks** tab. |

## Removing a Subroutine from an Event

To remove the association between a subroutine and an event source, follow these steps:

| Step | Action |
|---|---|
| 1 | Click the **Tasks** tab in the left-hand area of the **Programming** window. |
| 2 | Remove the subroutine from the event source by one of the following methods:<br>• Select the event source in the **Events** list and click  (**Unassign Free POU** button) on the toolbar at the top of the **Tasks** tab.<br>• Right-click the event source in the **Events** list and choose **Unassign Free POU** from the contextual menu that appears.<br>**Result**: The selected subroutine is removed from the event source and available as a Free POU in the **Free POUs** branch of the **Tasks** tab. |

## Verifying Events with System Bits and Words

The following system bits are used to verify the events:

| System Bit | Description |
|---|---|
| *%S38* | Used to enable (*%S38* = 1) or disable (*%S38* = 0) events processing. |
| *%S39* | Used to determine if events are lost. |

The following system words are used to check the events:

| System Word | Description |
|---|---|
| *%SW48* | The number of events that have been executed since the last cold start of the logic controller. |

The values of *%S39* and *%SW48* are reset to 0 and the value of system bit *%S38* is set to its initial state 1 following a cold restart or after an application is loaded. Their values remain unchanged after a warm restart.

# Using Tools

## Messages

### Overview

While editing the program, EcoStruxure Machine Expert - Basic analyzes the source code in the **Programming** tab.

EcoStruxure Machine Expert - Basic also analyzes the program whenever the **Compile** button  on the toolbar is clicked.

If errors or advisories are detected, a clickable icon is displayed in the **Programming** tab:

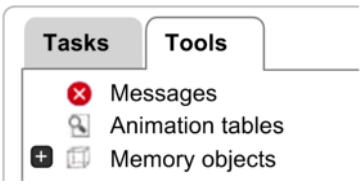

Clicking on this icon opens the Messages window.

The icon that is displayed depends on the message severity:

| Icon | Meaning |
|---|---|
|  | Advisory. The rung is incomplete. |
|  | A syntax error is detected. |

If both advisory and error messages are detected, only the Error icon  is displayed.

The icon is also displayed in the **Tools** tab next to **Messages**:



## Displaying Messages

To display a list of error and advisory messages:

| Step | Action |
|---|---|
| 1 | Click the icon on the **Programming** tab<br>or:<br>Click **Tools** > **Messages**<br>A list of messages is displayed in the lower central area of the **Programming** window. |
| 2 | In the **Messages** area, click the **Advisory** button to display advisory messages, or the **Error** button to display error messages. Click the button again to hide the list of messages. |

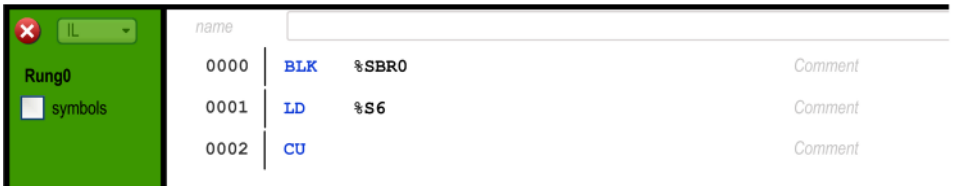
## Rung Status

EcoStruxure Machine Expert - Basic also displays the status of each rung in the program individually.

If the rung is syntactically valid and complete, there are no messages to display and a green tick symbol is displayed:


An Advisory icon appears if the rung is incomplete, for example, it does not contain a final instruction such as an *END*, *CALL*, or *Jump*:


An Error icon appears if EcoStruxure Machine Expert - Basic detects one or more syntax errors that would prevent successful compilation of the rung:




Advisory and Error icons are also displayed next to the name of each rung with errors in the **Tasks** tab:

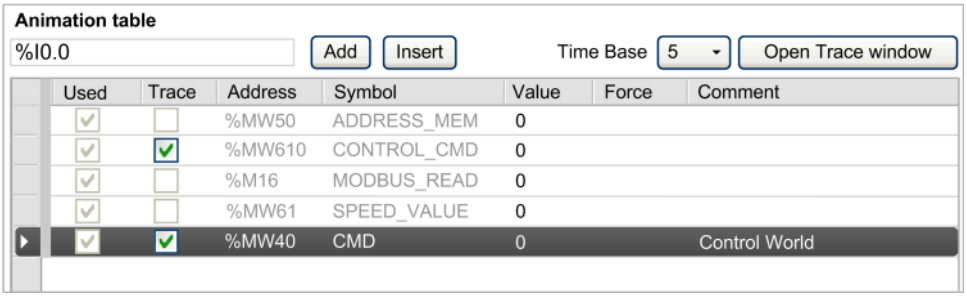
# Animation Tables

## Overview

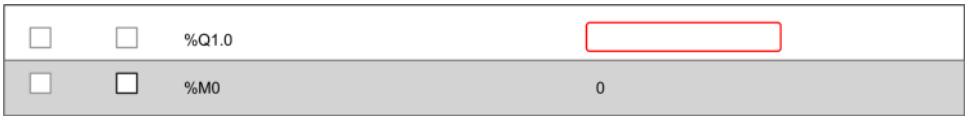
You can add objects to animation tables to:

- View symbols and comments associated with objects.
- View and modify the real-time values of certain object types when EcoStruxure Machine Expert - Basic is connected to the logic controller (online mode).
- Select objects to be displayed in the **Trace** window, page 149.

Animation tables are a component of an EcoStruxure Machine Expert - Basic application, and so are downloaded to the logic controller as part of the non-program data together with the program. This allows the objects stored in animation tables to be retrieved when an application is later uploaded from the logic controller.




If you add an object that does not exist to an animation table, the **Value** field is displayed with a red outline. For example, if you add %Q1.0 but there is no corresponding digital output module in the configuration.




## Creating an Animation Table

| Step | Action |
|---|---|
| 1 | Select the **Tools** tab in the left-hand area of the **Programming** window. |
| 2 | Right-click **Animation tables** and choose **Add new animation table** from the contextual menu that appears.<br><br>**Result**: A new animation table item appears below the **Animation Tables** area of the **Tools** window, and a properties window appears in the lower central area of the window. |

## Adding Individual Objects to an Animation Table

| Step | Action |
|---|---|
| 1 | Select the **Tools** tab in the left-hand area of the **Programming** window. |
| 2 | Select the animation table to configure in the **Animation tables** area of the **Tools** window.<br><br>**Result**: The properties window appears in the lower central area of the window. |
| 3 | To add a new object to the bottom of the animation table, type the object name into the text box and press Enter, or click **Add**.<br><br>The following objects can be added to an animation table:<br>• I/O objects<br>• Function block objects. For example, for a Timer function block `%TM0`, `%TM0.V`, `%TM0.P` and `%TM0.Q` are automatically added to the animation table.<br>• Bit strings (example: `%Mx:L` where `L` is the bit count, multiple of 8)<br>• Word tables (example: `%MWx:L` where `L` is the word count)<br>• Bits of words (example: `%MWx:X` where `X` is the offset of the bit)<br>• Network objects (`%QWE`, `%IWE`, `%QWM`, `%IWM`)<br>**NOTE:** Network objects are only available if either the EtherNet/IP adapter (see Modicon M221, Logic Controller, Programming Guide) of the logic controller is enabled, or **Modbus mapping** is enabled in the Modbus TCP Configuration. |
| 4 | To add a new object immediately above an existing object, select a row in the animation table, type the name of the object to add into the text box, and click **Insert**. |

Addresses of I/O objects in animation tables are not automatically modified following configuration changes. For example, `%Q3.0` is not automatically changed to `%Q1.0` when the position of the corresponding module changes in the configuration. You must take into account any adjustments made to I/O memory assignments within your application and update them accordingly.

**⚠WARNING**

**UNINTENDED EQUIPMENT OPERATION**

Inspect and modify as necessary any immediate I/O addresses used in the application after modifying the configuration.

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

Always verify and update animation tables following a configuration change.

## Adding All Objects Used in a Rung to the Animation Table

| Step | Action |
|---|---|
| 1 | If more than 1 animation table exists, select an animation table in the **Animation tables** area of the **Tools** window.<br><br>**Result**: The animation table properties window appears in the lower central area of the window. |
| 2 | Select the **Tasks** window. |
| 3 | Right-click a rung and choose **Add rung objects to the current animation table** from the contextual menu that appears.<br><br>**Result**: Objects used in the rung are added to the animation table. |

**NOTE:**

- The rung must contain no errors detected (error icon does not appear).
- Only the first 64 objects used in the rung are added (the maximum size of an animation table).
- If the same object appears more than once in a rung, only the first occurrence is added to the animation table.

## Animation Table Properties

This table describes the properties of the animation tables:

| Parameter | Editable | Value | Description |
|---|---|---|---|
| **Used** | No | True/False | Indicates whether the object is currently being used in a program. |
| **Trace** | Yes(1) | True/False | Select the object to trace in the Trace window, page 149. |
| **Address** | No | Object address | Displays the address of the object. |
| **Symbol** | No | A valid symbol | The name of the symbol associated with this object, if defined. |
| **Value** | Yes(2) | Current value | The value of the object.<br><br>If the object type has read/write access and you are in online mode, page 20, double-click and type a new object value if required. The value of the object is updated in real time in the program running in the logic controller.<br><br>See Modifying Real-Time Values, page 151 for details. |
| **Force** | Yes(2) | **Force to 0**<br><br>**Force to 1**<br><br>**Not Forced** | Only appears for digital inputs and digital outputs. Only editable when in online mode, page 20. Allows you to force the value of the input or output to 0 or 1 as required.<br><br>Choose **Not Forced** to remove any forcing currently applied to the address.<br><br>**NOTE:** Forcing is performed at the end of the scan cycle. The image table of the outputs, however, may be modified due to the logic of your program, and may appear in animation tables and other data displays contrary to the forced state you selected. At the end of the scan, this will be corrected by acting upon the requested forced state and the physical output will indeed reflect that forced state. |
| **Comment** | No | A valid comment | The comment associated with this object, if defined. |

**(1)** You can select up to 8 objects.

**(2)** Depending on the object type, and whether you are in online mode.

## Configuring Items in an Animation Table

To search for and optionally replace an object in an animation table, right-click the object and choose **Search and Replace**. Refer to Search and Replace, page 105 for further details.

To remove an object from an animation table, right-click the object and choose **Remove from animation table**.

## Copying/Cutting Existing Animation Tables

| Step | Action |
|---|---|
| 1 | Select the **Tools** tab in the left-hand area of the **Programming** window. |
| 2 | Select one or more animation tables in **Animation tables** by pressing and holding the CTRL key. |
| 3 | Right-click one of the selected animation tables in **Animation tables** and choose **Copy animation table** or **Cut animation table**. |
| 4 | To paste the animation table, either:<br>• Right-click **Animation tables** and choose **Paste animation table**.<br>• Right-click an existing animation table and choose **Paste animation table**.<br>**Result:** The **Confirmation** window appears. To keep the symbols and comments, clear the check box, then click **Ok**.<br>**Result:** One or more animation tables are added at the end of **Animation tables** or after the selected animation table.<br>When copying/pasting an animation table, EcoStruxure Machine Expert - Basic automatically assigns a new name. For example: **Animation_table_2** becomes **Animation_table_2_0**. |

When you paste an animation table to a project with a lower functional level, page 54, only the object configurations supported by this functional level are copied.

If the symbols contained in the pasted animation table are already used in the project, EcoStruxure Machine Expert - Basic replaces the pasted symbol.

## Deleting an Animation Table

| Step | Action |
|---|---|
| 1 | Right-click the animation table to delete in the **Animation tables** area of the **Tools** window and click **Delete animation table**. |

## Renaming an Animation Table

| Step | Action |
|---|---|
| 1 | Right-click the animation table to rename in the **Animation tables** area of the **Tools** window and click **Rename animation table**. |
| 2 | Type the new name of the animation table and press Enter. |

## Exporting Animation Tables

| Step | Action |
|---|---|
| 1 | Select the **Tools** tab in the left-hand area of the **Programming** window. |
| 2 | To select the animation table either:<br>• Right-click **Animation tables**.<br>• Select one or more existing animation tables by pressing and holding the CTRL key, then right-click. |
| 3 | Click **Export animation table**. |
| 4 | Choose a folder and save the animation tables (.csv). |

## Importing Animation Tables

| Step | Action |
|---|---|
| 1 | Select the **Tools** tab in the left-hand area of the **Programming** window. |
| 2 | To select the animation table either:<br>• Right-click **Animation tables**.<br>• Right-click an existing animation table. |
| 3 | Click **Import animation tables**. |
| 4 | Navigate to the folder containing the animation table file (*.csv). |
| 5 | Select the animation tables, then click **Open**.<br>**Result:** The animation tables are added at the end of **Animation tables** or before the selected existing animation table. |

If the symbols contained in the imported animation table are already used in the project, EcoStruxure Machine Expert - Basic replaces the imported symbol.

## Opening the Trace Window

| Step | Action |
|---|---|
| 1 | Select up to 8 objects in the **Trace** column of an animation table. |
| 2 | Connect, page 162 to the logic controller or launch the simulator, page 177. |
| 3 | Select a value in the **Time Base** list. This determines the refresh frequency of the Trace window, page 149, in seconds. |
| 4 | Click **Trace**.<br>The **Trace** window is displayed. |

# Memory Objects

## Overview

Memory objects include:

- Memory bits
- Memory words
- Constant words

## Selecting the Memory Allocation Mode

Before viewing or updating the properties of memory objects, choose the memory allocation mode, page 47 to use.

## Displaying the Used Memory Objects

To display the used memory objects, select the **Table view** check box:

| Color | Parameter | Description |
|---|---|---|
| Dark green | **Used** | Indicates whether the memory object is currently being used in a program. |
| Light green | **Equ Used** | Indicates whether part of the memory area of the memory object is currently being used. Refer to Possibility of Overlap Between Objects (see EcoStruxure Machine Expert - Basic, Generic Functions Library Guide). |
| Orange | **Used & Equ Used** | Indicates whether the memory object and a part of the memory area of the memory object is currently being used. |

## Memory Bit Properties

This table describes each parameter of the **Memory bits** screen:

| Parameter | Editable | Value | Default Value | Description |
|---|---|---|---|---|
| **Used** | No | True/False | False | Indicates whether the memory bit is currently being used in a program. |
| **Address** | No | Refer to Bit Objects | N/A | Displays the address of the memory bit, where *x* is the number of memory bits supported by the logic controller. |
| **Symbol** | Yes | A valid symbol | *None* | Allows you to associate a symbol with this memory bit. |
| **Value** | Yes | Refer to Bit Objects. | 0 | The value of this memory bit. |
| **Comment** | Yes | A valid comment | None | Allows you to associate a comment with this memory bit. |

# Memory Word Properties


First choose the memory word type to display properties for:

- **%MW** Memory words.
- **%MD** Double words.
- **%MF** Floating-point words.

This table describes the properties of **Memory words**:

| Parameter | Editable | Value | Default Value | Description |
|---|---|---|---|---|
| **Used** | No | True/False | False | Indicates whether the memory word is currently being used in a program. |
| **Equ Used** | No | True/False | False | Equivalent used. Indicates whether part of the memory area of the memory word is currently being used. Refer to Possibility of Overlap Between Objects (see EcoStruxure Machine Expert - Basic, Generic Functions Library Guide). |
| **Address** | No | Refer to Word Objects (see EcoStruxure Machine Expert - Basic, Generic Functions Library Guide) | N/A | Displays the address of the memory word. |
| **Symbol** | Yes | A valid symbol | *None* | Allows you to associate a symbol with this memory word. |
| **Value** | Yes | Refer to Word Objects (see EcoStruxure Machine Expert - Basic, Generic Functions Library Guide). | 0 | The value of this memory word. |
| **Comment** | Yes | A valid comment | None | Allows you to associate a comment with this memory word. |