MACHINELOGIC

Runtime Reference Guide

CTC Parker Automation

MACHINELOGIC

Runtime Reference Guide

CTC Parker Automation *Phone: 513-831-2340*
50 W. TechneCenter Drive, Milford, Ohio 45150 *Technical Support: 513-248-1714*

A3-05796-103

Copyright and Trademark Notice

Contents

Introduction

This chapter provides information about:

- ☐ MachineLogic Runtime
- ☐ MachineLogic Development
- ☐ The documentation for MachineLogic Runtime
- ☐ Conventions used in this manual

Introduction

What is MachineLogic Runtime?

The MachineLogic Runtime kernel is a highly efficient runtime system for complex control applications. It has been especially designed for IEC 1131-3 and includes nearly the full range of IEC 1131-3 features. MachineLogic runs on the MachineLogic Slot Card (MLSC). One of these cards must be present in your runtime workstation for MachineLogic to run.

MachineLogic Runtime uses an efficient standard real-time multitasking system. MachineLogic Runtime, in conjunction with the real-time multitasking system, offers the possibility of using a CTC PowerStation or any industrial PC for automation tasks such as measuring, controlling, regulation, visualization, etc.

MachineLogic Runtime has an open interface which allows it to adapt to the InteractX or third-party operator interface software package. Because of its open I/O interface, MachineLogic Runtime can be used with decentralized I/O and fieldbusses like DeviceNet and Profibus.

MachineLogic Runtime operates in a multitasking kernel. This multitasking kernel allows MachineLogic Runtime to run independently of the PC's CPU.

MLSC

The MLSC is a hardware card installed inside your runtime workstation. The MLSC has its own processor on the card itself where MachineLogic resides.

MachineLogic runs on the MLSC processor while your application runs on the PowerStation or industrial PC's processor. The MLSC does not have to share a processor on the runtime workstation to execute tasks.

The MLSC also has an Ethernet port built onto the card itself.

MachineLogic Runtime and MachineLogic Development

MachineLogic Runtime is delivered in conjunction with the IEC 1131-3 programming standard. MachineLogic allows easy programming in SFC, FBD, LD, IL and ST under Windows 95/98/2000/NT/ME. From the users' point of view MachineLogic Runtime is the tool working in the background. All essential operations are done via the graphical user interface of MachineLogic including downloading and online debugging. Nevertheless it is important to know something about the structure of the runtime system to configure MachineLogic and the control program in the optimal way.

What kind of documentation is provided for MachineLogic Runtime?

The documentation for MachineLogic Runtime consists of several parts. These parts are dedicated to different users and purposes. For an understanding of all parts we are assuming knowledge about using MS-Windows.

This reference guide provides all background information for a better understanding of the concepts of MachineLogic Runtime and of the operations to be done. This manual gives you information on how to proceed for structuring your MachineLogic project.

The context-sensitive help for MachineLogic Runtime can be viewed within MachineLogic by pressing F1. The online help provides detailed reference information along with basic information on the product.

Who should read this manual and how is it structured?

This manual is dedicated to all users who want to learn about the concepts on which MachineLogic Runtime is based for better structuring their application task. The manual consists of the following chapters:

- Getting familiar with MachineLogic Runtime: Provides an overview of the MachineLogic Runtime and its features, technical data, and performance.
- The MachineLogic Runtime memory: Describes the structure of memory and where the different data are stored.
- Inputs, outputs, and flags: Describes how to handle input and output signals and how to write the I/O configuration.
- MachineLogic projects and MachineLogic Runtime: Describes resources, tasks, and pre-emptive scheduling.
- Downloading, controlling, and debugging: Provides information about the different control states, downloading, and debugging.

Symbols and textual conventions

The following symbols are used in this manual:

*	is used for enumeration.
•	is used for an operation which has to be done.
▫	is used for an operation which is optional.
🖱	is used for a sequence of operations to be done with the mouse.
⌨	is used for a sequence of operations to be done with the keyboard.
☝	is used to provide important information.
📖	is used to introduce references to other documents.

The following textual conventions have been set up for this manual:

'	commas are used for names of icons, menu items or names of objects e.g. menu item 'Cut'; function block 'Level'.

<ALT>	brackets are used for the name of keys on your keyboard and for words you have to enter.
<ALT> + <F4>	is used if you have to press two keys at the same time.
driver name	Italic letters are used as place holders for proper names.

Getting familiar with MachineLogic Runtime

This chapter provides information about:

- ☐ The MachineLogic development system and MachineLogic Runtime
- ☐ The different parts of MachineLogic Runtime
- ☐ Platforms andTechnical data and system requirements
- ☐ IEC 1131-3 compliance

Getting familiar with MachineLogic Runtime

The MachineLogic development system and MachineLogic Runtime

The development procedure of a MachineLogic project and how MachineLogic Runtime controls the process appears below:

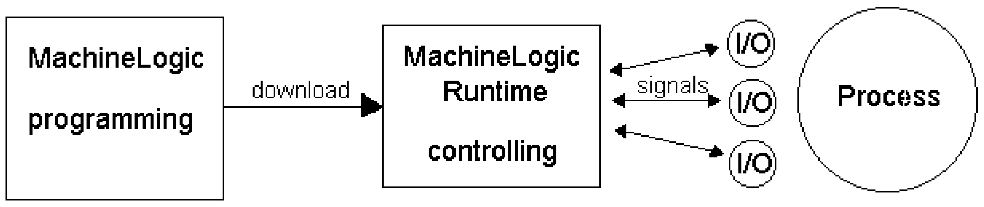


Figure 2-1: Components of the development system

The MachineLogic project is first programmed in MachineLogic and then it is downloaded to MachineLogic Runtime where it is stored in the MachineLogic Runtime memory. After downloading, the control program can be started. The process signals are transmitted to MachineLogic Runtime via the local or decentralized I/O system and drivers. MachineLogic Runtime stores the input signals coming from the process in its memory. During a runtime cycle, the control program reads the inputs, executes the program logic, and then writes the outputs. Using the driver the outputs are transmitted from the MachineLogic Runtime memory to the I/Os and the process.

Parts of MachineLogic Runtime

MachineLogic Runtime consists of several parts as shown in the following figure.

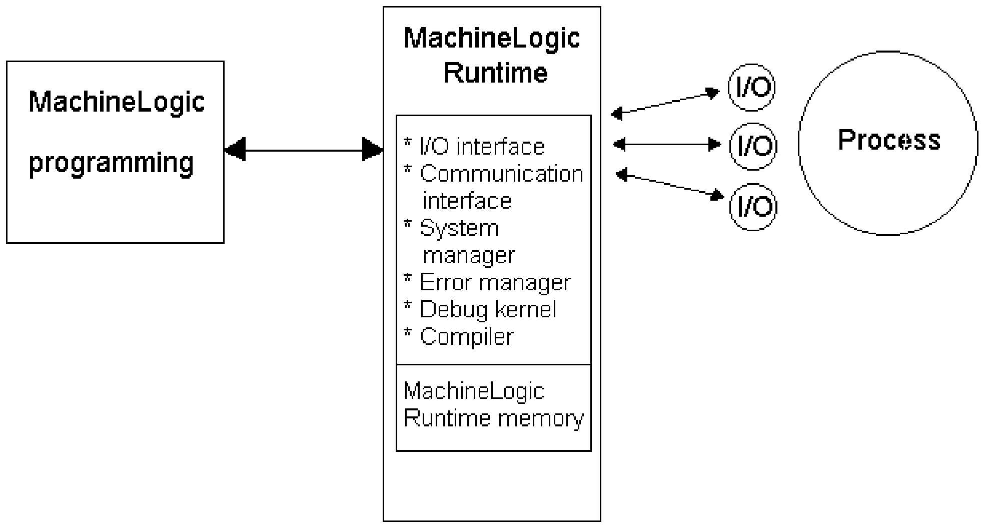


Figure 2-2: Parts of MachineLogic Runtime

I/O interface

The I/O interface is used to communicate with any I/O system. A decentralized I/O system can be used such as a fieldbus like DeviceNet or Profibus. The I/O interface uses the driver, such as Ethernet Open Modbus (ENOMB), DeviceNet, or Profibus, to pass the signals between the I/O systems and the memory for signal data.

Communication interface

Via the communication interface all different types of communication protocols between MachineLogic and MachineLogic Runtime are handled. Such communication protocols include downloading the project, debugging, and transmitting error messages from MachineLogic Runtime to MachineLogic. You can download using different protocols.

System manager

MachineLogic Runtime includes a system manager that executes the different parts of the control program. The system manager handles different tasks and coordinates the execution of programs.

Error manager

The error manager is able to detect and to handle several types of errors that may occur during program execution. If an error occurs during online communication with MachineLogic, an error message is displayed in the error catalog of MachineLogic. The user can view the error message from the control dialog.

In case of critical errors like exceeding the fixed time interval for the execution of a program, the error manager activates a system task. In those cases the system program associated with the task is executed. If there are no system tasks declared in MachineLogic, program execution stops.

Debug-Kernel

The debug kernel of MachineLogic Runtime offers multiple debug functions such as overwriting and forcing variables. Debugging is normally done while the process is running. Nevertheless it is also possible to correct programming errors while the process is stopped by setting breakpoints. All debug operations are executed via the MachineLogic user interface.

Compiler for control program

In MachineLogic the compiling process leaves the program in a control intermediate code. The MachineLogic Runtime compiler translates the control intermediate code into machine-executable code.

MLSC Control Panel

The MachineLogic Slot Card (MLSC) Control Panel is the primary user interface for runtime workstations running the Slot Card on a Windows machine. The Control Panel interfaces with MachineLogic and provides essential communication functions between a development system running the MachineLogic and a runtime workstation, such as a PowerStation, with the MLSC installed. These communication functions include transferring project files as well as creating and restoring remote backups.

The Control Panel displays detailed information about the runtime workstation and allows you to monitor and configure many of the workstation's hardware and software settings.

The purpose of the Control Panel is to simplify navigation and configuration of the runtime workstation from a central source. It is designed to reduce the number of steps to perform a task and to minimize the complexity of the system.

For more information, see the online Help that comes with the Control Panel.

Technical data

For organizing your control program it is useful to know some of the technical data relevant for MachineLogic Runtime. The technical data sometimes differs from hardware to hardware.

Characteristic	Technical data
Speed	80486 CPU at 33 MHz: 1000 Boolean global variables: 0.38 ms 1000 BOOL8 and INT global variables: 0.15 ms Pentium CPU at 90 MHz: 1000 Boolean global variables: 0.1 ms 1000 BOOL8 and INT global variables: 0.02 ms
Shortest cycle period	configurable, standard 1 ms
Program memory capacity per POU	configurable, approx. 1000 instructions need 10 Kbyte RAM, in one POU 64 Kbyte program code can be executed.
I/O image capacity	• **MLSC**: limited to the actual memory available in the system
Data memory capacity	• **MLSC**: limited to the actual memory available in the system
Number of timers and counters	unlimited, depends on the program memory
Number of user tasks	16 tasks

Compliance with IEC 1131-3

This reference guide has an appendix that contains a complete list of IEC compliance. In the general help for MachineLogic and in the MachineLogic Reference Guide there is an introductory chapter on IEC 1131-3. Please refer to this documentation for detailed information.

The restrictions that apply to MachineLogic Runtime are described in the following sections.

Restrictions for data types

- Elementary data types: LINT, LREAL, TIME-OF-DAY, DATE-AND-TIME, and LWORD are not yet implemented in MachineLogic Runtime.
- User-defined data types can only be used within function blocks and programs. It is not possible to use them in functions.
- User-defined data types cannot be initialized.
- User-defined data types cannot be forced or overwritten. Exception: in the watchlist multi-element, data types can be forced and overwritten.
- While debugging arrays and structures the variables have to be copied into the watchlist. Copying these variables into the watchlist can only be done in the variable worksheet and not in the code body worksheet.
- Alias and subrange data types are not yet available.
- Multi-dimensional arrays cannot be declared. But it is possible to declare arrays of arrays as shown in the following example:

Type declaration:
```
TYPE
        graph           :           ARRAY [1..10] OF INT;
        my array        :           ARRAY [1..3] OF graph;
END_TYPE
```

Variable declaration:
```
VAR
        var1            :           my_array;
        var2            :           INT;
END_VAR
```

Code body declaration in ST:
```
var2                := var1[1] [3];
```

- While using direct variables for arrays or structures of the data type BOOL on inputs or outputs, an error message is displayed.
- Only the data type 'INT' can be used as an array index. Expressions like 'i + 1' are not possible.
- In arrays the data type BOOL corresponds to one byte.
- While using direct variables for arrays and structures it is up to the application programmer to check if the logical addresses really exist. No check is done in MachineLogic.
- It is not possible to declare an array within the declaration of a structure. For example:

```
TYPE
        machine                    :
        STRUCT
                x_pos              :           INT;
                performance        :           ARRAY[1..23] OF INT;
        END_STRUCT;
END_TYPE
```

However it is possible to declare it as shown in the following example:

```
TYPE
        perform_def                :           ARRAY[1..23] OF INT;
        machine                    :           STRUCT
                x_pos              :           INT;
                performance        :           perform_def;
        END_STRUCT;
END_TYPE
```

You have to declare each data type in a data type declaration using TYPE ... END_TYPE.

Restrictions for strings

- The data type STRING always has a size of 80 characters. If your variable of the data type STRING has four letters, the string needs 85 bytes in the MachineLogic Runtime memory. But it is possible to declare a new data type with a user defined string length as shown in the following example:

Type declaration:

```
TYPE
        my_string_1     :           STRING(10);
        my_string_2     :           STRING(20);
END_TYPE
```

Variable declaration:

```
VAR
        var1            :           my_string_1;
        var2            :           my_string_2;
END_VAR
```

Code body in IL:

```
LD   var1
ST   var2
```

If you declare several strings of different bit sizes they are all treated as the same data type. Therefore the code body in IL is correct.

- String functions and function blocks cannot be used in networks. You can only use single functions and function blocks storing the result to variables.
- Strings cannot be nested while editing in ST.
- Strings cannot be used in function POUs.
- While debugging strings in online mode it might be useful to change the width of the column for online values choosing the menu item 'Value width' in the submenu 'Online'.

Restrictions for ST

- In IF statements it is possible to compare Boolean variables. For example (var1 being a Boolean variable):
  ```
  IF var1=TRUE THEN ...;
  ```
  However, it is recommended you use another representation because it is faster in execution and compilation and less code is generated.
  Use the following representation: IF var1 THEN ....;
- In a FOR statement it can only be counted upwards. Therefore the increment value after the keyword BY must always be positive.
- In a FOR statement the variable 'a' and the increment value can be changed within the body of the statement. No error message or warning is displayed if you enter the following example:
  ```
  Example:          FOR i=0 TO 10 BY5
                    DO i:=10;
  ```
  Be very careful using such a statement structure.
- In a function call no assignments can be used. The following example is not possible: a:= SIN(b:=3);
- In an expression all operations are processed. Parts of an expression which have no impact on the result are also processed. In the following example c>d is also processed if a<b is FALSE:
  ```
  Example:          a:=a<b AND c>d;
  ```

MachineLogic Runtime Memory

This chapter provides information about:

- ☐ The general structure of the MachineLogic Runtime memory
- ☐ The different parts of the memory
- ☐ Variables and flags

MachineLogic Runtime Memory

Memory Guidelines

With the MLSC, MachineLogic Runtime does not require a specific memory area of the runtime system. All memory is dynamically allocated by the system. The HMA is not divided.

Variable declarations in MachineLogic

In MachineLogic you can either use non-direct or direct variables. In the following sections the method used to declare these variables in MachineLogic is described. Hints for structuring the variable declarations are also provided.

Non-direct variables

Non-direct variables are variables whose declaration consists of a name and a data type. You may also assign an initial value for initializing the variable. An example showing the declaration of a non-direct variable is shown in the following figure.

```
VAR
        var1     :        BOOL;
END_VAR
```

Figure 3-3: Declaration of a non-direct variable

In the example the name of the variable is 'var1' and its data type is BOOL. Configuring such a variable declaration in MachineLogic means that the variable is stored somewhere in memory. The logical memory address is not known by the application programmer.

All required variables of a temporary nature should be declared as non-directed variables. This is useful because you do not need to consider the actual memory addresses. It will be automatically assigned when you compile your project.

Direct variables

Direct variables are variables whose declarations consist of a name, the location, and the data type. You may also assign an initial value for initializing the variable. All variables defined in I and Q memory must be directed variables. An example showing the declaration of three direct variables is shown in the following figure.

```
VAR
        var1    AT %IX2.2     :     BOOL;
        var2    AT %QX2.2     :     BOOL;
        var3    AT %MX2.2     :     BOOL;
END_VAR
```

Figure 3-4: Declaration of direct variables

In the example the names of the variables are 'var1', 'var2' and 'var3' and their data types are BOOL. The location is declared using the IEC 1131-3 keyword 'AT' followed by '%'. 'I' refers to the Input region, 'Q' refers to the Output region, and 'M' refers to the Variable (M) region of the memory area. The prefix 'X' represents the bit size.

The variable 'var1' is stored in the MachineLogic Runtime memory area 'I' at the logical address '2.2' and with a size of one bit. The logical address and its bit size are set by the application programmer.

The logical addresses which can be used for 'I' or 'Q' variables are determined by the input and output devices (e.g. fieldbus) you are using. The addresses must correspond with the areas fixed in the I/O configuration of MachineLogic.

The logical addresses that can be used for 'M' variables, also called flags, are determined by the settings of the user areas in the dialog titled 'Data Area' within MachineLogic.

The memory 'M' where 'var3' of the example is stored is separated into two parts. One part is used for all direct variables of the type 'M'. This part is called the user part of memory 'M'. The other part is used for all non-direct variables. These non-direct variables are automatically stored by MachineLogic in free memory areas as mentioned in the previous chapter.

If direct 'M' variables are used, the checkbox 'Declare user memory in I/O configuration automatically' located on the dialog 'Data Area' should be checked.

To access an M variable in Interact via the MachineLogic Device Driver, you must define the variable as a directed variable.

☝ For logical addresses please refer to the chapter 'Configuring MachineLogic Runtime'.

☝ Be very careful using direct variables. It is up to you to check that no address is used twice. No error message is displayed in MachineLogic if you are using an address twice!

☝ Use only logical addresses of the user part of memory area 'M' while declaring direct variables of type 'M'! This is the only way to ensure data consistency!

☝ It might be useful to declare all inputs and outputs as global variables. This means they have to be declared as direct variables in the global variable declaration using the keyword 'VAR_GLOBAL'. In the 'VAR_EXTERNAL' declaration of programs and function blocks they are declared as non-direct variables. If the memory addresses change, you have to change them only in the global variable declaration and leave all other declarations unchanged.

Retentive variables

In MachineLogic it is also possible to declare retentive variables. It depends on your hardware memory configuration whether it is possible to store retentive data.

The memory for retentive data is located in non-volatile memory which retains the data when turning the system off or in case of a power failure. Retentive variables are declared in MachineLogic using the keyword VAR RETAIN in the corresponding variable declaration. For example:

```
VAR RETAIN
    var1                          :   BOOL;
    var2        AT %MX36243.0     :   BOOL;
END_VAR
```

Figure 3-5: Declaration of a non-direct retentive variable and a direct retentive variable

System flags

System flags provide information about the internal state of MachineLogic Runtime during runtime. The system flags have fixed memory addresses and can be used as non-direct variables in MachineLogic. All available system flags and their names are explained in the chapter 'Downloading, controlling and debugging'.

Shared memory firmware flags

MachineLogic reserves a section of the HMA, called the firmware flags, to share common data with Interact These shared memory flags can be addressed using the optional flag selector '3'. The following figure shows an example how to address a shared memory flag:

```
VAR
    var1        AT %MX3.1.1       :   BOOL;
END_VAR
```

Figure 3-6: Addressing a shared memory flag

The logical address consists of the flag selector, the addressed byte, and the addressed bit.

How to use variables in MachineLogic

- Declare global variables in the global variable declaration in the subtree 'Physical Hardware' of the project tree.
- Declare global variables as VAR_EXTERNAL in the variable declaration of the POU where you want to use them.
- Declare local variables in the variable declarations of the POU.
- Insert the variables in the corresponding code body worksheets.

Refer to the MachineLogic help for detailed information.

Configuring MachineLogic Runtime

This chapter provides information about:

- ☐ Inputs and outputs
- ☐ Logical memory addresses
- ☐ How to setup the I/O configuration in MachineLogic
- ☐ How to setup the data configuration

Configuring MachineLogic Runtime

Inputs, outputs and the process

While the MachineLogic project is running, the control program receives input signals via the I/O modules. These input signals are processed according to your settings in the control program. After the working cycle the control program transmits the output signals to the I/O modules.

The way the control handles these signals is set up by the user via the I/O configuration in MachineLogic. Depending on which version of MachineLogic you're running, you set up the I/O configuration differently. The method used to setup the I/O configuration is also determined by the control project and depends on the specific implementation of the I/O driver. This chapter explains the basic principles on configuration of data and I/O.

For more information see the documentation that came with MachineLogic.

☝ To determine which version of MachineLogic you're running, select the Help menu and then select About.

Setting up the I/O configuration in MachineLogic 2.11

The I/O configuration in MachineLogic 2.11 is set up using the I/O Configuration dialog boxes. Double-click IO_Configuration in the left pane (located within the subtree 'Physical Hardware' of the project tree) to launch the I/O Configuration dialog boxes.

To set up the I/O configuration you must declare the I/O module groups.

Declaring the I/O groups

Select the Input tab for input I/O groups, or select the Output tab for output I/O groups. Click the Add button to launch the Add I/O Group dialog box.

Enter a module name for the I/O group in the Name field. The module name is not automatically identical to the module identifier assigned by the I/O manufacturer. It is just the structure of the I/O configuration. According to your application task it may be useful to declare each I/O group as one I/O module in MachineLogic 2.11.

You must also associate the I/O group to a task. Select from the Task list box.

☝ You must always associate a program to a task. Associating a task ensures that the inputs are always read at the beginning and the outputs are always written at the end of the associated task and that the data which are processed are consistent.

Declaring the logical addresses

Declare the logical addresses using the Add I/O Group dialog box. Enter the starting address in the Start Address field. The starting address is in bytes of memory.

Enter the number of bytes to reserve for the I/O group address in the Length field.

☝ It might be useful to leave some memory addresses free among the different I/O modules. If you have to extend your I/O configuration later, it will be possible to use these free addresses and there will be no need to change the whole I/O configuration.

Declaring the I/O driver

Declare the I/O driver using the Add I/O Group dialog box. Select your driver from the Board / IO Module field and click the Driver Parameter button.

Using a driver automatically means that the data are read at the beginning of a working cycle of the task and that they are written to the I/O at the end of the working cycle. To use a driver you have to add parameters to your I/O configuration.

📖 Please refer to your driver online help or documentation for more information on setting up the driver.

Setting up the I/O configuration in earlier versions of MachineLogic

The I/O configuration in earlier versions of MachineLogic is set up using the I/O configuration editor. The I/O configuration editor is called by double-clicking the icon 'I/O configuration' located within the subtree 'Physical Hardware' of the project tree. For all versions of MachineLogic earlier than 2.11, you edit the I/O configuration by just entering the appropriate lines in the right pane.

To set up the I/O configuration you must declare the I/O module groups.

Declaring the I/O groups

In the first line of an I/O configuration the I/O group is introduced. According to your application task it may be useful to declare each I/O group as one I/O module in MachineLogic.

In the I/O configuration an I/O module has to be declared with the keyword 'PROGRAM'. For example:

```
Syntax:
PROGRAM      module name   WITH       task name    :    module type

Example:
PROGRAM      module_1      WITH       fast_task    :    INPUT
```

Figure 4-2: Declaring the I/O group

The module name can be chosen by the user. The module name is not automatically identical to the module identifier assigned by the I/O manufacturer. It is just the structure of the I/O configuration.

You MUST associate a program to a task. There is no Default task within MachineLogic Runtime.

☝ You must always associate a program to a task. Associating a task ensures that the inputs are always read at the beginning and the outputs are always written at the end of the associated task and that the data which are processed are consistent.

For input modules the module type 'INPUT' and for output modules the module type 'OUTPUT' have to be used.

Declaring the logical addresses

In the second line the logical addresses for the I/O groups are declared. This means that bytes of the inputs or outputs of the I/O module are assigned to a memory region in the MachineLogic Runtime memory. An example is shown in the following figure.

```
Syntax:
VAR_ADR          :=      value,
END_VAR_ADR      :=      value,

Example:
VAR_ADR          :=      0,
END_VAR_ADR      :=      3;
```

Figure 4-3: Declaring the memory addresses for the I/O group

The line beginning with the keyword VAR_ADR indicates the first memory address where the first input or output byte of the I/O module is stored. END_VAR_ADR specifies the last memory address for this I/O module.

☝ It might be useful to leave some memory addresses free among the different I/O modules. If you have to extend your I/O configuration later, it will be possible to use these free addresses and there will be no need to change the whole I/O configuration.

Declaring the I/O driver

In the following lines of the I/O configuration more parameters have to be used for declaring the I/O driver.

Using a driver automatically means that the data are read at the beginning of a working cycle of the task and that they are written to the I/O at the end of the working cycle. To use a driver you have to add the following parameters to your I/O configuration.

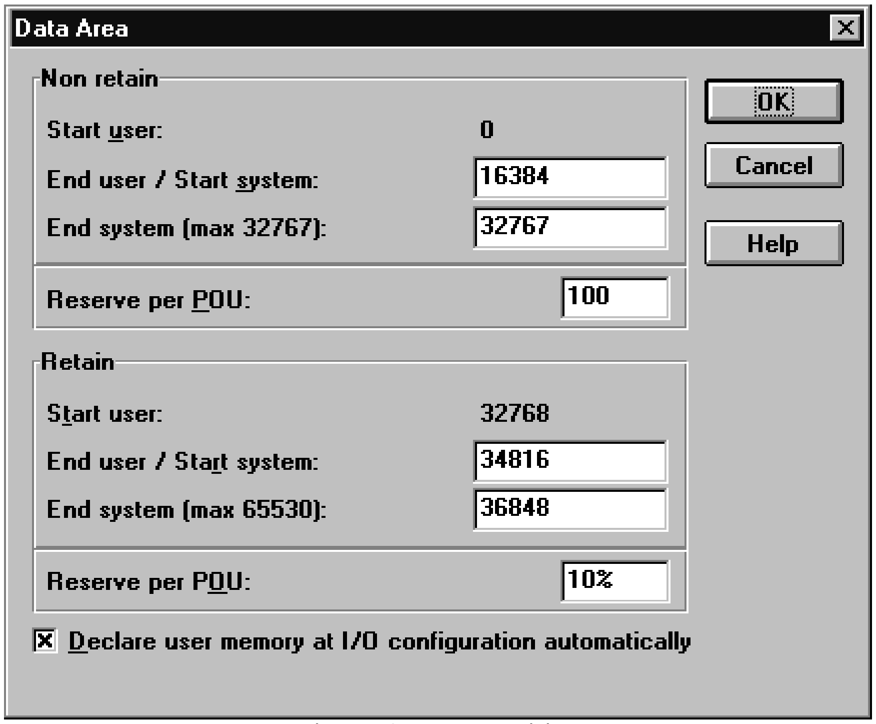
```
DEVICE           :=     DRIVER,
DRIVER_NAME      :=     'driver name',
DRIVER_PAR1      :=     parameter,       (*optional*)
DRIVER_PAR2      :=     parameter,       (*optional*)
DRIVER_PAR3      :=     parameter,       (*optional*)
DRIVER_PAR4      :=     parameter,       (*optional*)
```

Figure 4-4: Declaring the I/O driver

Please refer to your driver online help or documentation for the driver name and the parameters to include.

Setting up the data configuration

The Data Area dialog box:


Data Area

Non retain

Start user:	0
End user / Start system:	16384
End system (max 32767):	32767
Reserve per POU:	100

Retain

Start user:	32768
End user / Start system:	34816
End system (max 65530):	36848
Reserve per POU:	10%

[✓] Declare user memory at I/O configuration automatically

Figure 4-2: Data Area Dialog

Keep the *Declare user memory at I/O configuration automatically* check box checked so that the I/O configuration is updated automatically whenever you change the Direct Non-retain memory region in the Data Area dialog box.

Configuration of the following memory areas is accomplished through the data configuration. Memory for non-retentive and retentive user flags (directly represented or direct 'M' variables) and for automatically direct MachineLogic flags being non-retentive and retentive (symbolic variables) can be configured. This should be only done if you want to use directly represented or direct 'M' variables (flags) in MachineLogic (e.g. 'MYVAR AT %MX0.0 : BOOL;') or if the memory for non-direct variables has been used up.

The settings of the data areas are located on the dialog 'Data Area' of MachineLogic . The dialog 'Data Area' appears when clicking the button 'Data area...' in the dialog 'Resource settings...'. The dialog fields and buttons have the following meanings:

- Start user: Specifies the address where the memory area for direct addressing starts.
- End user / Start system: Specifies the address where the memory area for non-direct addressing starts.

• End system:	Specifies the address where the memory area for non-direct addressing ends.
• Reserve per POU:	Specifies the memory reserve for a POU used for Patch POU. The reserve can be entered in % or in bytes.

The checkbox 'Declare user memory in I/O configuration automatically' automatically creates a hidden entry (VAR_CONF) in the I/O configuration containing the memory areas set in this dialog. The activation of the checkbox is required if user flags are used.

MachineLogic does an automatic calculation of the data areas that are necessary to store the MachineLogic non-direct variables, including the memory reserve. The advantage is that only the specific memory of the MachineLogic project is allocated to the MachineLogic Runtime target. This automatic calculation is not done for directed variable areas. Therefore the full size of the configured directed variable areas is allocated if the already mentioned checkbox has been activated. To obtain information about the project size and the required memory space for non-directed variables click on the sheet tab 'Infos' in the message window to display this information after compiling.

☝ Please note that MachineLogic does not check any overlaps of the directed variables logical addresses with the MachineLogic non-directed variable addresses.

☝ When downloading, MachineLogic Runtime checks all variables and their logical addresses being used in a control program. If a program contains variables having logical addresses not matching a configured data area, the following error message occurs in MachineLogic Runtime: 'Operand not implemented or area exceeded at ...'

The settings in the dialog 'Data Area' and the automatic calculation of the data areas for MachineLogic variables result in a hidden entry (VAR_CONF) in the I/O configuration in MachineLogic. The maximum data areas are limited according to the settings in the dialog and the specific limits of the control type specified by the file 'LIMITS.REF'. The beginning of the user memory areas in the dialog also depends on the file 'LIMITS.REF'.

A fixed syntax is used for the data configuration. The default address range for user flags varies according to your hardware configuration.

Control Applications and MachineLogic Runtime

This chapter provides information about:

- ☐ Configurations and resources
- ☐ Tasks
- ☐ Error handling in MachineLogic Runtime and SPGs
- ☐ Programs
- ☐ Preemptive scheduling

Control Applications and MachineLogic Runtime

The control application in the MachineLogic Runtime memory

A control application is structured in the MachineLogic Runtime memory as shown in the following figure.

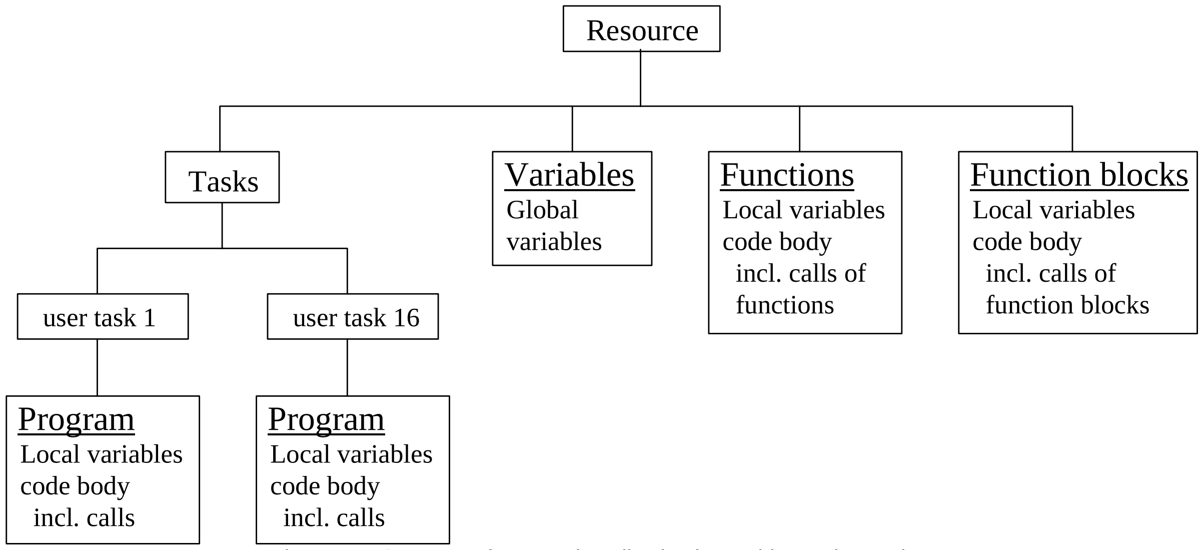


Figure 5-1: Structure of a control application in MachineLogic Runtime

Configurations and resources

According to IEC 1131-3 a control system consists of several configuration elements. The configuration element can be compared to a complete control program since it includes all elements with some I/O. A configuration contains one or several resources. A resource corresponds to an application for a control. It can be compared to a CPU.

A resource contains the task definitions and the definitions of all programs, function blocks, functions and variables. MachineLogic Runtime is able to handle several tasks within one resource. In MachineLogic Runtime the Programmable Controller System configuration and the resource have the same functionality. Nevertheless it is necessary to declare one configuration and one resource in the subtree 'Physical Hardware' of the project tree.

In MachineLogic, the Resource folder is found in the path Project\Physical Hardware\Configuration in the left pane.

How to declare resources in MachineLogic

- Insert the resource into the subtree 'Physical Hardware' of the project tree.
- Choose the resource settings in the dialog 'Resource settings...'. To display this dialog, right-click the Resource folder and select 'Settings...'.

You can select between 64K and 128K in the Resource Settings. Refer to the MachineLogic help for detailed information about inserting configurations, resources, and resource settings.

Tasks and MachineLogic Runtime

With the MLSC, MachineLogic Runtime is executed on the slot card itself. The execution is also controlled by task priorities. Four task levels are used in MachineLogic Runtime in addition to Interact:

- TCP/IP execution
- Supervisor task level
- User task level
- MachineLogic Runtime task level
- Interact

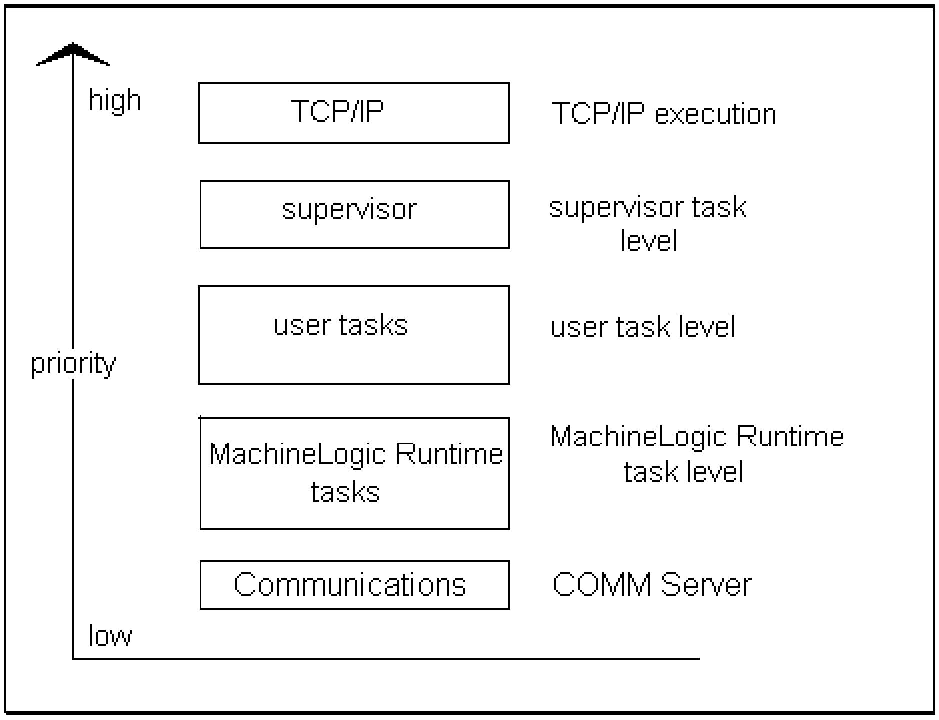


Figure 5-3: MLSC task levels in MachineLogic Runtime

Runtime and task description

MachineLogic Runtime itself consists of several tasks such as communication task, debug task, memory management task, and system control task. All these tasks are running in a specific priority level called MachineLogic Runtime task level.

User tasks are all tasks specified by the application programmer that rely on the IEC 1131-3 programming languages. User tasks also rely on the execution control of the programs under the control of tasks. These kinds of tasks run in the specific user task level. This means that all user-defined priorities are not absolute but relative to the user task level. The settings for these tasks are configured in MachineLogic. Interact does not belong to any user task and, thus, only executes when no other MachineLogic task is executing.

In MachineLogic Runtime there is a special protected task level: the supervisor level. The supervisor task detects errors such as division by zero or excess of execution time and activates the corresponding system task.

User tasks

User tasks have to be declared by the programmer. In MachineLogic Runtime several user tasks can be used. The different task types offer different characteristics for different application purposes.

Cyclic tasks

Cyclic tasks are executed in a fixed time interval. They are executed according to their priority, which is set in the dialog 'Task settings...'. This dialog is available by selecting the menu item 'Settings...' located in the context menu of the task in the subtree 'Physical Hardware' of the project tree. The priority value range is from 0 to 31 where 0 is the highest priority.

If the watchdog time of a cyclic task is higher than the configured interval time and the execution of the task is not finished before the end of the configured interval time is reached, one or more execution cycles are skipped.

System tasks

System tasks can be used in MachineLogic Runtime and MachineLogic. The system tasks are initial tasks and error tasks. System tasks are executed if the control program is started or if an error occurs. In both cases MachineLogic Runtime knows which system program has to be executed.

☝ System tasks are not supervised by the watchdog because they are not executed in the context of a user task.

System programs or SPGs are programs which are called automatically by MachineLogic Runtime if an error or a change of the operational state of the control program occurs. Different types of system programs are available as shown in the following table:

No.	Name	Meaning	Actions
SPG 0	WARM_START	is executed if a warm start is done	• retentive data are not initialized • non-retentive data are initialized • the open function of the I/O driver is executed • user tasks are activated • control passes into the 'RUN' state
SPG 1	COLD_START	is executed if a cold start is done	• all data are initialized • the open function of the I/O driver is executed • user tasks are activated • control passes into the 'RUN' state
SPG 2	TO_STOP	is executed if the program execution is stopped	• user tasks are deactivated • all outputs are updated • the close function of the I/O driver is executed • control passes into the 'STOP' state
SPG 10	WATCHDOG	is executed if the execution of a task has not been finished within its watchdog time	• user tasks are deactivated • all outputs are updated • the close function of the I/O driver is executed • control passes into the 'STOP' state
SPG 11	ZERODIV	is executed if a division by zero occurs during the program execution	• user tasks are deactivated • all outputs are updated • the close function of the I/O driver is executed • control passes into the 'STOP' state
SPG 12	STACKOVER	is executed if a stack overflow happens. Only executed if the check box 'Stack check' has been activated in the dialog 'Resource settings...' in MachineLogic	• user tasks are deactivated • all outputs are updated • the close function of the I/O driver is executed • control passes into the 'STOP' state
SPG 13	BADCAL	is executed if a firmware POU is called which does not exist	• user tasks are deactivated • all outputs are updated • the close function of the I/O driver is executed • control passes into the 'STOP' state
SPG 14	IOERROR	is executed if an error in the I/O driver occurs while the process is running	• control continues
*SPG 16	MATHERR	is executed if a floating point error in an arithmetic function occurs	• user tasks are deactivated • all outputs are updated • the close function of the I/O driver is executed

No.	Name	Meaning	Actions
			• control passes into the 'STOP' state
SPG 17	CPU_OVERLOAD	is executed if a CPU overload occurs	• user tasks are deactivated • all outputs are updated • the close function of the I/O driver is executed • control passes into the 'STOP' state
SPG 18	INITIODRV_ERR	is executed if an error occurs while initializing the I/O driver during a cold or a warm start	• control is not started
SPG 19	BOUNDS_ERR	is executed if the bounds of an array or structure have been exceeded. Only executed if the check box 'Index check' or the check box 'Array boundary check' has been activated in the dialog 'Resource settings...' in MachineLogic	• user tasks are deactivated • all outputs are updated • the close function of the I/O driver is executed • control passes into the 'STOP' state
SPG 20	BUS_ERR	is executed if variables with location and data type ≥ 2 bytes have been used with odd addresses or if there has been an internal error in MachineLogic. Only in case of Motorola platforms.	• user tasks are deactivated • all outputs are updated • the close function of the I/O driver is executed • control passes into the 'STOP' state
SPG 21	STRING_ERR	is executed if an error during a string operation appears, e.g. a string should be replaced by another but it is not found.	The behavior of a string exception has changed. In the default configuration the SPG 21 is called after a string exception occurs and an entry in the error catalog is made including the module and the line number. The control stays in the 'RUN' state. In MachineLogic Runtime the behavior is defined as the following if the “Stack check on PLC” check box, located on the Resource settings dialog, is checked: • all user tasks are deactivated • all outputs are updated • the close function of the I/O driver is executed • control passes into the 'STOP' state

☝ *The action described for SPG16 is accurate if MachineLogic Runtime is the ONLY program running that uses floating point operations. If another program (such as Interact or the MachineShop Shell) using floating point math is running, or has run, from the DOS command prompt, then SPG16 does not execute upon a divide_by_zero error. Instead, the result (quotient) is represented as INFINITY, and the control continues in the 'RUN' state. Refer to Chapter 7 for more information.

How to declare user tasks in MachineLogic

- Insert the task in the subtree 'Physical Hardware' of the project tree.
- Configure the task settings in the dialog 'Task settings...'. To open this dialog, right-click the folder with your task name (for example MyTask1) in the left pane and select 'Settings...'.
- Associate programs to the task.

Refer to the MachineLogic help for detailed information about inserting tasks and task settings.

How to use system tasks and SPGs in case of MachineLogic Runtime errors

If a runtime error occurs in MachineLogic Runtime, such as a division by zero or a stack overflow, the corresponding SPG is automatically executed without any programming effort from the application programmer.

If any additional functionality is needed such as setting all outputs to zero, the application programmer has the ability to edit the corresponding code in MachineLogic.

The function block CLR_OUT can be used to set all outputs to zero. The functions COLD_RESTART, WARM_RESTART, HOT_RESTART and CONTINUE can be used to restart or continue the program execution. These functions and function blocks are described in the online help.

To edit the additional functionality in MachineLogic:

- Edit a program with the code body containing the additional functionality in the subtree 'Logical POUs' of the project tree.
- Insert a task of the type 'SYSTEM' in the subtree 'Physical Hardware' of the project tree.
- Display the dialog 'Task settings...' to choose the SPG where you want to add the functionality.
- Associate the program to the system task.

Tasks and watchdogs

In MachineLogic Runtime each user-defined task has its own adjustable watchdog. This watchdog checks if the task execution has been finished at the end of the watchdog time interval. If it has not been finished, the system task SPG 10 'WATCHDOG' is executed and the control passes into the 'STOP' state if no other actions are programmed. An entry in the error catalog of MachineLogic Runtime is also made. The watchdog time interval starts when the task is ready for execution. The watchdog time interval is set in the dialog 'Task settings...' in MachineLogic. The 'Task settings…' dialog is discussed later this chapter.

If the task execution time and the watchdog time have nearly the same value and there is a high CPU load, the watchdog time could possibly be exceeded while completing some online operations.

One cause for this behavior may be that you have activated the address status with powerflow while debugging in online mode.

Online operations which cause a reading or writing access on the hard disk may also interrupt the task execution for a short time. These online operations are:

- Calling the dialog 'Resource:resource name' by clicking on the 'Info' button in the control dialog
- Downloading the boot project
- Downloading a zipped project

The following figure shows that a task exceeds its watchdog time because the execution of the task is interrupted for a short period of time, multiple times.

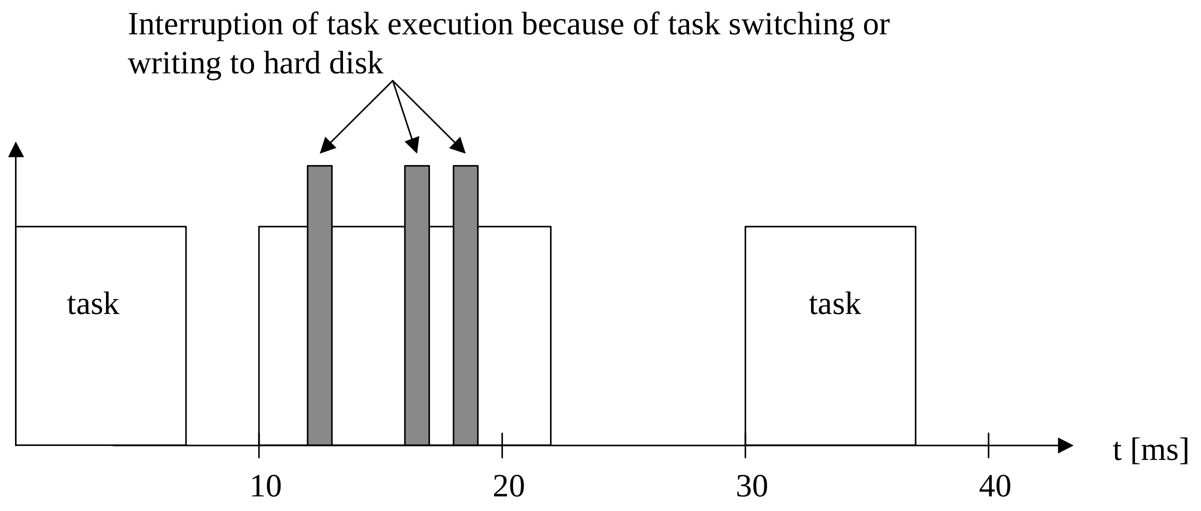

Interruption of task execution because of task switching or writing to hard disk
task
task
t [ms]
10
20
30
40

Figure 5-7: Tasks and Watchdog Times

In the example the watchdog time of the task is set to 10 ms. The task exceeds its watchdog time at 20 ms. If the watchdog time of the task is set to 20 ms, it is executed for the next time at 30 ms. In this case one task execution at 20 ms is skipped.

Tasks and programs

Programs are program organization units or POUs as defined in IEC 1131-3. The code bodies for programs are edited in MachineLogic. Within these programs other programs, function blocks, or functions can be called. Programs have to be associated to a task. Associating a program to a task means that the program is executed when the task is activated. In MachineLogic Runtime and MachineLogic it is possible to associate several programs to one task. In those cases the first program in the task directory is executed first. Then the program below is executed and so on.

Please refer to the MachineLogic help and documentation for detailed information about programs.

How to use programs in MachineLogic

- Insert the program in the subtree 'Logical POUs' of the project tree.
- Edit the code body.
- Perform the variable declaration.
- Compile the program.
- Associate the program to a task.

Tasks and preemptive scheduling

Let us assume we have a project with five user tasks and an Interact application. The tasks have the following properties.

Task	Priority	Interval Setting
A	0	5ms
B	1	10ms
C	2	15ms
D	3	4ms
E	4	8ms
Interact	NA	NA

At first the task having the highest priority (Task A) is executed. Then the task with the next lower priority (Task B) is executed. The programs associated to the task are executed according to their appearance in the project tree. After all tasks are executed then Interact is executed.

The data for a task that is interrupted is written to the stack. After a higher priority task finishes executing, the data for the lower priority task is read from the stack and execution is finished. These system level tasks are shown in figures 5-2 and 5-3.

This task switching with interrupting tasks with lower priority is called pre-emptive scheduling. The resulting task schedule is shown in the following figure.

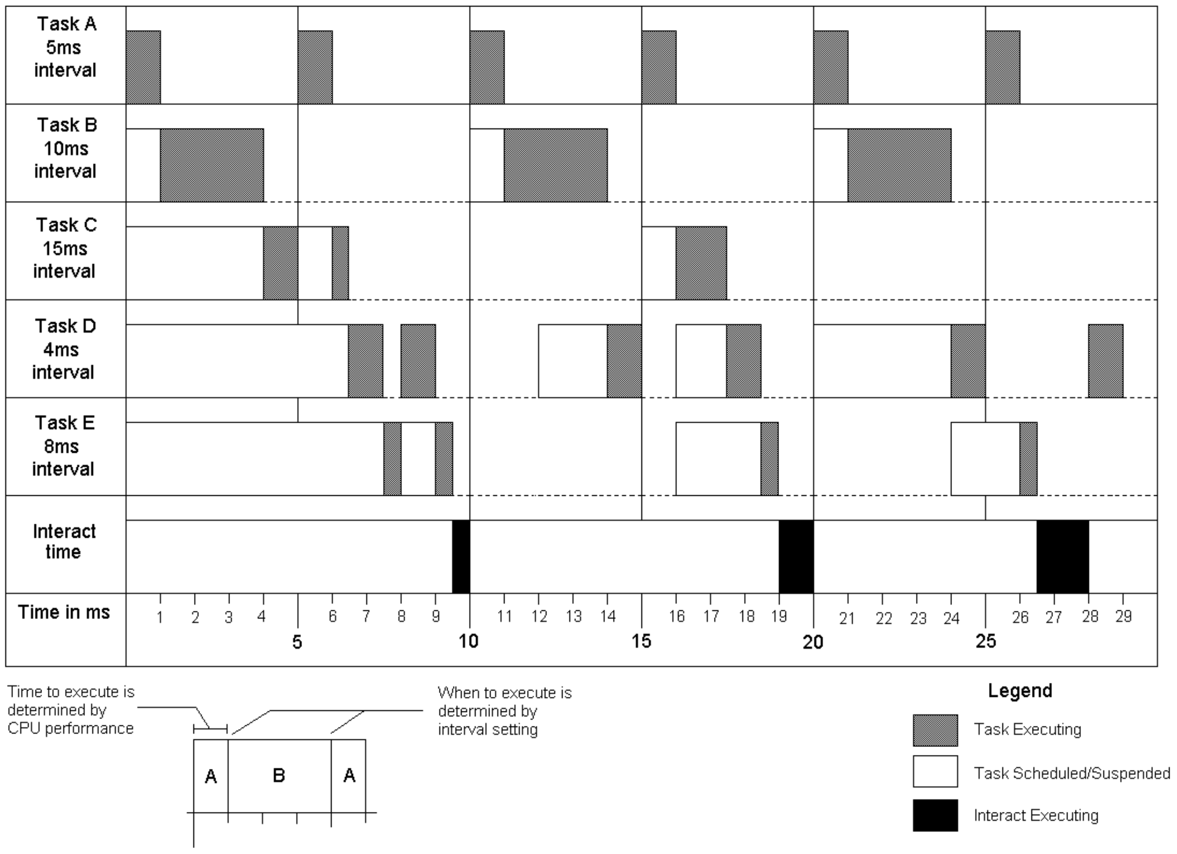


Figure 5-4: Tasks and preemptive scheduling

Downloading, Controlling and Debugging

This chapter provides information about:

- ☐ Downloading
- ☐ Projects and bootprojects
- ☐ Operational states
- ☐ Controlling and operational states
- ☐ Debugging and operational states
- ☐ System flags

Downloading, Controlling and Debugging

Downloading

Before downloading an application to MachineLogic Runtime you have to compile it in MachineLogic. Downloading means sending the project to the target, which in this case is a PowerStation. Downloading is accomplished by pressing the button 'Download' in the control dialog in MachineLogic, and by pressing the corresponding button in the dialog 'Download' which appears automatically.

Several possibilities of download can be used with MachineLogic Runtime:

- Downloading a project
- Downloading a zipped project
- Downloading a boot project

Downloading a project means that the full project including POUs and the configuration data are downloaded to the MachineLogic Runtime memory of the target. The control passes into the 'STOP' state. The project does not remain in memory if you switch off the control or if there is a power failure.

To start program execution after download, MachineLogic Runtime first determines if a warm start is even possible. If a warm start is possible, you have two options: either cold start or warm start. If a warm start is not possible, then a cold start is automatically performed.

During a cold start all data are initialized. During a warm start only non-retentive data are initialized.

Downloading a zipped project means that the project is zipped first and then downloaded to the runtime workstation. The zipped project can be uploaded via the dialog 'Upload' in MachineLogic. The zipped project should be downloaded for documentation purposes if application development has been finished and taken into operation.

Downloading a boot project means downloading a full project including the POUs and the configuration data to the hard drive of the target. The boot project remains on the hard drive if you switch off the runtime workstation or if there is a power failure. In case of switching the runtime workstation on, the boot project is loaded and the program execution is automatically started.

📖 Refer to the MachineLogic help for more information about downloading.

Operational states in MachineLogic Runtime

MachineLogic Runtime works with four different operational states:

Operational state	Description of the operational state
POWER ON	• power supply is on • no program is loaded
STOP	• program is loaded • user tasks are inactive • inputs of the I/O image are not updated • output signals are not transmitted to the I/Os
RUN	• program execution is activated • user tasks are active • inputs of the I/O image are updated according to the I/O configuration • outputs of the I/O image are updated according to the I/O configuration and the program execution
HALT	• program execution is halted at a breakpoint • user tasks are inactive • inputs of the I/O image are not updated • outputs of the I/O image are not updated

The current state of the control program is displayed either on the top of the control dialog or in the dialog 'Resource: *resource name*' in MachineLogic. If 'debug' is displayed behind the current state in the control dialog, it means that breakpoints have been set or variables have been forced.

Changing the operational states - overview

Using the control dialog in MachineLogic you can change the operational state of the control program. The following figure shows from which state you can switch to another. The names of the buttons in the control dialog are written next to the arrows. The buttons for changes that cannot be done in the current state are grayed in the control dialog.

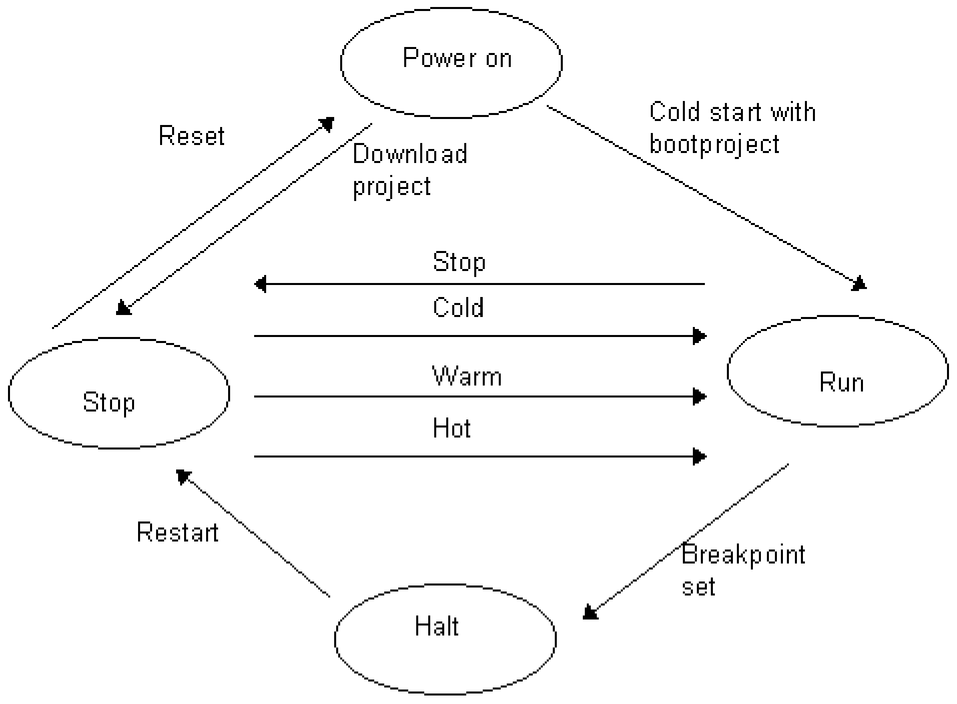


Figure 6-1: Changing into different operational state

Controlling and changing operational states

From MachineLogic you can control the control program; for example, starting and stopping the program execution. The following section describes how to change the states and what happens on MachineLogic Runtime.

Starting the program execution

State is changed from → to	Button to press in the control dialog	Description
Stop → Run	Cold	• a cold start is done • all data are initialized • SPG 1 is called • all user tasks are activated • the program execution is activated
Stop → Run	Warm	• a warm start is done • only non-retentive data are initialized • SPG 0 is called • all user tasks are activated • the program execution is activated
Stop → Run	Hot	• a hot start is done • no data are initialized • all user tasks are activated • the program execution is activated • Not available in case of starting the program execution for the first time after downloading

Stopping the program execution

State is changed from → to	Button to press in the control dialog	Description
Run → Stop	Stop	• all user tasks are set inactive having finished their working cycle • SPG 2 is called • the outputs of the I/O image are written • the program execution is stopped • according to the I/O driver the physical outputs are written or not

General reset

State is changed from → to	Button to press in the control dialog	Description

State is changed from → to	Button to press in the control dialog	Description
Stop → Power on	Reset	• the project is deleted • a general reset is done

Debugging and online mode

While the control is running, several functions for debugging your control program can be used in MachineLogic Runtime.

The following debug possibilities are available for MachineLogic Runtime:

- Viewing the current variable values represented with colors in the worksheets in online mode
- Switching between variable status and address status with powerflow
- Forcing and overwriting variables
- Setting and resetting breakpoints

All these functions can be handled via the graphical user interface of MachineLogic, where you can open the worksheets in online mode using the instance tree. For information on how to use the online mode refer to the MachineLogic help.

Using the address status with powerflow means that the program execution of the first task inserted in the project tree is interrupted just for a short time to read the current values of all variables. In case of a great number of addresses to be read the real-time conditions may not be met.

Debugging and operational states

Debugging with set breakpoints

Operational state	Button to press in the control dialog	Description
Halt [Debug]	Go	• only available if a breakpoint has been set • goes to the next breakpoint and halts the program execution at this point
Halt [Debug]	Trace	• only available if a breakpoint has been set • goes to the next line or object and halts the program execution at this point • if a function or function block call is reached, the function or function block code body is opened for debugging
Halt [Debug]	Step	• only available if a breakpoint has been set

Operational state	Button to press in the control dialog	Description
		• goes to the next line or object and halts the program execution at this point • if a function or function block call is reached, the call is stepped over and the next line is highlighted • if the end of a program is reached, the next STEP command starts the program execution at the beginning of the program. The program is stopped at the next breakpoint • if the end of a function or function block is reached , the next STEP command continues the program debugging at the caller of the function or function block

Stopping the control while debugging

State is changed from → to	Button to press in the control dialog	Description
Halt [Debug] → Stop [Debug]	Restart	• all user tasks are set inactive having finished their working cycle • SPG 2 is called • the outputs of the I/O image are written • the program execution is stopped • according to the I/O driver the physical outputs are written or not

System flags

System flags are flags giving information about the system state such as forced variables, CPU performance etc. These flags have fixed memory addresses and can be used by the control program to get this information.

All system flags in the following table are already declared as global variables in the global variable worksheet if you are creating a project in MachineLogic. To use them in POUs just declare a non-direct variable in the variable worksheet of the POU using the keyword VAR_EXTERNAL and the name of the system flag as the variable name.

The system flags are shown in the following table:

Name	**Data type**	**Logical address (Byte)**	**Logical address (Bit)**	**Description**
PLCMODE_ON	BOOL	0	0	TRUE := current control mode is POWER ON
PLCMODE_RUN	BOOL	0	1	TRUE := current control mode is RUN
PLCMODE_STOP	BOOL	0	2	TRUE := current control mode is STOP
PLCMODE_HALT	BOOL	0	3	TRUE := current control mode is HALT
PLCDEBUG_BPSET	BOOL	1	4	TRUE := one or more breakpoints are set

Name	Data type	Logical address (Byte)	Logical address (Bit)	Description
PLCDEBUG_FORCE	BOOL	2	0	TRUE := one or more variables are forced
PLCDEBUG_POWERFLOW	BOOL	2	3	TRUE := powerflow is active
PLC_TICKS_PER_SEC	INT	44	-	Number of system ticks per second used by MachineLogic Runtime as the system time base. This value defines the time resolution of MachineLogic Runtime for timer function blocks like TON, TOF, or TP and the shortest cycle time of Interact and cyclic tasks.
PLC_SYS_TICK_CNT	DINT	52	-	Number of counted MachineLogic Runtime system ticks.

In addition to these system flags visible in the declaration of global variables, system flags containing task and error information can be used for special purposes.

User task information

The system flags shown in the following table are available for each user task:

Name	Data type	Logical address	Description
CURDURATION	INT	1032	Current task duration including preemptive scheduling in ticks
MINDURATION	INT	1034	Minimum task duration in ticks
MAXDURATION	INT	1036	Maximum task duration in ticks
CURDELAY	INT	1040	Current task delay in ticks
MINDELAY	INT	1042	Minimum task delay in ticks
MAXDELAY	INT	1044	Maximum task delay in ticks

If there is more than one user task, the logical address for the next user task information is calculated by 1032 + N*64 where N is the task number. In this case the current task duration for the next user task is located at 1096.

To use these system flags the logical address is to be inserted in the field 'AT' in the dialog 'Declaration of variables or FB instances' in MachineLogic.

For example, if you want to declare the system flag CURDURATION you have to insert in the field 'AT':

% MW1.1032

Error catalog information

The system flags of the error catalog are shown in the following table:

Name	Data type	Logical address	Description
PLC_MAX_ERR	INT	60	Maximum size of the MachineLogic Runtime error catalog
PLC_ERR_CNT	INT	62	Current number of errors beginning at location %MD 1.64
ERROR	DWORD	64	First error number
ERROR_N	DWORD	68	Next logical address of error number is calculated by 64 + N*4 where N is the error index

To use these system flags, set the corresponding variable declaration in the global variable worksheet of the resource.

MachineLogic Runtime Data Types

This chapter provides information about:

- How MachineLogic reacts upon the detection of a floating point error condition
- How to use string data types in control programs

MachineLogic Runtime Data Types

Real Numbers

This section describes how MachineLogic Runtime reacts upon detection of floating point error conditions.

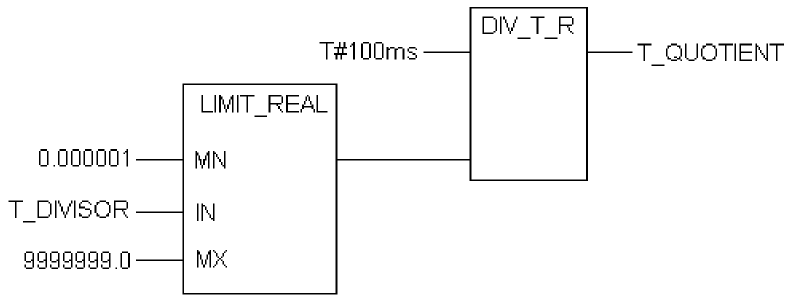
Error treatment of real numbers

MachineLogic Runtime reacts to floating point errors differently, depending on whether or not it is the only program running that uses floating point math operations. If MachineLogic is the only program running, then SPG16 is executed (if one exists) upon detection of a math error, followed by this sequence of actions:

- user tasks are deactivated
- all outputs are updated
- the close function of the I/O driver is executed
- the control passes into the 'STOP' state

If another program (such as Interact or the MachineShop Shell) using floating point math is running, or has run, from the DOS command prompt, then SPG16 does not execute upon a divide_by_zero error. Instead, the result (quotient) is represented as INFINITY, and the control continues in the 'RUN' state.

There is a known problem with the floating point error detection when using the DIV_T_R (Divide Time by Real) function. If the divisor is allowed to be 0.0, it can cause a sudden and unexpected system failure. To prevent this, we recommend that this function not be used. If it is absolutely necessary to use this function in your application, we recommend that a LIMIT_REAL function be used in conjunction with DIV_T_R to prevent the divisor from reaching a value of 0.0, as shown below:



Strings

This section describes how to use string data types in control programs. Also described is how to format strings with the provided MachineLogic string functions.

Error treatment of string errors

As described in the chapter "Control application and MachineLogic Runtime", only the SPG 21 is called after a string error occurs and an entry in the error catalog is made including the module and the line number. In a customer-specific MachineLogic Runtime version this behavior may be different.

☝ As the default configuration only the exception (SPG 21) is generated. If a string error occurs the control is not stopped.

The error reaction of the string functions is described in the following table.

String function	Error condition	Error reaction
LEFT	L > len(IN) L > maxlen(OUT)	OUT =''
RIGHT	L > len(IN) L > maxlen(OUT)	OUT =''
MID	L > maxlen(OUT) P < 0 P+L > len(IN)	OUT =''
CONCAT	len(IN1) + len(IN2) > maxlen(OUT) IN2 == OUT	OUT =''
INSERT	len(IN1) + len(IN2) > maxlen(OUT) P < 0 P > len(IN1) IN2 == OUT	OUT =''
DELETE	len(IN) – L > maxlen(OUT) P+L > len(IN) P < 0	OUT =''
REPLACE	len(IN) – L + len(IN2) > maxlen(OUT) IN2 == OUT P+L > len(IN1) P < 0	OUT =''
FIND	--	--
ASSIGN (:=,MOVE, ST)	len(IN) > maxlen(OUT)	OUT =''
*_TO_STRING	please refer chapter *_TO_STRING	OUT =''
STRING_TO_*	please refer chapter STRING_TO*	Value = 0

☝ The identifiers L,P,IN,... represent the parameter of the string functions in the IEC language FBD. '*len(...)*' represents the actual string length and '*maxlen(...)*' the maximum string length allowed (maxlen(Default string) = 80).

STRING_TO_*

The functions STRING_TO_* are used to generate a number from a string which includes a value. In the following some rules for numeric literals are described. These numeric literals can be connected to the several STRING_TO_* functions.

General rules

```
digit ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
hexDigit ::= digit | 'A' | 'B' | 'C' | 'D' | 'E' | 'F' |'a' | 'b' | 'c' | 'd' | 'e' | 'f'
integer ::= digit{['_'] digit}
realInteger ::= digit{digit}
signedRealInteger ::= ['+'|'-'] realInteger
exponent ::= ('E'| 'e')signedRealInteger
bitStream ::= BYTE | WORD | DWORD
unsignedTyp ::= USINT | UINT | UDINT
signedTyp ::= SINT | INT | DINT
timeTyp ::= 'T' | 't' | 'TIME' | 'time'
```

Allowed literal for the functions STRING_TO_BYTE, _WORD and _DWORD

```
hexInteger := 16#hexDigit{['_'] hexDigit}
bitStreamLiteral ::= [bitStream '#']digit{['_'] digit}
```

Allowed literal for the functions STRING_TO_SINT, _INT and _DINT

```
signedLiteral ::= [signedTyp '#'] ['+'|'-'] integer
```

Allowed literal for the functions STRING_TO_USINT, _UINT and _DINT

```
unsignedLiteral ::= [unsignedTyp '#'] digit{['_'] digit}
```

Allowed literal for the function STRING_TO_REAL

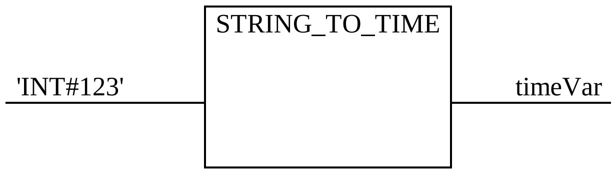
```
realLiteral ::= ['REAL' '#'] signedRealInteger'.'realInteger [exponent]
```

Allowed literal for the function STRING_TO_TIME

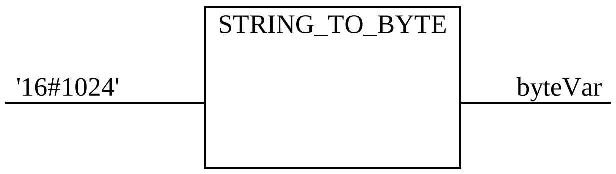
```
durationLiteral ::= timeTyp '#' digit{digit} ['ms' | 'MS']
```

Errors which may occur during the value transformation

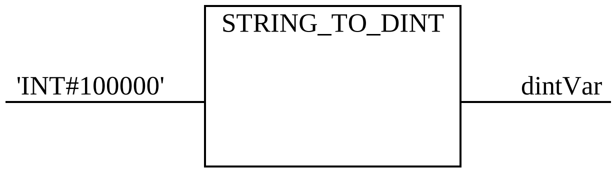
- The connected string represents a literal which is not valid for this data type.
  Example:



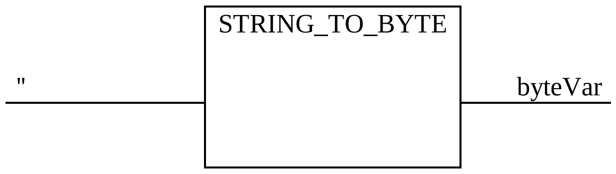
- The value is out of the range for the specified data type.
  Example:



- The value does not fit to the literal.
  Example:



- The connected string is empty.
  Example:



Error reaction

In the error conditions, described above, the result variable gets the value '0'. If the connected string is empty (even no blanks), the program is not interrupted. In all other cases the program checks whether the exception handling is enabled and the error process is started. For more information see the chapter 'Error treatment of string errors'.

Specialty of transformation

The program searches for the first occurrence of the character '#' in the connected string. If this character is found, an evaluation of the type or format identifier in front of this character is done. In principle it is possible to transform a number which is embedded in a string.

Example: 'Jack is INT#40 years old' Result: 40

The further evaluation is performed if the character '#' is not included in the string. If this character is not found in the string, the evaluation starts at the beginning of the string. All white spaces are skipped until the first character that is not a white space is found.

The evaluation is stopped if a character that is not a valid literal is found. The evaluation of the strings `'INT#50 J'` and `'INT#50J'` gives the result value of 50 for both.

☝ ASCII codes regarded as 'white spaces':

Character	Value
horizontal tab	9 (09h)
line feed	10 (0Ah)
vertical tab	11 (0Bh)
form feed	12 (0Ch)
carriage return	13 (0Dh)
blank	32 (20h)

*TO_STRING

The functions *_TO_STRING are used to put the value of a number into a string. Because you can create the text according to special conceptions, the functions get a format string that can include the conversion instructions.

Principle construction of the conversion instructions

```
digit ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
convertStr ::= '%'[width]['.'precision]['l'] typ
width ::= [0] digit {digit}
precision ::= digit {digit}
typ ::='d' | 'u' | 'x' | 'X' | 'e' | 'E' | 'f'
```

Valid conversion instructions of BYTE_, WORD_ and DWORD_TO_STRING

```
convStr ::= '%'[width]['.'precision]['l']('u' | 'x' | 'X')
```

Valid conversion instructions of SINT_, INT_ and DINT_TO_STRING

```
convStr ::= '%'[width]['.'precision]['l']('d')
```

Valid conversion instructions of USINT_, UINT_ and UDINT_TO_STRING

```
convStr ::= '%'[width]['.'precision]['l']('u')
```

Valid conversion instructions of REAL_TO_STRING

```
convStr ::= '%'[width]['.'precision]['l']('e' | 'E' | 'f')
```

Valid conversion instructions of TIME_TO_STRING

```
convStr ::= '%'[width]['.'precision]['l']('u')
```

Description of the optional instructions

Width
The optional instruction 'width' indicates how many characters are to put out at minimal. If necessary the output is padded with '0' (if a 0 is set before 'width') or with white spaces.

Precision
The optional instruction 'precision' indicates, how many characters are to put out at maximal, respectively the amount of significant decimal positions for the data type REAL.

Construction of the format string

Every conversion instruction may be embedded into a string. An example format string:

`'Jack is INT#`***`%ld`***` years old'`.

In this example the conversion instruction is shown in italic letters.

☝ If the character '%' appears in the string, the character must be put into the format string, twice.

Standard format strings for BYTE_, WORD_ and DWORD_TO_STRING

`'16#lX'`

Standard format strings for SINT_, INT_ and DINT_TO_STRING

`'%ld'`

Standard format strings for USINT_, UINT_ and UDINT_TO_STRING

`'%lu'`

Standard format strings for REAL_TO_STRING

`'%lE'`

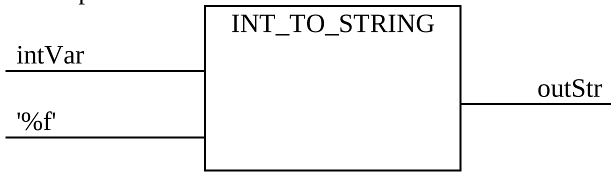
Standard format strings for TIME_TO_STRING

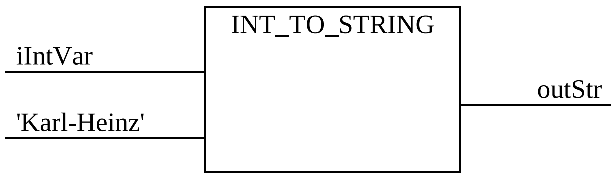
`'T#%lu ms'`

☝ The standard format strings are valid if an empty string is connected as format string.

Errors that may occur during the transformation

- Illegal or no conversion instruction
  Example:





- The result string exceeds the maximum allowed length of the connected output string.

Error reaction

In the error conditions, described above, the output string is cleared. Then the program checks whether the exception handling is enabled and the error process is started. For more information see the chapter 'Error treatment of string errors'.

IEC Compliance List

This chapter provides information about...

- ☐ the IEC compliance list

IEC compliance list

This document describes the available features with MachineLogic Development and Runtime, as relates to the IEC 1131 compliance list.

MachineLogic Version	MachineLogic Runtime Version	Feature	
		2.1.3 Table 1 Character set	
2.0	3.0	1	Windows character set
2.0	3.0	2	Lower case characters
2.0	3.0	3a	Number sign (#)
2.0	3.0	4a	Dollar sign ($)
2.0	3.0	5a	Vertical bar (|)
2.0	3.0	6a	left and right brackets "[ ]"
		2.1.3 Table 2 Identifier features	
2.0	3.0	3	Upper and lower case, numbers, leading or embedded underlines
2.0	3.0	2.1.5 Table 3 Comments	
		2.2.1 Table 4 Numeric literals	
2.0	3.0	1	Integer literals
2.0	3.0	2	Real literals
2.0	3.0	3	Real literals with exponents
2.0	3.0	4	Base 2 literals
2.0	3.0	5	Base 8 literals
2.0	3.0	6	Base 16 literals
2.0	3.0	8	Boolean FALSE and TRUE
2.0	3.0	2.2.2 Table 5 Character string literal features	

2.0		2.2.3.1 Table 7 Duration literal features
	3.0	1a Short prefix without underlines
	3.0	1b Long prefix without underlines
	3.0	2a Short prefix with underlines
	3.0	2b Long prefix with underlines
		2.3.1 Table 10 Elementary data types
2.0	3.0	1 BOOL
2.0	3.0	2 SINT
2.0	3.0	3 INT
2.0	3.0	4 DINT
2.0	3.0	6 USINT
2.0	3.0	7 UINT
2.0	3.0	8 UDINT
2.0	3.0	10 REAL
2.0	3.0	12 TIME
2.0	3.0	16 STRING
2.0	3.0	17 BYTE
2.0	3.0	18 WORD
2.0	3.0	19 DWORD
		2.3.3.1 Table 12 Data type declaration feature
2.0	3.0	4 Array data types
2.0	3.0	5 Structured data types
2.0	3.0	2.3.3.2 Table 13 Default initial values
		2.4.1.3 Table 15 Directly represented variables
2.0	3.0	1 Input location
2.0	3.0	2 Output location
2.0	3.0	3 Memory location
2.0	3.0	4,5 Single bit size
2.0	3.0	6 Byte (8 bits) size
2.0	3.0	7 Word Size
2.0	3.0	8 Double word (32 bits) size
		2.4.3 Table 16 Variable declaration keywords
2.0	3.0	VAR, VAR_INPUT, VAR_OUTPUT, VAR_IN_OUT, VAR_EXTERNAL, VAR_GLOBAL, AT, RETAIN

		2.4.3.1 Table 17 Variable type assignment features (-> Table 39)	
2.0	3.0	1	Declaration of directly represented, non-retentive variables
2.0	3.0	2	Declaration of directly represented, retentive variables
2.0	3.0	3	Declaration of locations of symbolic variables
2.0	3.0	4	Array location assignment (arrays as derived data type)
2.0	3.0	5	Automatic memory allocation of symbolic variables
2.0	3.0	7	Retentive array declaration (arrays as derived data type)
2.0	3.0	8	Declaration of structured variables
		2.4.3.2 Table 18 Variable initial value assignment features (Table 39)	
2.0	3.0	1	Initialization of directly represented, non retentive variables
2.0	3.0	2	Initialization of directly represented, retentive variables
2.0	3.0	3	Location and initial value assignment to symbolic variables
2.0	3.0	5	Initialization of symbolic variables
2.0		2.5.1.3 Table 19 Graphical negation of boolean signals	
	3.0	1	Negated input
	3.0	2	Negated output
		2.5.1.3 Table 20 Use of EN input and ENO output	
2.0	3.0	3	FBD without "EN" and "ENO"
2.0	3.0	2.5.1.4 Table 21 Typed and overloaded functions	
		1	Overloaded functions
		2	Typed functions
2.0	3.0	2.5.1.5.1 Table 22 Type conversion function features	
		1	*_TO_*
		2	TRUNC
		3	BCD_TO_*
		4	*_TO_BCD

2.0	3.0	2.5.1.5.2 Table 23 Standard functions of one numeric variable
		1 ABS
		2 SQRT
		3 LN
		4 LOG
		5 EXP
		6 SIN
		7 COS
		8 TAN
		9 ASIN
		10 ACOS
		11 ATAN
		2.5.1.5.2 Table 24 Standard arithmetic functions
2.0	3.0	12 ADD
2.0	3.0	13 MUL
2.0	3.0	14 SUB
2.0	3.0	15 DIV
2.0	3.0	16 MOD
2.0	3.0	17 EXPT
2.0	3.0	18 MOVE
2.0	3.0	2.5.1.5.3 Table 25 Standard bit-shift functions (FBD overloaded, IL typed)
		1 SHL
		2 SHR
		3 ROR
		4 ROL
		2.5.1.5.3 Table 26 Standard bitwise boolean functions
2.0	3.0	5 AND
2.0	3.0	6 OR
2.0	3.0	7 XOR
2.0	3.0	8 NOT

2.0	3.0	2.5.1.5.4 Table 27 Standard selection functions (typed, non extensible) 1 SEL 2a MAX 2b MIN 3 LIMIT
2.0	3.0	2.5.1.4 Table 28 Standard comparison functions (non extensible, typed for datatype STRING) 5 GT 6 GE 7 EQ 8 LE 9 LT 10 NE
2.0	3.0	2.5.1.5.5 Table 29 Standard character string functions 1 LEN 2 LEFT 3 RIGHT 4 MID 5 CONCAT 6 INSERT 7 DELETE 8 REPLACE 9 FIND
2.0	3.0	2.5.1.5.6 Table 30 Functions of time data type 1 ADD_T_T 4 SUB_T_T 10 MUL_T_AN 11 DIV_T_AN
		2.5.2.2 Table 33 Function block declaration features (Table 39)
2.0	3.0	1 RETAIN qualifier on internal variables
2.0	3.0	4a Input/output declaration (textual)
2.0	3.0	2.5.2.3.1 Table 34 Standard bistable function blocks 1 SR 2 RS

2.0	3.0	2.5.2.3.2 Table 35 Standard edge detection function blocks
		1 R_TRIG
		2 F_TRIG
2.0	3.0	2.5.2.3.3 Table 36 Standard counter function blocks
		1 CTU
		2 CTD
		3 CTUD
2.0	3.0	2.5.2.3.4 Table 37 Standard timer function blocks
		1 TP
		2a TON
		3a TOF
		2.5.3 Table 39 Program declaration features
2.0	3.0	1 RETAIN qualifier on internal variables
2.0	3.0	11 Declaration of directly represented, non-retentive variables
2.0	3.0	13 Declaration of locations of symbolic variables
2.0	3.0	15 Initialization of directly represented, non retentive variables
2.0	3.0	17 Location and initial value assignment to symbolic variables
2.0	3.0	19 Use of directly represented variables
		2.9.2 Table 40 Step features
2.0	3.0	1 Step, initial step (graphical form)
2.0	3.0	3a, 3b Step flag
		2.9.3 Table 41 Transitions and transition conditions
2.0	3.0	2 Transition condition using LD language
2.0	3.0	3 Transition condition using FBD language
2.0	3.0	4,4a,4b Use of connector
2.0	3.0	7 Use of transition name
2.0	3.0	7a Transition condition using LD
2.0	3.0	7b Transition condition using FBD
2.0	3.0	7c Transition condition using IL
2.0	3.0	7d Transition condition using ST

		2.9.4.1 Table 42 Declaration of actions	
2.0	3.0	1	Any boolean variable can be an action
2.0	3.0	2l	Graphical declaration in LD language
2.0	3.0	2f	Graphical declaration in FBD language
2.0	3.0	3s	Textual declaration in ST language
2.0	3.0	3i	Textual declaration in IL language
2.0	3.0	2.9.4.2 Table 43 Step/action association	
		1	Action block
		2	Concatenated action blocks
2.0	3.0	2.9.4.3 Table 44 Action block	
		1	Qualifier
		2	Action name
	3.0	2.9.4.4 Table 45 Action qualifiers	
2.0	3.0	2	Non-stored
2.0	3.0	3	overriding Reset
2.0	3.0	4	Set (Stored)
2.0	3.0	5	time limited
2.0	3.0	6	time delayed
2.0	3.0	7	Pulse
2.0	3.0	8	Stored and time delayed
2.0	3.0	9	Delayed and stored
2.0	3.0	10	Stored and time limited
		2.9.5 Table 46 Sequence evolution	
2.0	3.0	1	Single sequence
2.0	3.0	2a	Divergence of sequence selection
2.0	3.0	3	Convergence of sequence selection
2.0	3.0	4	Simultaneous sequences
2.0	3.0	5a	Sequence skip
2.0	3.0	6a	Sequence loop
2.0	3.0	7	Directional arrows

		2.7.1 Table 49 Configuration and resource declaration features	
2.0	3.0	1	CONFIGURATION
2.0	3.0	3	RESOURCE
2.0	3.0	4	VAR_GLOBAL within RESOURCE
2.0	3.0	5a	Periodic TASK
2.0	3.0	5b	Non-periodic TASK
2.0	3.0	6a	PROGRAM with PROGRAM-to-TASK association
2.0	3.0	6c	PROGRAM with no TASK association (by DEFAULT task)
2.0	3.0	7	Declaration of directly represented variables
2.0	3.0	2.7.2 Table 50 Task features	
		1a	Textual declaration of periodic TASK (by project tree)
		1b	Textual declaration of non-periodic TASK (by project tree)
		3a	Textual association with PROGRAMs (by project tree)
		5b	Preemptive scheduling

2.0	3.0	3.2.2 Table 52 Instruction list (IL) operators 1 LD 2 ST 3 S, R 4 AND 6 OR 7 XOR 8 ADD 9 SUB 10 MUL 11 DIV 12 GT 13 GE 14 EQ 15 NE 16 LE 17 LT 18 JMP 19 CAL 20 RET 21)
2.0	3.0	3.2.3 Table 53 Function block invocation features for IL language 2 CAL with load/store of inputs

2.0	3.0	3.3.1 Table 55 Operators on the ST language	
		1	Parenthesization
		2	Function evaluation
		3	Exponentiation
		4	Negation
		5	Complement
		6	Multiply
		7	Divide
		8	Modulo
		9	Add
		10	Subtract
		11	Comparison
		12	Equality
		13	Inequality
		14	Boolean AND (&)
		15	Boolean AND
		16	Boolean exclusive OR
		17	Boolean OR
2.0	3.0	3.3.2 Table 56 ST language statements	
		1	Assignment
		2	FB invocation and FB output usage
		3	RETURN
		4	IF
		5	CASE
		6	FOR (only upwards)
		7	WHILE
		8	REPEAT
		9	EXIT
		10	Empty statement

		4.1.3 Table 57 Representation of line and block	
2.0	3.0	2	Horizontal lines
2.0	3.0	4	Vertical lines
2.0	3.0	6	Horizontal/vertical connection (with connection dot)
2.0	3.0	8	Line crossings without connection (without gap)
2.0	3.0	10	Connected and non connected corners
2.0	3.0	12	Blocks with connecting lines
2.0	3.0	14	Graphic connectors
2.0	3.0	4.1.3 Figure 23 Feedback path	
		a	Explicit loop
		b	Implicit loop
		c	LD language equivalent
		4.1.4 Table 58 Graphic execution control elements	
2.0	3.0	1	Unconditional jump FBD
2.0	3.0	2	Unconditional jump LD
2.0	3.0	3	Conditional jump FBD
2.0	3.0	3	Conditional jump LD
2.0	3.0	5	Conditional return LD
2.0	3.0	6	Conditional return FBD
	3.0	4.2.1 Table 59 Power rails	
		1	Left power rail
		2	Right power rail
2.0	3.0	4.2.2 Table 60 Link elements	
		1	Horizontal link
		2	Vertical link
		4.2.3 Table 61 Contacts	
2.0	3.0	1	Normally open contact
2.0	3.0	3	Normally closed contact
		4.2.4 Table 62 Coils	
2.0	3.0	1	Coil
2.0	3.0	2	Negated coil
2.0	3.0	3	SET (latch) coil
2.0	3.0	4	RESET (unlatch) coil

MachineLogic Version	MachineLogic Runtime Version	Feature
		IEC 1131 Extensions
2.0	3.0	2 TR2: constant type specifier e.g. INT#12, BOOL#1

MachineLogic Version	MachineLogic Runtime Version	Feature
		supplementary features
2.0	3.0	BOOL8
2.0	3.0	Several functions and function blocks as described in the online function and function block help

Index