mapp Services

1 mapp CodeBox: Programming at runtime

mapp CodeBox can create and execute programs on the machine at runtime. Programs are created using a widget in the mapp View HMI application that allows the programming of ladder diagrams.

1.1 Concept

- PictureLink_Configuration
- PictureLink_IEC
- PictureLink_Diagnosis
- PictureLink_Visualization

User stories

**Machine operator**

- As a machine operator, I would like to have the ability to create and start programs during runtime.
- As a machine operator, I would like to have the ability to change and adapt machine commands during runtime.

**Application engineer**

- As an application engineer, I would like to be able to quickly and easily macros that can be used at runtime.
- As a machine operator, I would like to have the ability to protect certain variables from being modified at runtime.

1.1.1 Functionality

mapp CodeBox makes it possible to create and execute software on the machine during runtime. The programs are either created directly via the HMI application of the machine or imported externally. If programs are available, they can be started, stopped, deleted or exported at any time. The programs are executed in the context of MpCodeBoxCore. This means that when MpCodeBoxCore is called, the various programs defined via the parameters are also executed. The programs are executed in the same cycle time as function block MpCodeBoxCore. It must be ensured in the application that changes in the program at runtime do not damage the machine.
Preparation in Automation Studio

The MpCodeBox configuration must be configured in Automation Studio. This configuration applies to all MpCodeBox instances. A MpCodeBoxCore configuration is added for each MpCodeBox instance.
Which process variables are permitted to be accessed in the application is defined in the MpCodeBox configuration. This defines what should be possible later when creating programs directly on the machine and what should not. For example, the machine builder can limit variable access so that his customers can query the state of the application but cannot write to process values. Preparations include defining PV access and defining macros. Programming in the ladder editor is possible using Macros.

**During runtime**

A ladder diagram program can be created on the target system at runtime. Programs can be saved, started, stopped, etc. using the ladder editor. Programs can also be exported and imported to other machines.

**1.1.2 Macros**

This section explains how macros work.

Macros are required to program in the ladder editor. Macros are predefined so that the user does not have to know the names of the process variables by heart. This can be "Axis initialized", for example. This means that the user does not have to check whether the axis is switched on, referenced and ready for a command, but can use a macro to check these states and perform an action.

A macro can be an action or condition.

Macros must be defined in the MpCodeBox configuration. For how these must be defined, see here.

**Condition**

A condition represents a state of the machine. Any values can be processed; the result must always be a Boolean value. This means that the condition can be either **TRUE** or **FALSE**.

Conditions are always shown as contacts in the ladder editor. Examples of conditions:

- Mold open
- Product ready
- Button pressed
- State of digital input

**Action**
An action triggers a defined command. Only Boolean values can be processed. This means that the action can either be TRUE or FALSE.

Actions are always shown as coils in the ladder editor. If a condition occurs, then an action is triggered. Examples of actions:

- Start conveyor
- Stop conveyor
- Open mold
- Set digital output

**Arguments**

Arguments are additional parameters that can use conditions and actions. Arguments are defined in the MpCodeBox configuration in section Arguments. Different argument types can be created. For example, an index argument can be defined that can be used as a counter variable in arrays.

**1.1.3 Binding with mapp View**

If MpCodeBox is used in combination with widget "LadderEditor", then a ladder diagram program can be quickly and easily created, started, edited, imported, etc. via the programming interface. The widget provides several actions and events for this.

The ladder diagram program can be edited as often as needed via the widget.

**Requirements**

Function block MpCodeBoxCore must be enabled.

Actions and conditions must be created in section Macros of the MpCodeBox configuration.

For information about how the widget must be configured, see section Concept. For information about how to program in the ladder diagram editor, see section Programming in widget "LadderEditor".
1.1.4 Sandbox variable

A MpCodeBoxCore configuration must be added for each MpCodeBox instance. A “sandbox” variable can be specified in the configuration.

The Sandbox variable can be used to perform a secure download. The MpCodeBox function blocks access the addresses of process variables. Since addresses can change during a download, it is not recommended to work directly with the PVs to start machine processes, for example. Instead, the values of the sandbox variable can be mapped to local process variables via a PV mapping. This ensures a secure download.

This sandbox variable can be used to enable/disable different commands, for example. StartConveyor, StartHeating and StopGreenLight would be some examples of these commands.

For example:

A macro should be used to enable the heating unit on a machine. For this purpose, action "StartHeater" is created in the MpCodeBox configuration. As soon as the action is active, the heating unit should be enabled via the sandbox variable.

The sandbox variable is specified in the MpCodeBoxCore configuration:

The sandbox variable is mapped via a PV mapping to the variable that enables the heating unit:

1.2 Guides

1.2.1 Getting started

1.2.1.1 Creating a ladder diagram program at runtime

This section provides a step-by-step explanation of how to create a ladder diagram program at runtime using MpCodeBoxCore and widget "LadderEditor".
1.2.1.1.1 Creating a project

The first step is to create a new project with Automation Runtime Simulation.

1.2.1.1.2 Adding the mapp CodeBox configurations

The next step is to add the MpCodeBox and MpCodeBoxCore configurations:

The "mapp Link" we will need later for programming is located in the MpCodeBoxCore configuration:
1.2.1.1.3 Editing the MpCodeBox configuration

The MpCodeBox configuration must be edited next. A data storage device is specified under "Program device name". This must already be created in the target system configuration.

All ladder diagram programs are stored on the data storage device.

Macros must be created in order to later program in the ladder diagram editor.

A macro can be an action or condition. Actions are shown as coils in the ladder diagram editor; conditions are shown as contacts.

Action "Action_Input" and condition "Condition_Output" are now created.

For information about how to create macros, see section Macros.

If action "Action_Output" is TRUE, then variable "Output" in program "Data" is set to TRUE.

Condition "Condition_Input" becomes TRUE if variable "Input" in program "Data" is set to TRUE.

A program ("Data") is created that contains variables "Output" and "Input" of data type BOOL:
1.2.1.1.4 Adding a program

A program is added. In this case, we will add a Ladder Diagram program. We could use any of the other programming languages, however.

1.2.1.1.5 Adding MpCodeBoxCore to the program

Function block MpCodeBoxCore is added to the program:

1.2.1.1.6 Configuring MpCodeBoxCore

Connecting MpLink

First, connect input "MpLink" to the MpLink previously created in the Configuration View. Its address is passed on to the function block.
Enabling the function block

The function block is enabled by setting input "Enable". This is indicated by output "Active".
1.2.1.1.7 Creating the mapp View HMI application

A mapp View HMI application is created in the next step. For more information, see Creating a mapp View HMI application.

1.2.1.1.8 Adding widgets

Adding widget "LadderEditor"

Widget "LadderEditor" must then be added in the mapp View HMI application. For more information, see Concept. The same mapp Link that was connected to MpCodeBoxCore is used.

Adding button widgets

Two "ToggleButton" widgets are added so that the program can be tested later.

Inputs:

The first "ToggleButton" widget is added so variable "Input" can be set to TRUE or FALSE. Variable "Input" is selected under property "value". "Input" is specified as "text".

The important thing here is that the variable is enabled via OPC UA:

For Powerflow to be activated or deactivated, an event binding must be carried out. It can look like this, for example:

Enabling Powerflow with action "SetPowerflow" = True
Enabling Powerflow with action "SetPowerflow" = False

1.2.1.1.9 Adding library "MpServer"

Library "MpServer" must be transferred to the controller to use widget "LadderEditor". "MpServer" is used for internal communication.

![Library MpServer](image)

1.2.1.1.10 Generating the file structure

The project is then compiled and transferred to the target system.

1.2.1.1.11 Testing the HMI application

To conclude, the HMI application can be tested.

It is now possible to create and start ladder diagram programs using widget "LadderEditor".

Adding contacts and coils

For information about how to program in the ladder diagram editor, see section Programming in widget "LadderEditor".

The objective is to also have variable "Input" set to TRUE if variable "Output" in program "Data" is set to TRUE.

To do this, a contact is pulled into the editor using drag-and-drop. A dialog box opens automatically where condition "Conditon_Input" can be selected. A normally open contact is selected (" ").
A coil is then pulled into the editor using drag-and-drop. Action "Action_Output" is selected in the dialog box.

The program looks like this:
Saving the program

The new program is then saved. This is done by selecting "Save as" in the title bar. For information about how to work with the title bar, see here.

A dialog box appears where a name for the program can be assigned:
Loading and starting the program

The program can now be loaded via the title bar. A dialog box opens to select the previously created program.

The program is then started via the title bar.
Testing the program

If the program was started via the title bar, it can now be tested. To do so, Powerflow is enabled with button "Activate Powerflow".

Pressing button "Input" makes it clear that the program is working as expected.
1.3 Configuration

1.3.1 MpCodeBox configuration

All parameters in this configuration apply to all CodeBox instances being used in the project. This section describes the individual components of the configuration.

The configuration is composed of the following components:

- **General settings**: General settings can be defined in section "General".
- **PV access**: Variable access (read-only or full access) is defined under "PV access".
- **Arguments**: Different argument types can be defined under "Arguments".
- **Macros**: Actions and conditions that can be used in the editor are defined under "Macros".
- **Display**: Numerical arguments, units and limits that can be used in the various CodeBox instances can be defined in section "Display". The numerical arguments are defined in Arguments.

1.3.1.1 General settings

General settings are defined in section "General".

"Program device name" defines the data storage device that contains the programs that are executed at runtime. The data storage device must already be defined in the target system configuration.

Programs and macros can be protected with "Sign programs" and "Sign macros". "Sign XXX" must be set to "Required" for this. A password for signing the file must then be entered for "Passphrase". This ensures that the file cannot be manipulated externally.
1.3.1.2 PV access

"PV Access" defines which process variables in the project are permitted to be accessed.

If nothing is selected, then all variables that exist at runtime can be read- and write-accessed.

Different access groups can be defined using "Group". There are three possibilities here:

- All: "All" allows all variables in the project to be selected.
- All globals: If "All globals" is selected, a selection can be made for all global variables.
- Variable: A selection for a certain variable can be made with "Variable". The variable is defined under "PV name".

Whether read or write access is possible is defined for each group.

It is possible to select read access for all variables, for example ("All"). This makes all variables write-protected.

The variables that allow write access are then specified. This allows a command to be triggered, for example. In this example, write access is allowed for variables "Input" and "Output".

This ensures that write operations only take place on variables for which this is explicitly permitted.

If write access were permitted for all variables, then commands or values of a variable could be modified in such a way as to cause damage to a machine.

1.3.1.3 Arguments

Arguments that can be used in section "Macros" can be defined in section "Argument types". An argument is an additional parameter that can be used in a macro.

"Identifier" serves to uniquely identify the argument in the project. The name specified here can be used in the macros.

Various types of arguments can be created under "Type":

- None: With "None", the argument is not used.
- Numeric: With "Numeric", a numerical argument can be created. A unit can be assigned to numerical arguments under Display.
- Index: With "Index", an index argument can be created. Limit values ("Minimum", "Maximum") must be defined for this. The index argument can be used in the Macros as a variable for arrays, for example. If the argument is selected in the LadderEditor, then which array index should be used can be selected in a dialog box. When used, it could look something like this:
• **Process variable**: With "Process variable", a process variable can be used as argument type. This parameter is reserved for future use.

• **Sandbox**: A sandbox argument can be generated via "Sandbox". This parameter is reserved for future use.

### 1.3.1.4 Macros

Actions and conditions can be defined in section "Macros". For general information, see section Macros.

Actions can be used as coils in the LadderEditor.

Conditions can be used as contacts in the LadderEditor.

If a certain condition is met, then an action can be carried out.

"Identifier" allows a condition or action to be uniquely identified in the program.

#### 1.3.1.4.1 Line

"Line" specifies a program line or program that should should be executed when the action/condition is enabled. It is important to note the following rules.

**ST programming**

It is possible to use all operators, statements and keywords that are available in the ST editor. Character ";" must be added after each program line.

- **No functions or function blocks from libraries can be called**, for example function MpAlarmXCheckState or CfgSetIPAddr.

This looks something like this for an IF statement in the configuration:
Accessing process variables

PVs from the controller can be specified directly; they must also be specified in section PV access for this.

Process variables must be specified in "Line" as follows:

- **Global variables**: PVName
- **Local variables**: $TaskName:PVName$

**Local variables:**
Character ":" is used when accessing local variables. In order for this to be interpreted correctly, character "$" must be used in front of the task name. ";" must be entered after each program line. To prevent ";" from being added to the variable name when the PV is specified, an additional "$" character must be appended to the end of the PV specification. It thus looks like this: "$TaskName:PVName$;"

Example from the configuration:

![Diagram showing access to local variables](image)

This accesses variable "Input2" and task "CodeBox".

**Global variable**
Character "$" is not used when specifying global variables. The name of the variable can be entered directly.

Example from the configuration:

![Diagram showing access to global variables](image)

**mapp keyword "#R"**

"#R" is used to determine the state of a contact. A condition is thus assigned to result "#R". If a condition is created, this keyword must be used under "Line". It can look like this, for example:

![Diagram showing condition #R](image)

Variable "Input1" is assigned to #R in this example. This allows the variables to be used as a contact in the ladder editor.
mapp keyword "#IN"

"#IN" defines the input state of a coil. "#IN" is thus assigned to an action. The keyword must be used inside an action. It can look like this, for example:

```
In this example, #IN is assigned to output variable "Output2". In this way, variable "Output2" can be used as a coil in the ladder editor.
```

mapp keyword "#ARG"

A previously defined argument can be specified using "#ARGn", where n stands for the index of the argument. The argument must first have been created in section Arguments. It must then be specified for the condition or action under "Arguments" in order for it to be used under "Line". This could look like this in the configuration:

```
Argument "Position" was specified under "ARG1". A numerical argument type was used for this. It is therefore the first argument. In order to access the argument under "Line", "ARGn" ("ARG1" in this example) must be used.

Using an index argument:

If an argument of type "Index" is added in section Arguments, it can be used as follows, for example:
```

Condition "TestIndexARG" is created in the example. Array variable "MyArray" of type BOOL was created in program "Data". Which index of variable "MyArray" is accessed is defined using index argument "IndexARG".

If the condition in LadderEditor is added, then a dialog box appears to select the index that should be accessed:
mapp Services

Data sheet V

mapp keyword "#SB"

If keyword "#SB" is used, then it is possible to access the sandbox variable/structure specified in the MpCodeBoxCore configuration. This looks something like this in the configuration:

![Configuration Example](image)

Declaring variables

It is also possible to declare variables in "Line". This works as follows:

```plaintext
VAR {Scope}
VarName : BOOL := FALSE;
END_VAR
```

The following values can be used for the {scope}:

- `{}`: If no scope is specified, then the variable is valid only within the macro.
- `{LOCAL}`: With "LOCAL", the variable is only valid/visible within the program in which the macro is called.
- `{GLOBAL}`: With "GLOBAL", the variable is valid/visible within the entire MpCodeBoxCore instance.
- `{SHARED}`: With "SHARED", the variable is valid/visible within all MpCodeBoxCore instances.

The first parameter "VarName" defines the name of the variable. "BOOL" specifies the IEC data type. All IEC data types can be used. An array is specified with "ARRAY[0..n] OF BOOL". At the end, the variable can be assigned an initial value. In the example above, the initial value is "TRUE". Example of an initialization for an array:

```plaintext
VAR {Scope}
testArray: ARRAY[0..9] OF SINT:= [0,2,45,2,8,7,26,74,88,52];
END_VAR
```

It is not possible to create a structure.

This can be specified in the configuration as follows, for example:
1.3.1.4.2 Examples

This section lists different examples for macros.

Enabling a machine command

This example illustrates how to enable a machine command. The goal is to issue the command for heating if a certain temperature value is reached.

Macros:

Condition "TempLimit" and action "SetHeater" are created. As soon as variable "TempReached" = TRUE, variable "HeaterOn" should become TRUE.

Ladder editor:

The program looks like this in the ladder editor:
Requesting a state

This example explains how to request a state. Various values can be set depending on the result. The goal is to set the value of a variable based on whether the action is active.

**Macros:**

Condition "Temperature" and action "SetTempLimit" are created.

If the input value of action "SetTempLimit" is **TRUE**, a value should be assigned to variable "TempZone1". If the input value of the action is **FALSE**, the value is assigned to variable "TempZone2".

![Ladder editor diagram](image)
1.3.1.5 Display

This section makes it possible to define a unit and limit values for numerical variables that have been defined in section "Argument types".

The name of the defined argument is entered under "Identifier".
"Engineering unit" describes the unit that is used for calculations on the controller.
"Display unit" is only seen by the user. Conversion between units takes place based on the selected unit system.

1.3.2 MpCodeBoxCore configuration

A MpCodeBoxCore configuration is required for each MpCodeBox instance.

Sandbox variable
A variable/structure that can only be used by one MpCodeBox instance can be specified using a sandbox variable. For more information, see here.

1.4 Libraries

1.4.1 Data types and enumerators

1.4.1.1 Data types

1.4.1.1.1 MpCodeBoxCoreInfoType

This data type provides additional information about the MpCodeBoxCore component.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumberOfActivePrograms</td>
<td>UINT</td>
<td>Number of active programs.</td>
</tr>
<tr>
<td>Program</td>
<td>ARRAY[0..9] of MpCodeBoxPro-</td>
<td>Name of the programs currently being executed.</td>
</tr>
<tr>
<td>diag</td>
<td>diag</td>
<td>Diagnostic structure for the status ID.</td>
</tr>
</tbody>
</table>

1.4.1.1.2 MpCodeBoxCoreProgramsType

This data type returns a list of programs.
## 1.4.1.3 MpCodeBoxDiagType

This data type is used as a substructure within the structure to hold additional data for diagnostic purposes as well as to supply additional data about the status ID.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StatusID</td>
<td>MpCodeBoxStatusIDType</td>
<td>StatusID diagnostic structure</td>
</tr>
</tbody>
</table>

## 1.4.1.4 MpCodeBoxInfoType

This data type is used to provide additional information for components MpCodeBoxManager and MpCodeBoxProgramControl.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diag</td>
<td>MpCodeBoxDiagType</td>
<td>Diagnostic structure for the status ID.</td>
</tr>
</tbody>
</table>

## 1.4.1.5 MpCodeBoxProgramInfoType

This data type returns additional information for a program.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>STRING[100]</td>
<td>Name of the program.</td>
</tr>
<tr>
<td>State</td>
<td>MpCodeBoxProgramStateEnum</td>
<td>Current state of the program.</td>
</tr>
<tr>
<td>Error</td>
<td>DINT</td>
<td>Error number of the program.</td>
</tr>
</tbody>
</table>

## 1.4.1.6 MpCodeBoxStatusIDType

This data type is used as a substructure within the structure to hold additional data for diagnostic purposes as well as to supply additional data about the status ID.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>MpCodeBoxErrorEnum</td>
<td>Error code for mapp component.</td>
</tr>
<tr>
<td>Severity</td>
<td>MpComSeveritiesEnum</td>
<td>Describes the type of information supplied by the status ID (success, information, warning, error)</td>
</tr>
<tr>
<td>Code</td>
<td>UINT</td>
<td>Code for the status ID. This error number can be used to search for additional information in the help documentation.</td>
</tr>
</tbody>
</table>

## 1.4.2 Enumerators

### 1.4.2.1 MpCodeBoxItemEnum

This enumerated data type defines whether a macro or program should be used as an item.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpCODEBOX_ITEM_PROGRAM</td>
<td>Program is used as the item type.</td>
</tr>
<tr>
<td>mpCODEBOX_ITEM_MACRO</td>
<td>Macro is used as the item type.</td>
</tr>
</tbody>
</table>

### 1.4.2.2 MpCodeBoxProgramStateEnum

This enumerated data type defines the current state of the program.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpCODEBOX_STATE_UNDEFINED</td>
<td>Program undefined</td>
</tr>
<tr>
<td>mpCODEBOX_STATE_LOADING</td>
<td>Program loading</td>
</tr>
<tr>
<td>mpCODEBOX_STATE_INIT_UP</td>
<td>Program's initialization subroutine loaded</td>
</tr>
<tr>
<td>mpCODEBOX_STATE_RUNNING</td>
<td>Program running</td>
</tr>
<tr>
<td>mpCODEBOX_STATE_SUSPENDED</td>
<td>Program halted</td>
</tr>
<tr>
<td>mpCODEBOX_STATE_ERROR</td>
<td>Program in error state</td>
</tr>
</tbody>
</table>

## 1.4.2 Function blocks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MpCodeBoxCore</td>
<td>Function blockMpCodeBoxCore executes programs at runtime. The function block can load a list composed of several different programs and call it cyclically.</td>
</tr>
<tr>
<td>MpCodeBoxManager</td>
<td>MpCodeBoxManager can be used to delete, export and import programs/macros.</td>
</tr>
<tr>
<td>MpCodeBoxProgramControl</td>
<td>MpCodeBoxProgramControl can be used to interact with a program.</td>
</tr>
</tbody>
</table>
1.4.2.1 MpCodeBoxCore

Function block MpCodeBoxCore executes programs at runtime. The function block can load a list composed of several different programs and call it cyclically.

**Function block**

## Optional parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;MpComIdentType</td>
<td>MpLink</td>
<td>Connection to mapp (MpLink of an MpCodeBoxCore configuration).</td>
</tr>
<tr>
<td>BOOL</td>
<td>Enable</td>
<td>The function block is active as long as this input is set.</td>
</tr>
<tr>
<td>BOOL</td>
<td>ErrorReset</td>
<td>Resets function block errors.</td>
</tr>
<tr>
<td>MpCodeBoxCoreProgramsType</td>
<td>Programs</td>
<td>List of programs</td>
</tr>
<tr>
<td>BOOL</td>
<td>Start</td>
<td>Command used to start a program.</td>
</tr>
<tr>
<td>BOOL</td>
<td>Stop</td>
<td>Command used to stop a currently running program.</td>
</tr>
<tr>
<td>MpCodeBoxCoreInfoType</td>
<td>Info</td>
<td>Additional information about the component.</td>
</tr>
</tbody>
</table>

**Interface**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>MpLink</td>
<td>Pointer to MpComIdentType</td>
<td>Connection to mapp (MpLink of an MpCodeBoxCore configuration).</td>
</tr>
<tr>
<td>IN</td>
<td>Enable</td>
<td>BOOL</td>
<td>The function block is active as long as this input is set.</td>
</tr>
<tr>
<td>IN</td>
<td>ErrorReset</td>
<td>BOOL</td>
<td>Resets function block errors.</td>
</tr>
<tr>
<td>IN</td>
<td>Programs</td>
<td>MpCodeBoxCoreProgramsType</td>
<td>List of programs</td>
</tr>
<tr>
<td>IN</td>
<td>Start</td>
<td>BOOL</td>
<td>Command used to start a program.</td>
</tr>
<tr>
<td>IN</td>
<td>Stop</td>
<td>BOOL</td>
<td>Command used to stop a currently running program.</td>
</tr>
<tr>
<td>OUT</td>
<td>Active</td>
<td>BOOL</td>
<td>Function block active.</td>
</tr>
<tr>
<td>OUT</td>
<td>Error</td>
<td>BOOL</td>
<td>Error occurred during execution.</td>
</tr>
<tr>
<td>OUT</td>
<td>StatusID</td>
<td>DINT</td>
<td>Status information.</td>
</tr>
<tr>
<td>OUT</td>
<td>Running</td>
<td>BOOL</td>
<td>Specifies whether a program is currently being executed.</td>
</tr>
<tr>
<td>OUT</td>
<td>CommandBusy</td>
<td>BOOL</td>
<td>Function block currently executing command.</td>
</tr>
<tr>
<td>OUT</td>
<td>Info</td>
<td>MpCodeBoxCoreInfoType</td>
<td>Additional information about the component.</td>
</tr>
</tbody>
</table>

**mapp concept**

Section mapp components explains how mapp components are structured. In addition, it provides important notes for correctly using mapp components (e.g. for downloads).

For mapp function blocks, asynchronous handling does not have to be carried out in the initialization subroutine or in an acyclic task. However, an appropriately high stack must be configured in acyclic tasks (~6 kB).

1.4.2.1.1 Description

The MpLink of an MpCodeBoxCore configuration is used for this function block.

Setting command "Start" to TRUE allows a list of programs that have been defined on input parameter "Programs" to be started and called cyclically. The cycle time corresponds to the task class in which MpCodeBoxCore is called. If MpCodeBoxCore is in a task that is called with a cycle time of 10 ms, the cycle time in which the programs are called is also 10 ms. The programs must be located on the data storage device defined in den general settings in the MpCodeBox configuration.

All currently executing programs on MpCodeBoxCore are stopped by setting "Stop" to TRUE.
"Running" set to **TRUE** indicates that there are programs currently being executed; "Running" set to **FALSE** indicates no currently running programs.

The programs are launched in the order defined by input parameter "Programs". New programs are lined up at the back and launched afterwards.

Additional information is included in structure "Info". "NumberOfActivePrograms" indicates how many programs are currently being executed by MpCodeBoxCore. "Program" indicates the names of the programs currently being executed.

**Widget "LadderEditor"** can be used for a visual ladder diagram editor. For more information, see Binding with mapp View.

### 1.4.2.2 MpCodeBoxManager

MpCodeBoxManager can be used to delete, export and import programs/macros.

#### Function block

```plaintext
MpCodeBoxManager
&MpComIdentType MpLink
BOOL Enable
BOOL ErrorReset
MpCodeBoxItemEnum ItemType
&STRING[255] ItemName
&STRING[50] DeviceName
&STRING[255] FileName
BOOL Export
BOOL Import
BOOL Delete
BOOL CommandDone
MpCodeBoxInfoType Info
```

#### Interface

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>MpLink</td>
<td>Pointer to MpComIdentType</td>
<td>Connection to mapp (MpLink of an MpCodeCore configuration).</td>
</tr>
<tr>
<td>IN</td>
<td>Enable</td>
<td>BOOL</td>
<td>The function block is active as long as this input is set.</td>
</tr>
<tr>
<td>IN</td>
<td>ErrorReset</td>
<td>BOOL</td>
<td>Resets function block errors.</td>
</tr>
</tbody>
</table>
### mapp Services

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>ItemType</td>
<td>MpCodeBoxIte-</td>
<td>Defines the type of program.</td>
</tr>
<tr>
<td></td>
<td>ItemName</td>
<td>mEnum</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>DeviceName</td>
<td>Pointer</td>
<td>File device (data storage medium) where the files are stored.</td>
</tr>
<tr>
<td></td>
<td>FileName</td>
<td>STRING[255]</td>
<td>Name of the file that contains the program.</td>
</tr>
<tr>
<td>IN</td>
<td>Overwrite</td>
<td>BOOL</td>
<td>Defines whether imported or exported programs with the same name are overwritten.</td>
</tr>
<tr>
<td>IN</td>
<td>Export</td>
<td>BOOL</td>
<td>Command to start an export.</td>
</tr>
<tr>
<td>IN</td>
<td>Import</td>
<td>BOOL</td>
<td>Command to start an import.</td>
</tr>
<tr>
<td>IN</td>
<td>Delete</td>
<td>BOOL</td>
<td>Command to delete a specified program.</td>
</tr>
<tr>
<td>OUT</td>
<td>Active</td>
<td>BOOL</td>
<td>Function block active.</td>
</tr>
<tr>
<td>OUT</td>
<td>Error</td>
<td>BOOL</td>
<td>Error occurred during execution.</td>
</tr>
<tr>
<td>OUT</td>
<td>StatusID</td>
<td>DINT</td>
<td>Status information.</td>
</tr>
<tr>
<td>OUT</td>
<td>CommandBusy</td>
<td>BOOL</td>
<td>Function block currently executing command.</td>
</tr>
<tr>
<td>OUT</td>
<td>CommandDone</td>
<td>BOOL</td>
<td>Execution successful. Function block is finished.</td>
</tr>
<tr>
<td>OUT</td>
<td>Info</td>
<td>MpCodeBoxInfo-</td>
<td>Additional information about the component.</td>
</tr>
<tr>
<td></td>
<td>Type</td>
<td>Type</td>
<td></td>
</tr>
</tbody>
</table>

#### mapp concept

Section mapp components explains how mapp components are structured. In addition, it provides important notes for correctly using mapp components (e.g. for downloads).

For mapp function blocks, asynchronous handling does not have to carried out in the initialization subroutine or in an acyclic task. However, an appropriately high stack must be configured in acyclic tasks (~6 kB).

#### 1.4.2.2.1 Description

The MpLink of an MpCodeBoxCore configuration is used for this function block.

"ItemType" defines which item type should be exported. It is possible to choose either a macro or program. "ItemName" defines which program or macro should be used. For more information about macros, see section Macros. An export or import procedure can be started by setting input parameter "Export" to TRUE or "Import" to TRUE. The data storage device that should be used is defined with parameter "DeviceName". "FileName" defines which file should be exported or imported.

An imported program can be called cyclically by MpCodeBoxCore. The start command on MpCodeBoxProgramControl can be used for this.

The program/macro specified on "FileName" can be deleted by setting "Delete" to TRUE.

Setting "Overwrite" to TRUE will overwrite a program/macro with the same name on import/export.

#### 1.4.2.3 MpCodeBoxProgramControl

MpCodeBoxProgramControl can be used to interact with a program.

**Function block**

## Optional parameters
mapp Services

**Interface**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>MpLink</td>
<td>Pointer to MpComIdentType</td>
<td>Connection to mapp (MpLink of an MpCodeBoxCore configuration).</td>
</tr>
<tr>
<td>IN</td>
<td>Enable</td>
<td>BOOL</td>
<td>The function block is active as long as this input is set.</td>
</tr>
<tr>
<td>IN</td>
<td>ErrorReset</td>
<td>BOOL</td>
<td>Reset function block errors.</td>
</tr>
<tr>
<td>IN</td>
<td>Name</td>
<td>STRING[255]</td>
<td>Name of the program.</td>
</tr>
<tr>
<td>IN</td>
<td>Start</td>
<td>BOOL</td>
<td>Command to start the program.</td>
</tr>
<tr>
<td>IN</td>
<td>Suspend</td>
<td>BOOL</td>
<td>Command to suspend the program.</td>
</tr>
<tr>
<td>IN</td>
<td>Resume</td>
<td>BOOL</td>
<td>Command to resume the program. Reserved for future use.</td>
</tr>
<tr>
<td>IN</td>
<td>Stop</td>
<td>BOOL</td>
<td>Command to stop the program.</td>
</tr>
<tr>
<td>OUT</td>
<td>Active</td>
<td>BOOL</td>
<td>Function block active.</td>
</tr>
<tr>
<td>OUT</td>
<td>Error</td>
<td>BOOL</td>
<td>Error occurred during execution.</td>
</tr>
<tr>
<td>OUT</td>
<td>StatusID</td>
<td>DINT</td>
<td>Status information.</td>
</tr>
<tr>
<td>OUT</td>
<td>CommandBusy</td>
<td>BOOL</td>
<td>Function block currently executing command.</td>
</tr>
<tr>
<td>OUT</td>
<td>CommandDone</td>
<td>BOOL</td>
<td>Execution successful. Function block is finished.</td>
</tr>
<tr>
<td>OUT</td>
<td>Info</td>
<td>MpCodeBoxInfoType</td>
<td>Additional information about the component.</td>
</tr>
</tbody>
</table>

**mapp concept**

Section mapp components explains how mapp components are structured. In addition, it provides important notes for correctly using mapp components (e.g., for downloads).

For mapp function blocks, asynchronous handling does not have to be carried out in the initialization subroutine or in an acyclic task. However, an appropriately high stack must be configured in acyclic tasks (~6 kB).

**1.4.2.3.1 Description**

The MpLink of an MpCodeBoxCore configuration is used for this function block.
**MpCodeBoxCore** must be active to use this function block since only this function block can execute programs cyclically!

This function module manages a single program without affecting the other running programs. This program can be started, stopped or paused.

Input parameter "Name" defines which program should be managed. The desired program must be located on the data storage device specified in the **MpCodeBox configuration** when starting up the target system. If the program is not yet on the data storage device, it must be imported using **MpCodeBoxManager**.

Various commands can then be executed for the program:

- **Start**: Command "Start" reloads the specified program. If the program was stopped by setting command "Stop" to **TRUE**, it can be restarted by setting "Start" to **TRUE**. If changes were made to the program in the meantime and the program is saved, then these changes will be applied and executed.
- **Suspend**: Suspends the currently running program. It can be restarted with command "Resume".
- **Reload**: Reloads the program. Reserved for future use
- **Resume**: Resumes the program suspended previously with "Suspend". Changes to the program that were made previously are not applied. For this, the program would have to be reloaded, i.e. the program must be stopped (command "Stop") and then restarted (command "Start").
- **Stop**: Stops the program currently being executed. The program is uninstalled and can only be reloaded and executed with command "Start".

### 1.4.3 Status numbers

#### 1.4.3.1 -1064169452: Could not convert program

**Description:**
Converting program {2:ProgramName} on data storage device {3:FileName} in ST failed ({1:ErrorNumber}).

**Reaction:**
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.

**Cause/Solution:**
- Use of nonexistent macros.
- Check the macros.
- See the Logger for additional information.

**Additional information**
- {1:ErrorNumber}: Error number.
- {2:ProgramName}: Program that cannot be converted.
- {3:FileName}: Data storage device.

**Constant:**
mpCODEBOX_ERR_CONVERT_PROGRAM

**These function blocks / functions can report this error:**
- MpCodeBoxCore
- MpCodeBoxProgramControl

#### 1.4.3.2 -1064169453: Program stop failed

**Description:**
Program ({2:ProgramName}) could not be stopped. (Cause of error: {1:ErrorNumber})
Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.

Cause/Solution:
- Use MpComDump to create a file that contains all information about the components and services being used. Save the Logger information as well. Contact B&R Support with this information.

Additional information
- {1:ErrorNumber}: Error number.
- {2:ProgramName}: Name of the program that could not be stopped.

Constant:
mpCODEBOX_ERR_STOP_PROGRAM

These function blocks / functions can report this error:
- MpCodeBoxCore
- MpCodeBoxProgramControl

1.4.3.3 -1064169456: Program execution failed

Description:
An error occurred while executing the program.

Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.

Cause/Solution:
- Check the specified macros for e.g. array limits, division by 0, etc.
- See the Logger for additional information.

Constant:
mpCODEBOX_ERR_PROGRAM_EXECUTION

These function blocks / functions can report this error:
- MpCodeBoxCore
- MpCodeBoxManager
- MpCodeBoxProgramControl

1.4.3.4 -1064169457: Program load failed

Description:
An error occurred while loading the program.

Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.

Cause/Solution:
- Check the macros being used for errors.
mapp Services

Constant:

mpCODEBOX_ERR_START_PROGRAM

These function blocks / functions can report this error:

- MpCodeBoxCore

1.4.3.5 -1064169458: Incorrect signature for import

Description:
An error occurred while importing the program/macro. The signature is incorrect. The program was manipulated externally.

Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.

Cause/Solution:
- Check whether the program/macro was manipulated externally.

Constant:

mpCODEBOX_ERR_SIGNATURE

These function blocks / functions can report this error:

- MpCodeBoxManager

1.4.3.6 -1064169459: Invalid macro

Description:
The specified macro is invalid and cannot be used.

Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.

Cause/Solution:
- The specified macro is invalid.
- For information about specifying macros, see Macros.

Constant:

mpCODEBOX_ERR_MACRO_ERROR

These function blocks / functions can report this error:

- MpCodeBoxManager

1.4.3.7 -1064169460: Macro already exists

Description:
The specified macro already exists.

Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.
Cause/Solution:
  • Check the macro.

Constant:
mpCODEBOX_ERR_MACRO_EXISTS

These function blocks / functions can report this error:
  • MpCodeBoxManager

1.4.3.8 -1064169461: Macro not found

Description:
The specified macro does not exist.

Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.

Cause/Solution:
  • Check the macro.
  • Create the macro.
  • Import the macro using MpCodeBoxManager.

Constant:
mpCODEBOX_ERR_MACRO_MISSING

These function blocks / functions can report this error:
  • MpCodeBoxManager

1.4.3.9 -1064169462: Invalid item name

Description:
An invalid item name was specified.

Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.

Cause/Solution:
  • A null pointer or empty string was specified on "ItemName".

Constant:
mpCODEBOX_ERR_INVALID_ITEM_NAME

These function blocks / functions can report this error:
  • MpCodeBoxManager

1.4.3.10 -1064169463: Cannot access file

Description:
File {2:FileName} on data storage device {3:DeviceName} cannot be accessed. ({1:ErrorNumber})
Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.

Cause/Solution:
• Check the specified data storage device.

Additional information
• {1:ErrorNumber}: Error number.
• {2:FileName}: File that cannot be accessed.
• {3:DeviceName}: Data storage device containing the file.

Constant:
mpCODEBOX_ERR_FILE_SYSTEM

These function blocks / functions can report this error:
• MpCodeBoxManager

1.4.3.11 -1064169464: Invalid data storage device

Description:
The specified data storage device is invalid.

Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.

Cause/Solution:
• Nothing is attached to input "DeviceName".
• The attached file device does not exist.
• A connection to the desired file device could not be established.

Constant:
mpCODEBOX_ERR_INVALID_FILE_DEV

These function blocks / functions can report this error:
• MpCodeBoxManager

1.4.3.12 -1064169465: Invalid filename

Description:
The specified filename is invalid.

Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.

Cause/Solution:
• Check the specified filename on input parameter "FileName".

Constant:
mpCODEBOX_ERR_INVALID_FILE_NAME
These function blocks / functions can report this error:
  • MpCodeBoxManager

1.4.3.13 -1064169467: Invalid program

Description:
Program \{2:ProgramName\} is invalid. Error number: \{1:ErrorNumber\}.

Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.

Cause/Solution:
  • Check whether the program name is correct.
  • Check whether the program exists more than once.
  • See the Logger for additional information.

Additional information
  • \{2:ProgramName\}: Name of the invalid program.
  • \{1:ErrorNumber\}: Error number.

Constant:
mpCODEBOX_ERR_INVALID_PROGRAM

These function blocks / functions can report this error:
  • MpCodeBoxManager

1.4.3.14 -1064169468: Unsupported program type

Description:
The specified program type is not supported.

Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.

Cause/Solution:
  • Select/Specify a supported program type.

Constant:
mpCODEBOX_ERR_PROGRAM_TYPE

These function blocks / functions can report this error:
  • MpCodeBoxCore
  • MpCodeBoxProgramControl

1.4.3.15 -1064169469: Program not loaded

Description:
The program has not yet been loaded and cannot be executed.

Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.
Cause/Solution:

- A program can only be started after it has finished loading. The program must have state "mpCODEBOX_STATE_RUNNING". This can be read on the informational structure of MpCodeBoxCore, for example.

Constant:

```plaintext
mpCODEBOX_ERR_PROGRAM_NOT_LOADED
```

These function blocks / functions can report this error:

- MpCodeBoxCore
- MpCodeBoxProgramControl

1.4.3.16 -1064169470: Cannot execute program

Description:
The program cannot be executed. Error in the file.

Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.

Cause/Solution:

- Check the program.
- See the Logger for additional information.

Constant:

```plaintext
mpCODEBOX_ERR_PROGRAM_ERROR
```

These function blocks / functions can report this error:

- MpCodeBoxCore
- MpCodeBoxProgramControl

1.4.3.17 -1064169471: Program not found

Description:
Specified program {1:ProgramName} does not exist.

Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.

Cause/Solution:

- Check the name of the program.
- Import the program using MpCodeBoxManager.

Additional information

- {1:ProgramName}: Name of the specified program

Constant:

```plaintext
mpCODEBOX_ERR_PROGRAM_MISSING
```
These function blocks / functions can report this error:

- MpCodeBoxCore
- MpCodeBoxManager
- MpCodeBoxProgramControl

1.4.3.18 -1064169472: Program already exists

Description:
The program already exists.

Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.

Cause/Solution:
- Change the name of the program.
- Set input parameter "Overwrite" from MpCodeBoxManager to TRUE.

Constant:
mpCODEBOX_ERR_PROGRAM_EXISTS

These function blocks / functions can report this error:

- MpCodeBoxCore
- MpCodeBoxManager
- MpCodeBoxProgramControl

1.4.3.19 -1064239091: Invalid configuration

Description:
Could not read configuration while creating components. See the Logger for additional information.

Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.

Cause/Solution:
- The configuration is damaged.
- The function is not enabled in the configuration.

These function blocks / functions can report this error:

- MpCodeBoxCore
- MpCodeBoxManager
- MpCodeBoxProgramControl

Constant
mpCODEBOX_ERR_CONFIG_INVALID

1.4.3.20 -1064239098: MpLink already in use

Description:
This MpLink is already in use.
Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.

Cause/Solution:
- If "Overload" is the selected download mode, then the mapp component should be disabled in the exit subroutine with "Enable" = FALSE. It is also possible to use "Copy" or "One cycle" mode. In these cases, it is not necessary to disable the mapp component.
- Check which components are already using this MpLink.
- Create a new MpLink.

These function blocks / functions can report this error:
- MpCodeBoxCore
- MpCodeBoxProgramControl

Constant
mpCODEBOX_ERR_MPLINK_IN_USE

1.4.3.21 -1064239099: Invalid MpLink contents

Description:
The value of variable "MpLink" on the function block input is invalid.

Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.

Cause/Solution:
- Writing to the values in structure MpComIdentType is not permitted.
- A corresponding configuration for this mapp component must be available (see first paragraph of the description for the functions / function blocks listed below).

These function blocks / functions can report this error:
- MpCodeBoxCore
- MpCodeBoxProgramControl

Constant
mpCODEBOX_ERR_MPLINK_CORRUPT

1.4.3.22 -1064239100: MpLink modified

Description:
The value on input "MpLink" was modified while the components were running ("Enable" = TRUE).

Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.

Cause/Solution:
- The value of input "MpLink" can only be changed while the component is inactive ("Enable" = FALSE).

These function blocks / functions can report this error:
Constant

mpCODEBOX_ERR_MPLINK_CHANGED

1.4.3.23 -1064239101: MpLink connection not permitted

Description:
The value on input "MpLink" is not allowed.

Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.

Cause/Solution:

• The component is connected directly to mpCOM_MAIN or mpCOM_STANDALONE. This is not supported.

These function blocks / functions can report this error:

• MpCodeBoxCore
• MpCodeBoxProgramControl

Constant

mpCODEBOX_ERR_MPLINK_INVALID

1.4.3.24 -1064239102: MpLink is null pointer

Description:
Input "MpLink" is not connected, null pointer.

Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.

Cause/Solution:

• Check input "MpLink" on the function block.

These function blocks / functions can report this error:

• MpCodeBoxCore
• MpCodeBoxProgramControl

Constant

mpCODEBOX_ERR_MPLINK_NULL

1.4.3.25 -1064239103: Could not create component

Description:
The mapp component could not be created and is not enabled. See the Logger for additional information.

Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.
Cause/Solution:

- Service for configuring the function block not available -> Problem with MpCom
- Unable to read registry -> Problem with MpCom
- Details about the cause of error in the logger

These function blocks / functions can report this error:

- MpCodeBoxCore
- MpCodeBoxManager
- MpCodeBoxProgramControl

Constant

mpCODEBOX_ERR_ACTIVATION

1.4.3.26 -1064299408: Error in interpreter

Description:
The program cannot be executed due to an error in the interpreter.

Cause/Solution:

- Syntax error in the program
- See the Logger for additional information.

1.4.3.27 -1064299409: Invalid node index

Description:
The program contains an invalid node index.

Cause/Solution:

- Check the program.
- The program was manipulated externally.

1.4.3.28 -1064299410: Error in program code

Description:
The program contains invalid code in {2:LineNumber}.

Cause/Solution:

- Check the program.

Additional information

- {2:LineNumber}: Position of the error in the program

1.4.3.29 -1064299411: Unexpected end of program

Description:
The program ended unexpectedly and cannot be executed.

Cause/Solution:

- Check the program.
- Tag "END_PROGRAM" is missing in the program.
1.4.3.30 -1064299413: Could not create program

Description:
Program \{2:FileName\} could not be created on data storage device \{3:DeviceName\}. (\{1: ErrorNumber\})

Cause/Solution:
- Use of nonexistent macros.
- Check the macros.
- See the Logger for additional information.

Additional information
- \{1:ErrorNumber\}: Error number.
- \{2:FileName\}: Program that could not be created.
- \{3:DeviceName\}: Data storage device.

1.4.3.31 -1064299414: Program save failed

Description:
The program could not be saved.

Cause/Solution:
- Check the data storage device.
- See the Logger for additional information.

1.4.3.32 -1064299415: Program manager initialization failed

Description:
The program manager manages all programs.
Initialization of the program manager failed. (\{1:ErrorNumber\})

Cause/Solution:
- For additional information, see the additional error numbers and Logger.

Additional information
- \{1:ErrorNumber\}: Error number.

1.4.3.33 -1064299416: Cannot export program

Description:
Program \{2: ProgramName\} could not be exported. (\{1: ErrorNumber\})

Cause/Solution:
- Check the program name.
- Check the data storage device.

Additional information
- \{1:ErrorNumber\}: Error number.
- \{2:ProgramName\}: Program that could not be exported.
1.4.3.34 -1064299417: Could not delete program

Description:
Program {2: ProgramName} could not be deleted. ({{1: ErrorNumber}})

Cause/Solution:
- Check the program name.
- Check whether the program file exists.

Additional information
- {1:ErrorNumber}: Error number.
- {2:ProgramName}: Program that could not be deleted.

1.4.3.35 -1064299418: Could not import macro

Description:
Macro {2: MacroName} could not be imported. ({{1:ErrorNumber}}).

Cause/Solution:
- Check the macro to be imported.

Additional information
- {1:ErrorNumber}: Error number.
- {2: MacroName}: Macro that could not be imported.

1.4.3.36 -1064299419: Could not create macro

Description:
Macro {2: MacroName} could not be created by the configuration. ({{1: ErrorNumber}})

Cause/Solution:
- Check the macro.

Additional information
- {1:ErrorNumber}: Error number.
- {2: MacroName}: Macro that could not be created.

1.4.3.37 -1064299420: Error in macro

Description:
The program contains an invalid macro.

Cause/Solution:
- Check the macro.

1.4.3.38 -1064299421: Sandbox variable not specified

Description:
No sandbox variable was specified.

Cause/Solution:
- Specify a sandbox variable in the MpCodeBoxCore configuration.
1.4.3.39 -1064299422: Argument not found

Description:
The argument being used is missing.

Cause/Solution:
- Check the argument.
- Add the argument in the configuration.

1.4.3.40 -1064299423: Macro ID already exists

Description:
The imported/added macro ID already exists.

Cause/Solution:
- Change the macro ID.

1.4.3.41 -1064299424: Access to configuration failed

Description:
Could not access section {3: SectionName} in configuration {2: ConfigurationName}.

Cause/Solution:
- Check the configuration.

Additional information
- {2: ConfigurationName}: Name of the configuration.
- {3: SectionName}: Name of the section.

1.4.3.42 -1064299425: Invalid entry

Description:
Enter {2:GroupName} / {3:EntryName} / {4:TypeName} is invalid (cause of error: {1:ErrorNumber}).

Cause/Solution:
- Check the specified entry.
- Adjust the entry.

Additional information
- {1:ErrorNumber}: Cause of error
- {2:GroupName}: Name of the section in the configuration that contains the entry.
- {3:EntryName}: Invalid entry
- {4:TypeName}: Type used for the entry

1.4.3.43 -1064299426: Missing entry

Description:
Enter {2:GroupName} / {3:EntryName} is missing in the configuration.

Cause/Solution:
- Add the missing entry in the configuration.
Additional information

- (2:GroupName): Name of the section in the configuration where the entry is missing.
- (3:EntryName): Missing entry

1.4.3.44 -1064299427: Unknown macro in import file

Description:
The import file contains an unknown macro.

Cause/Solution:
- Create the unknown macro.
- Import the unknown macro.

1.4.3.45 -1064299428: Invalid dependency

Description:
The program contains an invalid dependency. An incorrect chaining is in the program. The program structure is no longer correct.

Cause/Solution:
- Check the program.
- Check whether the program was manipulated externally.

1.4.3.46 -1064299429: Invalid time format

Description:
The program contains an invalid time format.

Cause/Solution:
- Specify an invalid time format.
- Check if the time of function block "TON" is set to "5" instead of "T#5s", for example.

1.4.3.47 -1064299430: Program name not found

Description:
No program name was found in the program.

Cause/Solution:
- Specify a program name.

1.4.3.48 -1064299431: Invalid variable specification

Description:
The variable scope in the program is invalid. See Macros for valid variable scopes.

Cause/Solution:
- Change the variable scope.

1.4.3.49 -1064299432: Invalid coil type

Description:
A coil in the program is invalid.
Cause/Solution:
• Specify a valid coil.

1.4.3.50 -1064299433: Invalid contact

Description:
A contact in the program is invalid.

Cause/Solution:
• Specify a valid contact.

1.4.3.51 -1064299434: Invalid node type

Description:
An invalid node type was specified.

Cause/Solution:
• Specify a valid node type such as contact or coil.

1.4.3.52 -1064299435: Syntax error

Description:
Syntax error: Expected tag {2:TagName}

Cause/Solution:
• Specify the expected tag, such as ";" or "THEN".

Additional information
• {2:TagName}: Expected tag

1.4.3.53 -1064299436: Variable already exists

Description:
The specified variable already exists.

Cause/Solution:
• Change the variable name.

1.4.3.54 -1064299437: Variable not declared

Description:
Variable {2:Name} is not declared in the program.

Cause/Solution:
• Check the variable name.
  • Declare the variable in the program.

Additional information
• {2:Name}: Name of the variable
1.4.3.55 -1064299438: Process variable not found

Description:
Process variable (2:Name) does not exist on the target system.

Cause/Solution:
- Check whether the process variable was specified correctly.
- Create the process variable.
- Call the process variable in the project.

Additional information
- (2:Name): Name of the process variable

1.4.3.56 1083314193: Waiting on function block

Description:
Function block MpCodeBoxCore must be enabled.

Reaction:
The function block indicates an active error on outputs "StatusID" and "Error". No other functions are available during this time.

Cause/Solution:
- Wait until MpCodeBoxCore is active.

Constant:
mpCODEBOX_INF_WAIT_CORE_FB

These function blocks / functions can report this error:
- MpCodeBoxCore
- MpCodeBoxManager
- MpCodeBoxProgramControl

1.4.4 Alarms

1.4.4.1 mpCODEBOX_ALM_IMPORT_FAILED: Import failed

Description:
Import of the program or macro failed.

Reaction:
A mapp alarm is triggered if a MpAlarmX component is active.

Cause/Solution:
- Check whether the program/macro already exists.
- Set input variable "Overwrite" on function block MpCodeBoxManager to TRUE.

Behavior
Edge alarm

1.4.4.2 mpCODEBOX_ALM_START_FAILED: Program start failed

Description:
Starting the program failed.
Reaction:
A mapp alarm is triggered if a MpAlarmX component is active.

Cause/Solution:
- Check whether the program exists.
- Import the program using MpCodeBoxManager.
- Check whether the program is executable.

Behavior
Edge alarm