

Project automation

straton user guide – Rev. 4

sales@straton-plc.com



straton



STRATON AUTOMATION, All Rights Reserved

The information contained in this document is the property of STRATON AUTOMATION. The distribution and/or reproduction of all or part of this document in any form whatsoever is authorized only with the written authorization of STRATON AUTOMATION. The technical data are used only for the description of the product and do not constitute a guarantee of quality in the legal sense of the term. We reserve the right to make technical changes.

Content

1. OVERVIEW	4
2. GUIDELINES.....	4
2.1. Creating a new Project Automation script.....	5
2.2. Developing and testing the script.....	6
2.3. Registering the script as a "New project" wizard	7
2.4. Registering the script as a tool.....	7
2.5. Running a script from other applications.....	7
3. REFERENCE.....	8
3.1. Project level services.....	8
3.1.1. paCreateProject - Create the target project	8
3.1.2. paCreateProjectFromTemplate - Create the target project from a template	8
3.1.3. paOpenProject - Open the target project for modification	9
3.1.4. paSetProjectComment - Change the description of the target project	9
3.1.5. paAddLibrary - Add a link to a library.....	9
3.1.6. paSetOption - Set a project option.....	10
3.2. Declaring objects	11
3.2.1. Declaring Programs	11
3.2.2. Declaring data structures	15
3.2.3. Declaring variables	17
3.2.4. Declaring IO boards	23
3.3. Generating documents.....	26
3.3.1. Common file services	26
3.3.2. Text file writing services	28
3.3.3. IEC Source code file writing services	29
3.4. Networking and fieldbuses - File writing services.....	37
3.4.1. MODBUS.....	37
3.4.2. AS-i.....	40
3.4.3. Fieldbus configurations	44
3.4.4. Binding.....	53
3.4.5. Watch window documents - file writing services	59
3.4.6. Resource documents - file writing services.....	61
3.5. Templates	63
3.5.1. Variable keywords for copying templates.....	63
3.5.2. Script parameters	64
3.5.3. Miscellaneous.....	66

1. Overview

The workbench enables to automate the creation or change of your IEC61131-3 projects. Many powerful applications are possible by writing automation scripts:

- ▶ Create new wizards to build the skeleton of a new project.
- ▶ Create wizards that automate or import the I/O configuration.
- ▶ Automatically generate documentation about project items.
- ▶ Any import / export procedure.

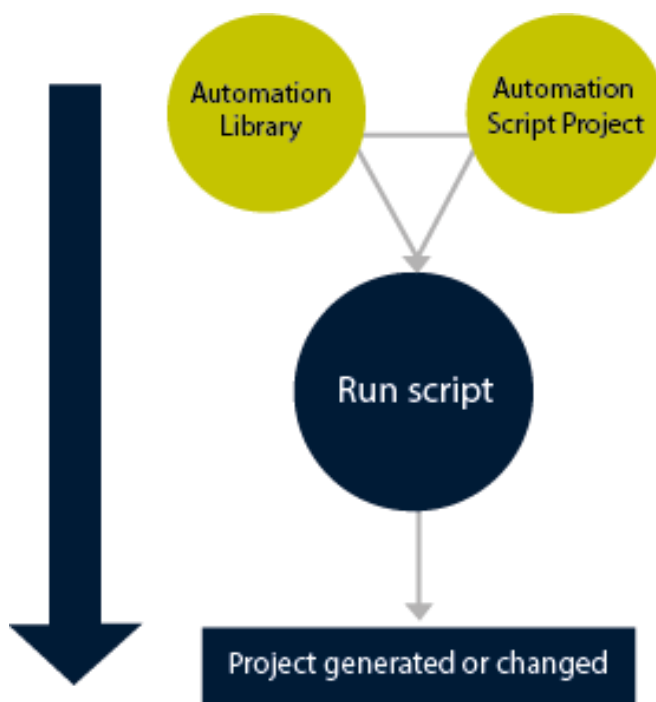
For that, no need of extra tool, and no need for you to get knowledge of other programming or scripting languages such as VB or C++. Simply use the Workbench to develop, test and run your scripts. Scripts are written using IEC languages (typically Structured Text)

2. Guidelines

The workbench enables to automate the creation or change of your IEC61131-3 projects. Many powerful applications are possible by writing automation scripts using ST language. This page explains the principle of project automation tools.

Principle

A project automation script is a program written in IEC languages. Structured Text is typically used as the programming language for scripts as it is the most adapted to automation feature. The script is developed as a project, linked to a dedicated library called AUTOMATION:

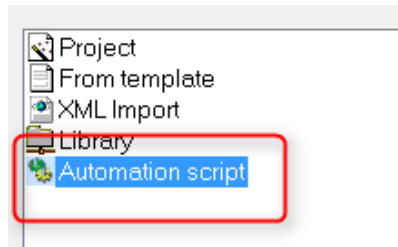


When the Workbench is used for the development of a project automation script, the 'Simulation' and 'On Line' commands are replaced by an 'Execute Script' command used for executing the script.

Unlike other IEC projects, the program of a script is executed only once, and is not repeated from cycle to cycle. Scripts can be used either for generating the skeleton of a new project, or for changing / completing an existing project. Thus the AUTOMATION library does not only contain functions for building the project, but also for enumerating and changing the existing items of a project.

2.1. Creating a new Project Automation script

To create a new Project Automation script, run the File / New Project command from the menu, and then select the `Automation script` choice in the project creation dialog box:



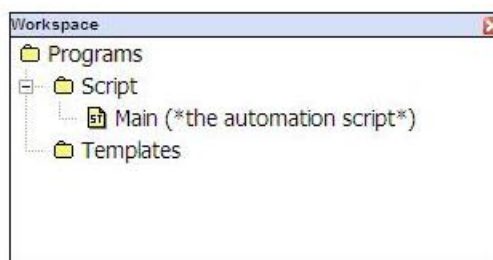
Then follow the instructions. You must specify how your script will be used:

- ▶ Generating a new project from scratch.
- ▶ Modifying an existing project.
- ▶ Modifying a project or creating it if not yet existing.

While creating the script, you already can define some parameters to be entered by the user when the script will be launched. The system will automatically create:

- ▶ Some global variables in your script project that will be the parameters.
- ▶ A list of variables grouping parameters.
- ▶ The few lines of ST code for prompting the user to enter parameters.

The script is generated with the following workspace:



The `main` program is the automation script. You can freely create sub-programs and UDFBs to be called by this program.

The `Templates` folder will contain all your template programs to be instantiated (copied) to the target project when the script is run.

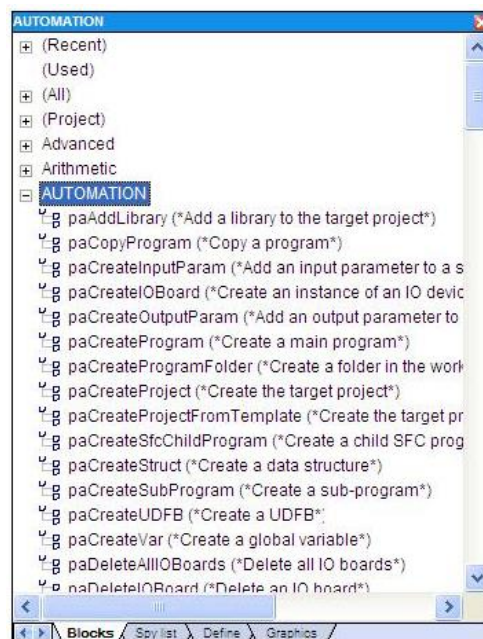
It is recommended to keep only one top level program for the script (the `main` program).

2.2. Developing and testing the script

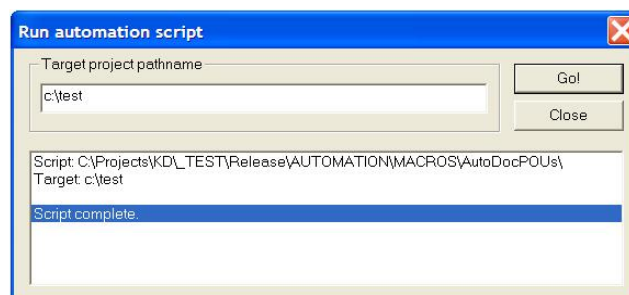
An Automation Script is developed as any project with the Workbench, but with some particular constraints. The biggest difference is that a script is a program (generally written in ST language) that is executed only once, and not repeated from cycle to cycle.

The `main` program is the automation script. You can freely create sub-programs and UDFBs to be called by this program, but it is recommended to keep only one top level program for the script (the `main` program).

The script project refers to a special library called AUTOMATION that contains all necessary functions for developing scripts:



When developing a script project, the commands 'Simulate' and 'On Line' of the Project menu are replaced by the 'Execute Script' command. This command enables to run your script for test purpose. It opens the following box:

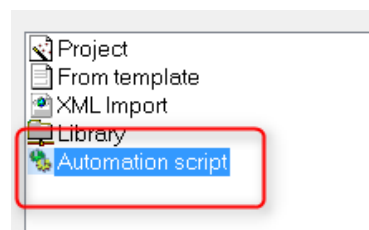


Enter the name of the target project (to be created or changed) in the upper box and press **Go!**. Any trace message or error report is displayed in the list below.

2.3. Registering the script as a “New project” wizard

If your script is intended to create the skeleton for a new project, you can register it so that the workbench proposes it in its standard New Project procedure. For that you simply need to copy the script folder under the AUTOMATION\SCRIPTS\ folder where the STRATON application data is installed.

Then your script will be available for creating a new project from the workbench. For that the user will have to select the Automation script choice in the **New Project** dialog box:



Your script must be compiled in order to be available as a wizard.

2.4. Registering the script as a tool

If your script is intended to modify an open project, you can register it so that the workbench proposes it from the Tools / Automation Script menu command. For that you simply need to copy the script folder under the AUTOMATION\MACROS\ folder where the straton application data is installed. Then your script will be available for modifying a project open with the workbench.

2.5. Running a script from other applications

Scripts developed with the workbench can be run from external applications. Below are possible calling mode:

From the command line :

Scripts can be executed directly from the Windows console or from .BAT files, using the K5Script.exe utility program installed with the Workbench. Syntax:

```
K5Script.exe <script> <target_project>
```

Arguments:

<script> : Name of the folder containing the script. If no pathname is specified, the script folder is searched under the AUTOMATION\SCRIPTS\.

<target_project> : full qualified pathname of the target project to be created or modified by the script.

From "C / C++" applications, your "C/C++" applications can link to the K5NETAS.DLL located with the installed Workbench and use the following export function to run a script:

From "C" language:

```
LPCSTR __declspec(dllimport) K5NETAS_RunScript (
    LPCSTR szScript, LPCSTR szDest
```

```
);
```

From "C++" language:

```
extern "C" LPCSTR __declspec(dllimport) K5NETAS_RunScript (
    LPCSTR szScript, LPCSTR szDest
);
```

Arguments:

szScript : Name of the folder containing the script. If no pathname is specified, the script folder is searched under the "AUTOMATION\SCRIPTS\" folder where the Workbench is installed.

szDest : full qualified pathname of the target project to be created or modified by the script

Return value:

The function returns a pointer to a string containing the report.

3. Reference

3.1. Project level services

3.1.1. paCreateProject - Create the target project

OK := paCreateProject ()

Parameters:

OK : BOOL;	TRUE if successful
------------	--------------------

This function creates the target project as an empty project. The destination folder should not exist yet. This function is typically called in a script that creates the skeleton of a new project.

You should call this function at the beginning of your script before any other declaration or creation function. Your script should terminate if this function returns FALSE.

3.1.2. paCreateProjectFromTemplate - Create the target project from a template

OK := paCreateProjectFromTemplate (TEMPLATE)

Parameters:

TEMPLATE : STRING;	Name of a template projects.
OK : BOOL;	TRUE if successful.

This function creates the target project as a copy of a template project. The destination folder should not exist yet. This function is typically called in a script that creates the skeleton of a new project. Template projects are located under the TEMPLATE\ folder of the installed Workbench. The TEMPLATE parameter specifies only the name of the template project, and does not include its path. You should call this function at the beginning of your script before any other declaration or creation function. Your script should terminate if this function returns FALSE.

3.1.3. paOpenProject - Open the target project for modification

OK := paOpenProject (CREATE)

Parameters:

CREATE : BOOL;	If TRUE, creates the project if not existing yet.
OK : BOOL;	TRUE if successful.

This function opens the target project as an empty project. If the CREATE parameter is FALSE, the project must exist. If CREATE is TRUE and the project does not exist yet, then it is created.

You should call this function at the beginning of your script before any other declaration or creation function. Your script should terminate if this function returns FALSE.

3.1.4. paSetProjectComment - Change the description of the target project

OK := paSetProjectComment (COMM)

Parameters:

COMM : STRING;	Project description.
OK : BOOL;	TRUE if successful.

This function sets the comment text used for describing the project. This text is displayed in the title bar of the Workbench when the project is open. Note that in case of a new project wizard, a comment text has already been entered by the user.

3.1.5. paAddLibrary - Add a link to a library

OK := paAddLibrary (PATH)

Parameters:

PATH : STRING;	Full qualified pathname of the library folder.
OK : BOOL;	TRUE if successful.

This function adds a link to a library for the target project.

3.1.6. paSetOption - Set a project option

OK := paSetOption (OPTION, VALUE)

Parameters:

OPTION : STRING;	Option name.
VALUE : STRING;	Value of the option.
OK : BOOL;	TRUE if successful.

This function sets an option in the project settings. The following options are available:

Name	Value	Description
TARGET	T5RTM / T5RTI	Type of target runtime.
DEBUG	ON / OFF	Compile in Debug mode.
CTSEG	ON / OFF	Complex variables in a separate segments.
FBDFLOW	ON / OFF	Color FBD lines during debug.
EMBEDSYMBOLS	ON / OFF	Embed all variable symbols.
EMBEDSYBCASE	ON / OFF	Keep case of embedded symbols.
WARNING	ON / OFF	Display warning messages.
SFCSAFE	ON / OFF	Check SFC charts safety.
SPCLEAN	ON / OFF	Remove code of uncalled sub-programs.
IECCHECK	ON / OFF	Check conformity to IEC standard.
SAFEARRAY	ON / OFF	Check array bounds at runtime.
LARGEJUMP	ON / OFF	Allow large jump instructions.
LOCK	IO / ALL / NONE	Range of lockable variables.
PASSWORD	number	Password for connection.

3.2. Declaring objects

3.2.1. Declaring Programs

PACREATEPROGRAM - CREATE A MAIN PROGRAM

OK := paCreateProgram (NAME, LANGUAGE)

Parameters:

NAME : STRING;	Name of the new program.
LANGUAGE : STRING;	Programming language (see notes).
OK : BOOL;	TRUE if successful.

This function creates a main program in the target project. Main programs are arranged in the cycle in the same order you create them from your script. The following values are predefined for the LANGUAGE parameter:

Value	Description
_LG_SFC	Sequential Function Chart.
_LG_FBD	Function Block Diagram.
_LG_LD	Ladder Diagram.
_LG_ST	Structured Text.
_LG_IL	Instruction List.

PACREATESUBPROGRAM - CREATE A SUB-PROGRAM

OK := paCreateSubProgram (NAME, LANGUAGE)

Parameters:

NAME : STRING;	Name of the new program.
LANGUAGE : STRING;	Programming language (see notes).
OK : BOOL;	TRUE if successful.

This function creates a sub-program in the target project. Values are predefined for the LANGUAGE parameter (see paCreateProgram)

PACREATEUDFB - CREATE A USER DEFINED FUNCTION BLOCK

OK := paCreateUDFB (NAME, LANGUAGE)

This function creates a User Defined Function Block in the target project. Values are predefined for the LANGUAGE parameter (see paCreateProgram)

PACREATESFCCHILDPROGRAM - CREATE A CHILD SFC PROGRAM

OK := paCreateUDFB (NAME, PARENT)

Parameters:

NAME : STRING;	Name of the new child program.
PARENT : STRING;	Name of the parent program.
OK : BOOL;	TRUE if successful.

This function creates a new SFC program to be a child of the specified parent SFC program. The parent SFC program must exist before calling this function.

PACOPYPROGRAM - DUPLICATE A PROGRAM

OK := paCopyProgram (SRC, DST)

Parameters:

SRC : STRING;	Name of the source program.
DST : STRING;	Destination program.
OK : BOOL;	TRUE if successful.

PASETPROGRAMCOMMENT - SET THE DESCRIPTION A PROGRAM

OK := paSetProgramComment (NAME, COMM)

Parameters:

NAME : STRING;	Name of the program.
COMM : STRING;	Description text.
OK : BOOL;	TRUE if successful.

This function sets the comment text of a program. It may be used for either programs or sub-programs or User Defined Function Blocks (UDFBs).

PAENUMPROGRAM - ENUMERATE PROGRAMS OF THE TARGET PROJECT

Function block: Inst_paEnumProgram (LOAD, CHECK, ITEM, FILTER)

Inputs:

LOAD : TRUE;	If TRUE, load the list of programs.
CHECK : BOOL;	If TRUE, open a checklist box to select some of the loaded programs.
ITEM : DINT;	Index of the item wanted on input (1 based).
FILTER : STRING;	Filtering mask for loading programs.

Outputs:

NB : DINT;	Number of programs.
Q : STRING;	Name of the item selected with ITEM.

This function block is used for enumerating the programs, sub-programs and UDFBs of the target project. You must first call it with LOAD input at TRUE in order to load the list of programs. You get the number of programs in the NB output. Then call it again with LOAD at FALSE and specifying the index of the wished program in the ITEM input to get the name of the selected item in the Q output. The numbering of items starts at 1.

When loading programs (LOAD inputs is TRUE), you can specify a filtering string that may include some '*' or '?' wildchars.

After loading, the block can be called again with the CHECK input at TRUE for opening a dialog box where the user can check wished programs. After the box is closed, only checked items remain in the loaded list.

Example:

```
// "ENU" is a declared instance of paEnumProgram function block
// load all programs
ENU (TRUE, FALSE, 0, '*');
// open a dialog box for the user to check wished items
ENU (FALSE, TRUE, 0, '');
// enumerate checked items
for i := 1 to ENU.NB do
    // select the item "i"
    ENU (FALSE, FALSE, i, '');
    // get the name of the item specified by "i"
    sProgName := ENU.Q;
end_for;
```

PAGETPROGRAMDESC - GET INFORMATION ABOUT A PROGRAM

Function block: Inst_paGetProgramDesc (NAME)

Inputs:

NAME : STRING;	Name of the program.
----------------	----------------------

Outputs:

OK : BOOL;	TRUE if successful.
KIND : STRING;	Kind of POU.
LANGUAGE : STRING;	Programming language.
PARENT : STRING;	Parent SFC program or empty string.
COMMENT : STRING;	Description text.

This function block is for getting information about a program of the target project. It can be used for programs, sub-programs or UDFBs. Values are predefined for the LANGUAGE parameter (see paCreateProgram).

The following values are predefined for the KIND output:

<i>Value</i>	<i>Description</i>
_POU_MAIN	Main program.
_POU_SP	Sub-program.
_POU_UDFB	User Defined Function Block.
_POU_CHILDOF	Child SFC program (the name of its parent program is in the PARENT output).

PADELETEPROGRAM - DELETE A PROGRAM

OK := paDeleteProgram (NAME)

Parameters:

NAME : STRING;	Name of the program (see notes).
OK : BOOL;	TRUE if successful.

This function deletes the specified program. It can be used for programs, sub-programs of UDFBS. The NAME parameter can contain '?' and '*' wildchars. For instance, paDeleteProgram (*) removes all the POUs of the target project. If the specified program is a SFC program having child programs, its children are deleted as well.²

3.2.2. Declaring data structures**PACREATESTRUCT - CREATE A DATA STRUCTURE**

OK := paCreateStruct (NAME)

Parameters:

NAME : STRING;	Name of the structure.
OK : BOOL;	TRUE if successful.

This function creates a data structure in the target project.

PASETSTRUCTCOMMENT - SET THE DESCRIPTION A STRUCTURE

OK := paSetStructComment (NAME, COMM)

Parameters:

NAME : STRING;	Name of the structure.
COMM : STRING;	Description text.
OK : BOOL;	TRUE if successful.

This function sets the comment text of a data structure.

PAENUMSTRUCT - ENUMERATE DATA STRUCTURES OF THE TARGET PROJECT

Function block: Inst_paEnumStruct (LOAD, CHECK, ITEM, FILTER)

Inputs:

LOAD : TRUE;	If TRUE, load the list of structures.
CHECK : BOOL;	If TRUE, open a checklist box to select some of the loaded structures.
ITEM : DINT;	Index of the item wanted on input (1 based).
FILTER : STRING;	Filtering mask for loading structures.

Outputs:

NB : DINT;	Number of structures.
Q : STRING;	Name of the item selected with ITEM.

This function block is used for enumerating the data structures of the target project. You must first call it with LOAD input at TRUE in order to load the list of structures. You get the number of structures in the NB output. Then call it again with LOAD at FALSE and specifying the index of the wished structure in the ITEM input to get the name of the selected item in the Q output.

The numbering of items starts at 1. When loading structures (LOAD inputs is TRUE), you can specify a filtering string that may include some '*' or '?' wildchars. After loading, the block can be called again with the CHECK input at TRUE for opening a dialog box where the user can check wished structures. After the box is closed, only checked items remain in the loaded list.

Example:

```
// "ENU" is a declared instance of paEnumStruct function block
// load all structures
ENU (TRUE, FALSE, 0, '*');
// open a dialog box for the user to check wished items
ENU (FALSE, TRUE, FALSE, 0, '');
// enumerate checked items
for i := 1 to ENU.NB do
    // select the item "i"
    ENU (FALSE, FALSE, i, '');
    // get the name of the item specified by "i"
    sProgName := ENU.Q;
end_for;
```


PAGETSTRUCTDESC - GET INFORMATION ABOUT A DATA STRUCTURE

Function block: Inst_paGetStructDesc (NAME)

Inputs:

NAME : STRING;	Name of the structure.
----------------	------------------------

Outputs:

OK : BOOL;	TRUE if successful.
COMMENT : STRING;	Description text.

This function block is for getting information about a structure of the target project.

PADELETESTRUCT - DELETE A DATA STRUCTURE

OK := paDeleteStruct (NAME)

Parameters:

NAME : STRING;	Name of the structure (see notes).
OK : BOOL;	TRUE if successful.

This function deletes the specified structure. The NAME parameter can contain '?' and '*' wildchars. For instance, paDeleteStruct (*) deletes all the structures of the target project.

3.2.3. Declaring variables**PACREATEVAR - CREATE A VARIABLE**

OK := paCreateVar (SNAME, STYPE)

Parameters:

SNAME : STRING;	Name of the new variable and specification of its group (see naming conventions).
STYPE : STRING;	Data type.
OK : BOOL;	TRUE if successful.

This function creates a program with the specified name in the specified group. The STYPE indicates its data type. It can be a type of function block for declaring an instance. In case of a STRING type, the type

must be followed by the string length. e.g. 'STRING(255)'. The same function can be used to create an item in a data structure.

PACREATEINPUTPARAM - CREATE AN INPUT PARAMETER OF A POU

OK := paCreateInputParam (SNAME, STYPE)

Parameters:

SNAME : STRING;	Name of the new variable in form 'POUName.VarName' (see naming conventions).
STYPE : STRING;	Data type.
OK : BOOL;	TRUE if successful.

This function creates an input parameter for the POU (sub-program or UDFB) specified as a prefix in the variable name. The STYPE indicates its data type. In case of a STRING type, the type must be followed by the string length. e.g. 'STRING(255)'.

PACREATEOUTPUTPARAM - CREATE AN OUTPUT PARAMETER OF A POU

OK := paCreateOutputParam (SNAME, STYPE)

Parameters:

SNAME : STRING;	Name of the new variable in form 'POUName.VarName' (see naming conventions).
STYPE : STRING;	Data type.
OK : BOOL;	TRUE if successful.

This function creates an output parameter for the POU (sub-program or UDFB) specified as a prefix in the variable name. The STYPE indicates its data type. In case of a STRING type, the type must be followed by the string length. e.g. 'STRING(255)'.

PASETVAR1DIM / PASETVAR2DIMS / PASETVAR3DIMS - SET THE DIMENSION(S) OF A VARIABLE (ARRAY)

OK := paSetVarDim (NAME, DIM) // 1 dimension

OK := paSetVar2Dims (NAME, DIMHIGH, DIMLOW) // 2 dimensions

OK := paSetVar3Dims (NAME, DIMHIGH, DIMMEDIUM, DIMLOW) // 3 dimensions

Parameters:

NAME : STRING;	Name of the variable and specification of its group (see naming conventions).
DIMxxx : DINT;	Dimension(s) of the array.
OK : BOOL;	TRUE if successful.

This function sets the dimension of a declared variable, in order to declare an array. The same function can be used for an item in a data structure.

PASETVARINITVALUE - SET THE INITIAL VALUE OF A VARIABLE

OK := paSetVarInitValue (NAME, VALUE)

Parameters:

NAME : STRING;	Name of the variable and specification of its group (see naming conventions).
VALUE : STRING;	Initial value in IEC 61131-3 syntax.
OK : BOOL;	TRUE if successful.

This function sets the initial value of a declared variable. The same function can be used for an item in a data structure.

PASETVARCOMMENT - SET THE DESCRIPTION A VARIABLE

OK := paSetVarComment (NAME, COMM, TAG)

Parameters:

NAME : STRING;	Name of the variable and specification of its group (see naming conventions).
COMM : STRING;	Description text.
TAG : STRING;	Short description text.
OK : BOOL;	TRUE if successful.

This function sets the comment text of a declared variable. The same function can be for an item in a data structure.

PAEMBEDVARSYMBOL - EMBED THE SYMBOL OF A VARIABLE

OK := paEmbedVarSymbol (NAME, SYB)

Parameters

NAME : STRING;	Name of the variable and specification of its group (see naming conventions).
SYB : BOOL;	TRUE if the variable symbol must be embedded.
OK : BOOL;	TRUE if successful.

This function specifies if the symbol of a declared variable must be embedded. The same function can be for an item in a data structure.

PAPROFILEVAR - DEFINES THE PROFILE AND EMBEDDED PROPERTIES OF A VARIABLE

OK := paProfileVar (PROFILE, PROPERTIES)

Parameters:

NAME : STRING;	Name of the variable and specification of its group (see naming conventions).
PROFILE : STRING;	Name of the profile.
PROPERTIES : STRING;	Embedded properties attached to the profile (if any).
OK : BOOL;	TRUE if successful.

This function specifies the profile and the corresponding embedded properties for a declared variable. Properties are written in form: prop=value,prop=value,...

PAENUMVAR - ENUMERATE VARIABLES OF A GROUP IN THE TARGET PROJECT

function block: Inst_paEnumVar (LOAD, CHECK, ITEM, FILTER)

Inputs:

LOAD : TRUE;	If TRUE, load the list of variables.
CHECK : BOOL;	If TRUE, open a checklist box to select some of the loaded variables.
ITEM : DINT;	Index of the item wanted on input (1 based).
FILTER : STRING;	Group specification and filtering mask for loading variables.

Outputs:

NB : DINT;	Number of variables.
Q : STRING;	Name of the item selected with "ITEM".

This function block is used for enumerating the variables of a group in the target project. You must first call it with LOAD input at TRUE in order to load the list of variables. You get the number of programs in the NB output. Then call it again with LOAD at FALSE and specifying the index of the wished variable in the ITEM input to get the name of the selected item in the Q output. The numbering of items starts at 1.

When loading programs (LOAD inputs is TRUE), you must specify the group in the FILTER parameter, possibly followed by the '.' separator and a filtering string including some '*' or '?' wildchars. No group name is required for global variables.

After loading, the block can be called again with the CHECK input at TRUE for opening a dialog box where the user can check wished variables. After the box is closed, only checked items remain in the loaded list.

Example:

```
// "ENU" is a declared instance of paEnumVar function block
// load all RETAIN variables
ENU (TRUE, FALSE, 0, 'RETAIN.*');
// open a dialog box for the user to check wished items
ENU (FALSE, TRUE, FALSE, 0, '');
// enumerate checked items
for i := 1 to ENU.NB do
    // select the item "i"
    ENU (FALSE, FALSE, i, '');
    // get the name of the item specified by "i"
    sVarName := ENU.Q;
end_for;
```

PAGETVARDESC - GET INFORMATION ABOUT A VARIABLE

function block: Inst_paGetVarDesc (NAME)

Inputs:

NAME : STRING;	Name of the variable and specification of its group (see naming conventions).
----------------	---

Outputs:

OK : BOOL;	TRUE if successful.
INPARAM : BOOL;	TRUE if it is an input parameter.
OUTPARAM : BOOL;	TRUE if it is an output parameter.
DATATYPE : STRING;	Data type.
DIMTOTAL : DINT;	Total number of items for an array.
DIMHIGH : DINT;	Highest dimension (0 for 1 or 2 dimension arrays).
DIMMEDIUM : DINT;	Medium dimension (0 for 1 dimension arrays).
DIMLOW : DINT;	Lowest dimension.
INIT : STRING;	Initial value in IEC syntax.
SYB : BOOL;	TRUE if the symbol of the variable is embedded.
PROFILE : STRING;	Name of the attached profile.
PROPS : STRING;	Embedded properties attached to the profile.
COMMENT : STRING;	Description text.
TAG : STRING;	Short description text.

This function block is for getting information about a declared variable. The same block can be used for an item in a data structure.

PADELETEVAR - DELETE A VARIABLE

OK := paDeleteVar (NAME)

Parameters:

NAME : STRING;	Name of the variable and specification of its group (see naming conventions).
OK : BOOL;	TRUE if successful.

This function deletes the specified variable. The same function can be used for an item in a data structure.

The name of the variable may contain '?' and '*' wildchars, but the group name cannot. For instance, paDeleteVar ('RETAIN.*') deletes all the retain variables.

3.2.4. Declaring IO boards**PACREATEIOBOARD - CREATE AN INSTANCE OF AN IO DEVICE**

OK := paCreateIOBoard (SLOT, DEVTYPE)

Parameters:

SLOT : DINT;	Slot number (0..255).
DEVTYPE : STRING;	Name of the type of IO device.
OK : BOOL;	TRUE if successful.

This function creates an instance of the specified IO device at the specified slot number.

PASETIOBOARDCOMMENT - SET THE DESCRIPTION AN IO DEVICE

OK := paSetIOBoardComment (SLOT, COMM)

Parameters:

SLOT : DINT;	Slot number (0..255).
COMM : STRING;	Description text.
OK : BOOL;	True if successful.

This function sets the comment text of the specified instance of an IO device.

PASETIOBOARDPARAM - SET THE VALUE OF A PARAMETER OF AN IO DEVICE

OK := paSetIOBoardComment (SLOT, COMM)

Parameters:

BOARD : STRING;	Board name (e.g. '%QX0').
PARAM : STRING;	Name of the parameter.
VALUE : STRING;	Value of the parameter.
OK : BOOL;	TRUE if successful.

This function sets the value of an IO board parameter. Refer to OEM instruction for details about available parameters.

PASETIOALIAS - DEFINE AN ALIAS FOR AN IO CHANNEL

OK := paSetIOBoardComment (SLOT, COMM)

This function defines a readable name for the specified IO channel. The alias can then be used in place of the "%" name in programs.

PADELETEIOBOARD - DELETE AN INSTANCE OF AN IO DEVICE

OK := paDeleteIOBoard (SLOT)

Parameters:

SLOT : DINT;	Slot number (0..255).
OK : BOOL;	TRUE if successful.

This function deletes the specified instance of an IO device. In case of a complex device, the whole device (with all its groups) is removed.

PADELETEALLIOBOARDS - DELETE ALL INSTANCES OF IO DEVICES

OK := paDeleteAllIOBoards ()

Parameters:

OK : BOOL;	TRUE if successful.
------------	---------------------

This function deletes all the existing instances of any IO device.

PAENUMIOBOARDS - ENUMERATE IO BOARDS OF THE TARGET PROJECT

Function block: Inst_paEnumIOBoard (LOAD, CHECK, ITEM, FILTER)

Inputs:

LOAD : TRUE;	If TRUE, load the list of boards.
CHECK : BOOL;	If TRUE, open a checklist box to select some of the loaded boards.
ITEM : DINT;	Index of the item wanted on input (1 based).

Outputs:

NB : DINT;	Number of boards.
Q : STRING;	Name of the item selected with "ITEM".

This function block is used for enumerating the IO boards of the target project. You must first call it with LOAD input at TRUE in order to load the list of IO boards. You get the number of boards in the NB output. Then call it again with LOAD at FALSE and specifying the index of the wished structure in the ITEM input to get the name of the selected item in the Q output. The numbering of items starts at 1.

For complex IO devices, only the child boards of the device are listed. The complex device itself does not occur in the list.

After loading, the block can be called again with the CHECK input at TRUE for opening a dialog box where the user can check wished boards. After the box is closed, only checked items remain in the loaded list.

Example:

```
// "ENU" is a declared instance of paEnumIOBoard function block
// load all boards
ENU (TRUE, FALSE, 0);
// open a dialog box for the user to check wished items
ENU (FALSE, TRUE, FALSE, 0);
// enumerate checked items
for i := 1 to ENU.NB do
  // select the item "i"
  ENU (FALSE, FALSE, i);
  // get the name of the item specified by "i"
  sBoardName := ENU.Q;
end_for;
```

PAGETIOBOARDDESC - GET INFORMATION ABOUT AN IO BOARD

Function block: `Inst_paGetIOBoardDesc (NAME)`

Inputs:

NAME : STRING;	Name of the board (e.g. '%QX0').
----------------	----------------------------------

Outputs:

OK : BOOL;	TRUE if successful.
NBCHANNEL : DINT;	Number of channels.
DEVNAME : STRING;	IO Device name.
DEVGROUP : STRING;	In case of a complex IO device, name of the group within the device.
COMMENT : STRING;	Description text.

This function block is for getting information about an IO board of the target project.

3.3. Generating documents

3.3.1. Common file services

PAFILEOPENWRITE - OPEN ANY FILE IN THE TARGET PROJECT FOR WRITING

FID := `paFileOpenWrite (PREFIX, SUFFIX)`

Parameters:

PREFIX : STRING;	File prefix.
SUFFIX : STRING;	File suffix.
FID : DINT;	File identifier or 0 if fail.

This function opens a target project file for writing. It should normally not be used as dedicated file open functions are available for most of possible project documents. This functions removes the contents of the file if it already exist.

When written, the file must be closed by calling the `paFileClose` function.

PAFILEOPENWRITEPROGRAMSRC - OPEN A POU SOURCE FILE FOR WRITING

FID := paFileOpenWriteProgramSrc (NAME)

Parameters:

NAME : STRING;	Program name.
FID : DINT;	File identifier or 0 if fail.

This function opens for writing the file that contains the source code of a program, sub-program or UDFB. Then you should use the appropriate file writing functions according to the language of the specified POU. This functions removes the contents of the file if it already exist.

When written, the file must be closed by calling the paFileClose function.

PAFILEOPENWRITEPROGRAMDEF - OPEN A POU DEFINITIONS FILE FOR WRITING

FID := paFileOpenWriteProgramDef (NAME)

Parameters:

NAME : STRING;	Program name.
FID : DINT;	File identifier or 0 if fail.

This function opens for writing the file that contains the local definitions of a program, sub-program or UDFB. Then you should use the text file writing functions to fill the file. This functions removes the contents of the file if it already exist.

When written, the file must be closed by calling the paFileClose function.

PAFILECLOSE - CLOSE AN OPEN FILE

OK := paFileClose (FID)

Parameters:

FID : DINT;	File identifier returned by any file open function.
-------------	---

This function closes a file open by any of available file open functions.

3.3.2. Text file writing services

PAFTEXT_EOL - WRITE END OF LINE CHARACTERS IN A TEXT FILE

OK := paFTextEol (FID)

Parameters:

FID : DINT;	File identifier returned by the file open function.
-------------	---

This function writes end of line characters in a text file open for writing.

PAFTEXTLINE - WRITE A LINE IN A TEXT FILE

OK := paFTextLine (FID, TEXT)

Parameters:

FID : DINT;	File identifier returned by the file open function.
TEXT : STRING;	Text to be written.

This function writes the string specified by the TEXT parameter plus end of line characters in a text file open for writing.

PAFTEXTSTRING - WRITE A STRING IN A TEXT FILE

OK := paFTextEol (FID, TEXT)

Parameters:

FID : DINT;	File identifier returned by the file open function.
TEXT : STRING;	Text to be written.

This function writes the string specified by the TEXT parameter in a text file open for writing.

3.3.3. IEC Source code file writing services

Use text file services for ST and IL languages.

FBD - SEQUENTIAL FILE WRITING SERVICES

The functions below enable you to write sequentially (from the top to the bottom) the source code of a program written in FBD language. It has limited features (it is not possible to link 2 blocks together) but provides a very easy way to generate FBD as you do not need to care with the position of the elements in the diagram. Files are open using the `paFileOpenWriteSrc` function. When complete, they must be closed by calling the `paFileClose` function.

Use the following functions for inserting objects sequentially:

- ▶ `paFbdsBreak` - Insert a network break.
- ▶ `paFbdsComment` - Insert a multiline comment block.
- ▶ `paFbdsLabel` - Insert a label.
- ▶ `paFbdsJump` - Insert an unconditional jump.
- ▶ `paFbdsReturn` - Insert an unconditional <RETURN> statement.
- ▶ `paFbdsBlock` - Insert a block.

Then, just after inserting a block you can define its inputs and outputs:

- ▶ `paFbdsInputVar` - Connect a variable on input of the last block.
- ▶ `paFbdsOutputVar` - Connect a variable on output of the last block.
- ▶ `paFbdsOutputJump` - Connect a jump statement on output of the last block.
- ▶ `paFbdsOutputReturn` - Connect a <RETURN> statement on output of the last block.

PAFBDSBREAK - ADD A NETWORK BREAK

OK := `paFbdsBreak` (FID, TEXT)

Parameters:

FID : DINT;	File identifier answered by <code>paFileOpenWriteSrc</code> .
TEXT : STRING;	Text written on the network break.
OK : BOOL;	TRUE if successful.

This function adds a network break under the last added item in a FBD source file open for writing.

PAFBDSCOMMENT - ADD A COMMENT BLOCK

OK := paFbdsComment (FID, TEXT, HEIGHT)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
TEXT : STRING;	Comment text.
HEIGHT : DINT;	Height of the comment block (in diagram grid unit).
OK : BOOL;	TRUE if successful.

This function adds a multiline comment block under the last added item in a FBD source file open for writing. You can use the '\$N' sequence in the comment text to specify an end of line.

PAFBDSLABEL - ADD A LABEL

OK := paFbdsLabel (FID, LABEL)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
LABEL : STRING;	Label name.
OK : BOOL;	TRUE if successful.

This function adds a label under the last added item in a FBD source file open for writing. The label is placed on the left side of the diagram.

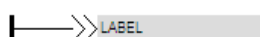
PAFBDSJUMP - ADD A UNCONDITIONAL JUMP

OK := paFbdsLabel (FID, LABEL)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
LABEL : STRING;	Target label name.
OK : BOOL;	TRUE if successful.

This function adds a jump instruction under the last added item in a FBD source file open for writing. It is an unconditional jump instruction. The jump symbol is placed on the left side of the diagram, connected to a LD left power rail (always true):



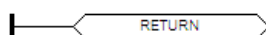
PAFBDSRETURN - ADD A UNCONDITIONAL <RETURN> SYMBOL

OK := paFbdsReturn (FID)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
OK : BOOL;	TRUE if successful.

This function adds a <RETURN> jump instruction under the last added item in a FBD source file open for writing. It is an unconditional instruction. The jump symbol is placed on the left side of the diagram, connected to a LD left power rail (always true):

**PAFBDSBLOCK - ADD A BLOCK**

OK := paFbdsBlock (FID, NAME, INSTANCE)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
NAME : STRING;	Block name.
INSTANCE : STRING;	Instance name in case of a function block.
OK : BOOL;	TRUE if successful.

This function adds a block symbol under the last added item in a FBD source file open for writing. It can be a basic operator, a function or a function block. The name of the instance must be specified in case of a function block.

Immediately after calling this function, use the FbdsInput... and FbdsOutput... functions in order to connect the input and output pins of the block.

PAFBDSINPUTVAR - CONNECT AN INPUT OF THE LAST ADDED BLOCK

OK := paFbdsInputVar (FID, NAME, PIN, NEGATE)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
NAME : STRING;	Variable name or constant expression.
PIN : DINT;	Index of the input pin of the block (the first input pin is 1).
NEGATE : BOOL;	If TRUE, the connection between the variable and the block has a boolean negation.
OK : BOOL;	TRUE if successful.

This function connects a variable box on input of the last block added by the paFbdsBlock function, and must be called immediately after.

PAFBDSOUTPUTVAR - CONNECT AN OUTPUT OF THE LAST ADDED BLOCK

OK := paFbdsOutputVar (FID, NAME, PIN, NEGATE)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
NAME : STRING;	Variable name.
PIN : DINT;	Index of the output pin of the block (the first output pin is 1).
NEGATE : BOOL;	If TRUE, the connection between the block and the variable has a boolean negation.
OK : BOOL;	TRUE if successful.

This function connects a variable box on output of the last block added by the paFbdsBlock function, and must be called immediately after.

PAFBDSOUTPUTJUMP - CONNECT A JUMP SYMBOL ON OUTPUT OF THE LAST ADDED BLOCK

OK := paFbdsOutputJump (FID, LABEL, PIN, NEGATE)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
LABEL : STRING;	Target label name.
PIN : DINT;	Index of the output pin of the block (the first output pin is 1).
NEGATE : BOOL;	If TRUE, the connection between the block and the variable has a boolean negation.
OK : BOOL;	TRUE if successful.

This function connects a jump instruction on output of the last block added by the paFbdsBlock function, and must be called immediately after.

PAFBDSOUTPUTRETURN - CONNECT A <RETURN> SYMBOL ON OUTPUT OF THE LAST ADDED BLOCK

OK := paFbdsOutputReturn (FID, PIN, NEGATE)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
PIN : DINT;	Index of the output pin of the block (the first output pin is 1).
NEGATE : BOOL;	If TRUE, the connection between the block and the variable has a boolean negation.
OK : BOOL;	TRUE if successful.

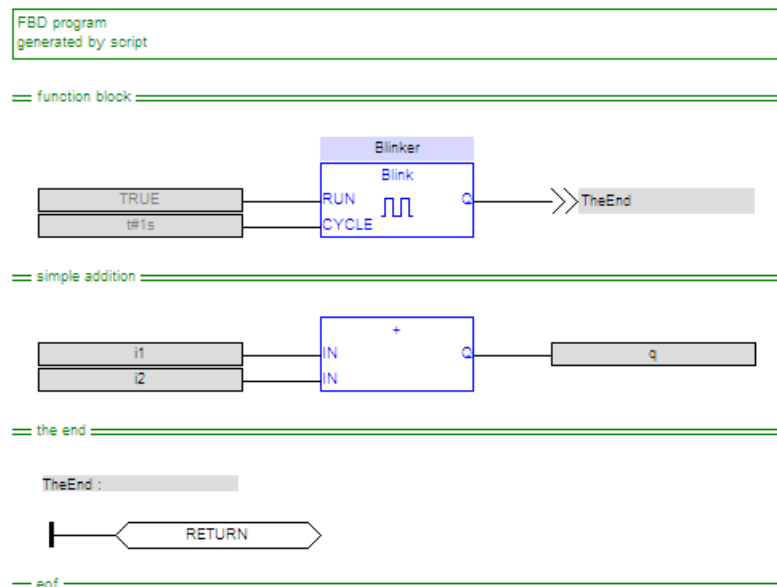
This function connects a <RETURN> instruction on output of the last block added by the paFbdsBlock function, and must be called immediately after.

Example

Below is an example of script that generates a FBD file in the target project:

```
// make FBD code - sequential writing
f := paFileOpenWriteProgramSrc ('ProgFBD');
if f <> 0 then
  // add a comment block (2 lines of text)
  paFbdsComment (f, 'FBD program$Ngenerated by script', 2);
  // add a function block and its inputs and outputs
  paFbdsBreak (f, 'function block');
  paFbdsBlock (f, 'Blink', 'Blinker');
    paFbdsInputVar (f, 'TRUE', 1, FALSE);
    paFbdsInputVar (f, 't#1s', 2, FALSE);
    paFbdsOutputJump (f, 'TheEnd', 1, FALSE);
  // add a function and its inputs and outputs
  paFbdsBreak (f, 'simple addition');
  paFbdsBlock (f, '+', '');
    paFbdsInputVar (f, 'i1', 1, FALSE);
    paFbdsInputVar (f, 'i2', 2, FALSE);
    paFbdsOutputVar (f, 'q', 1, FALSE);
  // add a label and an unconditional return instruction
  paFbdsBreak (f, 'the end');
  paFbdsLabel (f, 'TheEnd');
  paFbdsReturn (f);
  // add a break at the end of the diagram
  paFbdsBreak (f, 'eof');
  // close the file
  paFileClose (f);
end_if;
```

Here is the FBD program generated by the script:



LD - FILE WRITING SERVICES

The functions below enable you to write sequentially (from the top to the bottom) the source code of a program written in LD language. The program is written rung after rung. Items are placed sequentially on the rung from the left to the right. Files are open using the paFileOpenWriteSrc function. When complete, they must be closed by calling the paFileClose function.

Use the following functions for inserting rungs in the diagram:

- ▶ paFLdCommentLine - Add a comment line (network break).
- ▶ paFLdStartRung - Start a new rung.
- ▶ paFLdEndRung - Terminate the rung.
- ▶ paFLdDivergence - Start an OR divergence on the current rung.
- ▶ paFLdConvergence - Terminate an OR divergence on the current rung.
- ▶ paFLdNextBranch - Add a parallel branch to the current divergence.

Use the following functions for adding items on the current rung:

- ▶ paFLdContact - Add a contact.
- ▶ paFLdCoil - Add a coil.
- ▶ paFLdJump - Add a jump instruction.
- ▶ paFLdReturn - Add a <return> jump instruction.
- ▶ paFLdBlock - Add a block.
- ▶ paFLdHorzSegment - Add a horizontal segment line.

PAFLDCOMMENTLINE - ADD A NETWORK BREAK

OK := paFLdCommentLine (FID, TEXT)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
TEXT : STRING;	Text written on the network break.
OK : BOOL;	TRUE if successful.

This function adds a comment line (network break) in between rungs.

PAFLDSTARTRUNG - START A NEW RUNG

OK := paFLdStartRung (FID, LABEL)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
LABEL : STRING;	Rung label name (empty string if no label defined).
OK : BOOL;	TRUE if successful.

This function starts a new rung. After calling this function, you can call other functions to add items on the rung from the left to the right. When complete, you need to call the paFLdEndRung function to terminate the rung.

PAFLDENDRUNG - TERMINATES THE RUNG

OK := paFLdEndRung (FID)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
OK : BOOL;	TRUE if successful.

This function terminates the current rung. You need to call it before starting another rung or adding a comment line.

PAFLDDIVERGENCE - STARTS A DIVERGENCE

OK := paFLdDivergence (FID)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
OK : BOOL;	TRUE if successful.

This function starts an OR divergence on the current rung. After calling this function, items can be added on the first (top) branch of the divergence. Parallel branches can be added by calling the paFLdNextBranch function. When complete, you need to call the paFLdConvergence function to terminate the divergence.

PAFLDCONVERGENCE - TERMINATES A DIVERGENCE

OK := paFLdConvergence (FID)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
OK : BOOL;	TRUE rue if successful.

This function terminates the current OR divergence. After calling this function, items can be added after the divergence on the parent rung.

3.4. Networking and fieldbuses - File writing services**3.4.1. MODBUS****PAFILEOPENWRITEMODBUS - OPEN THE MODBUS CONFIGURATION FOR WRITING**

FID := paFileOpenWriteModbus ()

Parameters:

FID : DINT;	File identifier or 0 if fail.
-------------	-------------------------------

This function opens the MODBUS configuration file for writing. This functions removes the contents of the configuration if it already exist. When written, the file must be closed by calling the paFileClose function.

PAFMODEBUSSLAVE - SET THE MODBUS SLAVE CONFIGURATION

OK := paFModbusSlave (FID, SLAVENO)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteModbus function.
SLAVENO : DINT;	Default slave number for the MODBUS Slave server.
OK : BOOL;	TRUE if successful.

This function defines the slave number used for the MODBUS slave server.

PAFMODBUSMASTER - ADD A MODBUS MASTER PORT

OK := paFModbusMaster (FID, OPENMODBUS, ADDRESS, PORT, RECONNECT)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteModbus function.
OPENMODBUS : BOOL;	TRUE = Open Modbus protocol / FALSE = RTU protocol.
ADDRESS : STRING;	Configuration of the port or IP address.
PORT : DINT;	IP port number in case of MODBUS on ETHERNET.
RECONNECT : BOOL;	Option: tries to reconnect after error.
OK : BOOL;	TRUE if successful.

This function adds a new MODBUS master connection to the configuration.

PAFMODBUSREQUESTSLAVE - ADD A MODBUS SLAVE REQUEST

OK := paFModbusRequestSlave (FID, REQUEST, ADDRESS, NBITEM, NAME)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteModbus function.
REQUEST : STRING;	Type of request (see notes).
ADDRESS : DINT;	Base MODBUS address.
NBITEM : DINT;	Number of items (bits or words).
NAME : STRING;	Optionnal description text.
OK : BOOL;	TRUE if successful.

This function adds a new MODBUS slave request. You can define variables of the request just after calling this function. This function does not take care of address offsets defined by MODBUS. The first available address is always 0. The following values are possible for the REQUEST parameter:

_MB_I_REG	Input registers.
_MB_H_REG	Holding registers.
_MB_I_BIT	Input bits.
_MB_C_BIT	Coil bits.

PAFMODBUSREQUESTMASTER - ADD A MODBUS MASTER REQUEST

```
OK := paFModbusRequestMaster (FID, MODE, REQUEST, SLAVENO, ADDRESS,
                              NBITEM, PERIOD, TIMEOUT, TRIALS, NAME)
```

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteModbus function.
MODE : STRING;	Activation mode (see notes).
REQUEST : STRING;	Type of request (see notes).
SLAVENO : DINT;	Slave number of the device.
ADDRESS : DINT;	Base MODBUS address.
NBITEM : DINT;	Number of items (bits or words).
PERIOD : TIME;	Activation period (for periodic requests).
TIMEOUT : TIME;	Exchange timeout.
TRIALS : DINT;	Number of successive trials.
NAME : STRING;	Optionnal description text.
OK : BOOL;	TRUE if successful.

This function adds a new MODBUS master request. You can define variables of the request just after calling this function. This function does not take care of address offsets defined by MODBUS. The first available address is always 0. The following values are possible for the **MODE** parameter:

_MB_PERIODIC	Request activated periodically.
_MB_ONCALL	Request activated by a variable.
_MB_ONCHANGE	Request activated on a change of data (for "write" requests).

The following values are possible for the **REQUEST** parameter:

_MB_READ_I_REG	Read Input registers.
_MB_READ_H_REG	Read Holding registers.
_MB_WRITE_H_REG	Write Holding registers.
_MB_READ_I_BIT	Read Input bits.
_MB_READ_C_BIT	Read Coil bits.
_MB_WRITE_C_BIT	Write Coil bits.

PAFMODBUSVARIABLE - ADD A MODBUS VARIABLE

OK := paFModbusVariable (FID, MODE, OFFSET, MASK, TWOWORDS, SYMBOL)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteModbus function.
MODE : STRING;	Exchange mode (see notes).
OFFSET : DINT;	Offset in the request.
MASK : DINT;	Mask for data exchange.
TWOWORDS : BOOL;	TRUE if data stored on 2 consecutive words.
SYMBOL : STRING;	Symbol of the variable.
OK : BOOL;	TRUE if successful.

This function connects a variable to the request defined by the last call to paFileModbusRequestSlave or paFileModbusRequestMaster functions. The following values are possible for the **MODE** parameter:

_MB_DATA	Data exchange.
_MB_STATUS	The variable receives the connection status.
_MB_COMMAND	The variable is used for activating the request.
_MB_TRIAL	The variable receives the current trial number.

3.4.2. AS-i**PAFILEOPENWRITEASI - OPEN THE AS-I CONFIGURATION FOR WRITING**

FID := paFileOpenWriteASi ()

Parameters:

FID : DINT;	File identifier or 0 if fail.
-------------	-------------------------------

This function opens the AS-i configuration file for writing. This functions removes the contents of the configuration if it already exist. When written, the file must be closed by calling the paFileClose function.

PAFASIAPPLYCONFIG - SET AS-I CONFIGURATION MAIN FLAGS

OK := paFasiApplyConfig (FID, APPLY)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteASi function.
APPLY : BOOL;	If TRUE, the AS-i configuration is applied when the runtime stats.
OK : BOOL;	TRUE if successful.

PAFASIMASTER - ADD AN AS-I MASTER

OK := paFasiMaster (FID, NAME, SETTINGS)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteASi function.
NAME : STRING;	Description text.
SETTINGS : STRING;	AS-i controller configuration string.
OK : BOOL;	TRUE if successful.

This function adds a master device to the AS-i configuration. AS-i slaves and diagnostic variables for this master must be added just after calling this function.

PAFASISLAVE - ADD AN AS-I SLAVE

OK := paFasiSlave (FID, NAME, ADDRESS, PROFILE)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteASi function.
NAME : STRING;	Description text.
ADDRESS : DINT;	AS-i slave address.
PROFILE : STRING;	Slave profile written in AS-i convention: 'H.H.H.H' (hexadecimal digits).
OK : BOOL;	TRUE if successful.

This function adds a slave device under the last declared AS-i master. AS-i variables for this master must be added for this slave just after calling this function.

PAFASIMASTERDIAG - ADD AN AS-I MASTER DIAGNOSTIC VARIABLE

OK := paFasiMasterDiag (FID, NAME, DIAG)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteASi function.
NAME : STRING;	Symbol of the variable.
DIAG : DINT;	Wished diagnostic information.
OK : BOOL;	TRUE if successful.

This function adds a diagnostic variable to the last declared AS-i master. The possible values are available for the DIAG parameter:

_ASI_M_CONFOK	Configuration OK.
_ASI_M_SLV0	Slave detected at address 0.
_ASI_M_AUTOACT	Auto-addressing is active.
_ASI_M_AUTOAV	Auto-addressing is available.
_ASI_M_CNFACT	Configuration is active.
_ASI_M_NORMAL	Normal mode.
_ASI_M_PWFALL	Power fail detected.
_ASI_M_OFFLINE	Off Line mode.
_ASI_M_PERIPHOK	Periphery OK.

PAFASIINPUT - ADD AN AS-I INPUT VARIABLE

OK := paFasiInput (FID, NAME, OFFSET)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteASi function.
NAME : STRING;	Symbol of the variable.
OFFSET : DINT;	Input number (1 to 4) or 0 for all inputs in a byte.
OK : BOOL;	TRUE if successful.

This function adds an input variable to the last declared AS-i slave.

PAFASISLAVEDIAG - ADD AN AS-I SLAVE DIAGNOSTIC VARIABLE

OK := paFasiSlaveDiag (FID, NAME, DIAG)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteASi function.
NAME : STRING;	Symbol of the variable.
DIAG : DINT;	Wished diagnostic information.
OK : BOOL;	TRUE if successful.

This function adds a diagnostic variable to the last declared AS-i slave. The possible values are available for the **DIAG** parameter:

_ASI_S_ACT	Slave active.
_ASI_S_DET	Slave detected.
_ASI_S_PRJ	Slave projected.
_ASI_S_COR	Slave corrupted.
_ASI_S_FLT	Periphery fault.

PAFASIOUTPUT - ADD AN AS-I OUTPUT VARIABLE

OK := paFasiOutput (FID, NAME, OFFSET)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteASi function.
NAME : STRING;	Symbol of the variable.
OFFSET : DINT;	Output number (1 to 4) or 0 for all inputs in a byte.
OK : BOOL;	TRUE if successful.

This function adds an output variable to the last declared AS-i slave.

3.4.3. Fieldbus configurations

PAFILEOPENWRITEFIELDBUS - OPEN THE AS-I CONFIGURATION FOR WRITING

FID := paFileOpenWriteFieldbus (CONFIG)

Parameters:

CONFIG : STRING	Type of fieldbus - see notes.
FID : DINT;	File identifier or 0 if fail.

This function opens the selected fieldbus configuration file for writing. This functions removes the contents of the configuration if it already exist. When written, the file must be closed by calling the paFileClose function.

The CONFIG string is the prefix of the wizard DLL used for configuration. Available DLLs are stored in the \IOD folder where the Workbench is installed. You can also get the list of available configurations from the Prompt tab in the output window. For that, enter the following? FB command.

PAFFBCREATEMASTER - ADD A MASTER NODE

OK := paFfbCreateMaster (FID)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteFieldbus function.
OK : BOOL;	TRUE if successful.

This function adds a master node to the configuration.

PAFFBCREATESLAVE - ADD A SLAVE NODE

OK := paFfbCreateSlave (FID)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteFieldbus function.
OK : BOOL;	TRUE if successful.

This function adds a slave node to the configuration. The node is added under the last created or selected "master" node.

PAFFBCREATEVAR - ADD A VARIABLE NODE

OK := paFfbCreateVar (FID)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteFieldbus function.
OK : BOOL;	TRUE if successful.

This function adds a variable node to the configuration. The node is added under the last created or selected slave node.

PAFFBSETPROPERTY - SET THE VALUE OF A PROPERTY

OK := paFfbSetProperty (FID, PROP, VALUE)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteFieldbus function.
PROP : STRING	Internal name of the property - see notes.
VALUE : STRING	Value of the property in neutral format - see notes.
OK : BOOL;	TRUE if successful.

This function adds a variable node to the configuration. The node is added under the last created or selected slave node.

The name of the property must be entered in "neutral" form, independently from the language selected for the Workbench. Also, for a check box or a drop-list property, you need to enter the value in neutral form, and not as displayed in the fieldbus configuration tool. To know the relationship in between neutral and displayed names and values, go to the `Prompt` tab in the output window and hit the `?FB` Command followed by the name of the configuration. This command lists all properties for each level of the selected configuration tree, giving neutral names and readable names, and lists all possible neutral values for boolean and enumerated properties. Example of syntax:

```
>?FB K5BusCan
```

PAFFBSELECTROOT - SELECTS THE ROOT NODE

OK := paFfbSelectRoot (FID)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteFieldbus function.
OK : BOOL;	TRUE if successful.

This function selects the root node of the configuration tree. The root node is automatically selected when the file is open. Selecting a node is required for setting its properties or adding child nodes. It is not work to select a node immediately after its creation.

PAFFBSELECTMASTER - SELECTS A MASTER NODE

OK := paFfbSelectMaster (FID, MASTER)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteFieldbus function.
MASTER : DINT	1-based index in the list of master nodes.
OK : BOOL;	TRUE if successful.

This function selects a master in the configuration tree. Selecting a node is required for setting its properties or adding child nodes. It is not work to select a node immediately after its creation.

PAFFBSELECTSLAVE - SELECTS A SLAVE NODE

OK := paFfbSelectSlave (FID, MASTER, SLAVE)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteFieldbus function.
MASTER : DINT	1-based index of the parent master node.
SLAVE : DINT	1-based index in the list of slave nodes for the selected parent master.
OK : BOOL;	TRUE if successful.

This function selects a slave in the configuration tree. Selecting a node is required for setting its properties or adding child nodes. It is not work to select a node immediately after its creation.

PAFFBSELECTVAR - SELECTS A VARIABLE NODE

OK := paFfbSelectVar (FID, MASTER, SLAVE, VARIABLE)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteFieldbus function.
MASTER : DINT	1-based index of the parent master node.
SLAVE : DINT	1-based index of the parent slave node.
VARIABLE : DINT	1-based index in the list of variable nodes for the selected parent slave.
OK : BOOL;	TRUE if successful.

This function selects a variable in the configuration tree. Selecting a node is required for setting its properties or adding child nodes. It is not work to select a node immediately after its creation.

PAFLDNEXTBRANCH - ADD A BRANCH TO A DIVERGENCE

OK := paFLdNextBranch (FID)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
OK : BOOL;	TRUE if successful.

This function starts a new parallel branch in the current OR divergence. After calling this function, items can be added on the new branch.

PAFLDJUMP - ADD A JUMP SYMBOL

OK := paFLdJump (FID, LABEL)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
LABEL : STRING;	Target label name.
OK : BOOL;	TRUE if successful.

This function adds a jump instruction on the current rung.

PAFLDCONTACT - ADD A CONTACT

OK := paFLdContact (FID, TYP, NAME)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
TYP : STRING;	Type of contact (see notes).
NAME : STRING;	Name of the variable attached to the contact.
OK : BOOL;	TRUE if successful.

This function adds a contact on the current rung. The following values are available for the **TYP** parameter:

_LD_DIR	Normal contact.
_LD_INV	Negated contact.
_LD_P	Positive pulse contact.
_LD_N	Negative

PAFLDCOIL - ADD A COIL

OK := paFLdCoil (FID, TYP, NAME)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
TYP : STRING;	Type of coil (see notes).
NAME : STRING;	Name of the variable attached to the coil.
OK : BOOL;	TRUE if successful.

This function adds a coil on the current rung. The following values are available for the **TYP** parameter:

_LD_DIR	Normal coil.
_LD_INV	Negated coil.
_LD_P	Positive pulse coil.
_LD_N	Negative pulse coil.
_LD_SET	Set coil.
_LD_RESET	Reset coil.

PAFLDRETURN - ADD A <RETURN> JUMP SYMBOL

OK := paFLdReturn (FID)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
OK : BOOL;	TRUE if successful.

This function adds a <RETURN> jump instruction on the current rung.

PAFLDBLOCK - ADD A BLOCK

OK := paFLdBlock (FID, NAME, INSTANCE, INPUTS, OUTPUTS)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
NAME : STRING;	Type of block.
INSTANCE : STRING;	Name of the instance in case of a function block.
INPUTS : STRING;	List of input values (see notes).
OUTPUTS : STRING;	List of output values (see notes).
OK : BOOL;	TRUE if successful.

This function adds a block on the current rung. In case of a function block, you must specify the name of the instance. Blocks are connected to the rung by their first input and output. Name of additional inputs and outputs are passed in strings separated by comas. Below is the example of an "addition" block with 2 additional inputs called 'i1' and 'i2' and 1 additional output called 'q':

```
paFLdStartRung (f, '');
    paFLdBlock (f, '+', '', 'i1,i2', 'q');
    paFLdCoil (f, _LD_DIR, '');
paFLdEndRung (f);
```

PAFLDHORZSEGMENT - ADD AN HORIZONTAL SEGMENT

OK := paFLdHorzSegment (FID)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
OK : BOOL;	TRUE if successful.

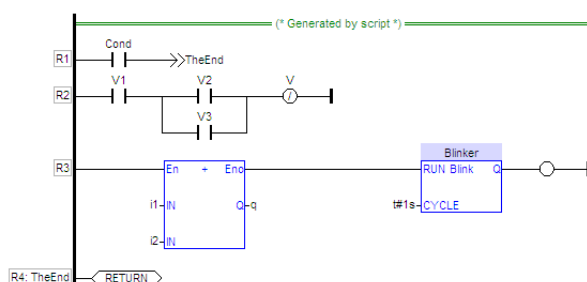
This function adds a horizontal line segment on the current rung. This can be used for aligning items vertically in the diagram.

Example:

Below is an example of script that generates a LD file in the target project:

```
// make LD code
f := paFileOpenWriteProgramSrc ('ProgLD');
if f <> 0 then
  // comment line
  paFLdCommentLine (f, 'Generated by script');
  // a simple rung finishing with a jump
  paFLdStartRung (f, '');
  paFLdContact (f, _LD_DIR, 'Cond');
  paFLdJump (f, 'TheEnd');
  paFLdEndRung (f);
  // a rung with contacts and coils
  paFLdStartRung (f, '');
  paFLdContact (f, _LD_DIR, 'V1');
  paFLdDivergence (f);
  paFLdContact (f, _LD_DIR, 'V2');
  paFLdNextBranch (f);
  paFLdContact (f, _LD_DIR, 'V3');
  paFLdConvergence (f);
  paFLdCoil (f, _LD_INV, 'V');
  paFLdEndRung (f);
  // a rung with blocks
  paFLdStartRung (f, '');
  paFLdBlock (f, '+', '', 'i1,i2', 'q');
  paFLdBlock (f, 'Blink', 'Blinker', 't#1s', '');
  paFLdCoil (f, _LD_DIR, '');
  paFLdEndRung (f);
  // a rung with a label and a RETURN instruction
  paFLdStartRung (f, 'TheEnd');
  paFLdReturn (f);
  paFLdEndRung (f);
  // close the file
  paFileClose (f);
end_if;
```

Here is the LD program generated by the script:



SFC - FILE WRITING SERVICES

The functions below enable you to write sequentially (from the top to the bottom) the source code of a program written in SFC language. The program is written sequentially, from the top to the bottom and branch per branch. Files are open using the `paFileOpenWriteSrc` function. When complete, they must be closed by calling the `paFileClose` function.

Use the following functions for inserting SFC items in the chart:

- ▶ `paSfcIStep` - Add an initial step.
- ▶ `paSfcStep` - Add a step.
- ▶ `paSfcTrans` - Add a transition.
- ▶ `paSfcJump` - Add a jump to a step.
- ▶ `paSfcDiv` - Start a divergence on the current branch.
- ▶ `paSfcBranch` - Start a new branch under the current divergence.
- ▶ `paSfcConv` - Finish the current divergence.
- ▶ `paSfcAddLv2String` - Add a text string to the level 2 program.
- ▶ `paSfcAddLv2Line` - Add a text line to the level 2 program.
- ▶ `paSfcAddLv2Eol` - Add end of line characters to the level 2 program.

PASFCISTEP - ADD AN INITIAL STEP

OK := `paSfcIStep` (FID, REF)

Parameters:

FID : DINT;	File identifier answered by <code>paFileOpenWriteSrc</code> .
REF : DINT;	Reference number of the step.
OK : BOOL;	TRUE if successful.

This function adds an initial step at the end of the current branch.

PASFCSTEP - ADD A STEP

OK := `paSfcStep` (FID, REF)

Parameters:

FID : DINT;	File identifier answered by <code>paFileOpenWriteSrc</code> .
REF : DINT;	Reference number of the step.
OK : BOOL;	TRUE if successful.

This function adds a step at the end of the current branch.

PASFCTRANS - ADD A TRANSITION

OK := paSfcTrans (FID, REF)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
REF : DINT;	Reference number of the transition.
OK : BOOL;	TRUE if successful.

This function adds a transition at the end of the current branch.

PASFCJUMP - ADD A JUMP TO A STEP

OK := paSfcJump (FID, REF)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
REF : DINT;	Reference number of the target step.
OK : BOOL;	TRUE if successful.

This function adds a jump to a step at the end of the current branch.

PASFCDIV - START A DIVERGENCE

OK := paSfcDiv (FID)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
OK : BOOL;	TRUE if successful.

This function starts a divergence the end of the current branch. It automatically initiates the first branch (left side) so you can start inserting items on it. The divergence will have to be ended by a convergence, even if its branches finish by a jump symbol (and thus do not actually converge). Divergences can be nested.

PASFCBRANCH - ADD A BRANCH TO THE LAST OPEN DIVERGENCE

OK := paSfcBranch (FID)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
OK : BOOL;	TRUE if successful.

This function adds a new branch on the right of the divergence created by the last call to paSfcDiv(). It is not needed to call this function for the first branch (on the left) as it is automatically initiated when the divergence is created.

PASFCCONV - FINISH A DIVERGENCE

OK := paSfcConv (FID)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
OK : BOOL;	TRUE if successful.

This function closes the divergence created by the last call to paSfcDiv(). The divergence will have to be ended by a convergence, even if its branches finish by a jump symbol (and thus do not actually converge). Divergences can be nested.

3.4.4. Binding**PAFILEOPENWRITEBINDING - OPEN THE BINDING CONFIGURATION FOR WRITING**

FID := paFileOpenWriteBinding ()

Parameters:

FID : DINT;	File identifier or 0 if fail.
-------------	-------------------------------

This function opens the binding configuration file for writing. This functions removes the contents of the configuration if it already exist. When written, the file must be closed by calling the paFileClose function.

PAFBINDPUBLICVAR - ADD A PUBLISHED VARIABLE TO THE BINDING CONFIGURATION

FID := paFbindPubilcVar (FID, ID, NAME, HYSTP, HYSTN)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteBinding function.
ID : DINT;	ID of the binding link.
NAME : STRING;	Symbol of the variable.
HYSTP : STRING;	Positive hysteresis in IEC syntax (or empty string).
HYSTN : STRING;	Negative hysteresis in IEC syntax (or empty string).
OK : BOOL;	TRUE if successful.

This function adds a public variable in the binding configuration.

PAFBINDEXTERNPORT - ADD AN EXTERNAL PORT TO THE BINDING CONFIGURATION

FID := paFbindExternPort (FID, ADDR, PORT, NAME)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteBinding function.
ADDR : STRING;	IP address.
PORT : DINT;	IP port number.
NAME : STRING	Optional description text.
OK : BOOL;	TRUE if successful.

This function adds a public variable in the binding configuration. Immediately after calling this function, you can use other functions to declare external bound variables.

PAFBINDCNXSTATUS - BINDS A VARIABLE TO THE CONNECTION STATUS OF THE LAST DECLARED PORT

FID := paFbindCnxStatus (FID, NAME)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteBinding function.
NAME : STRING;	Symbol of the bound variable.
OK : BOOL;	TRUE if successful.

This function binds a variable for getting the connection status. The variable is bound to external port declared by the last call to the paFbindExternPort function.

PAFBINDEXTERNVAR - BINDS A VARIABLE TO THE LAST DECLARED PORT

FID := paFbindExternVar (FID, ID, NAME)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteBinding function.
ID : DINT;	ID of the binding link.
NAME : STRING;	Symbol of the bound variable.
OK : BOOL;	TRUE if successful.

This function binds a variable for data exchange. The variable is bound to external port declared by the last call to the paFbindExternPort function.

PAFBINDVARSTATUS - BINDS A STATUS VARIABLE TO THE LAST DECLARED PORT

FID := paFbindVarStatus (FID, ID, NAME)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteBinding function.
ID : DINT;	ID of the binding link.
NAME : STRING;	Symbol of the bound variable.
OK : BOOL;	TRUE if successful.

This function binds a variable for getting the status of a binding link. The variable is bound to external port declared by the last call to the paFbindExternPort function.

PAFBINDVARDATESTAMP - BINDS A DATE STAMP VARIABLE TO THE LAST DECLARED PORT

FID := paFbindVarDateStamp (FID, ID, NAME)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteBinding function.
ID : DINT;	ID of the binding link.
NAME : STRING;	Symbol of the bound variable.
OK : BOOL;	TRUE if successful.

This function binds a variable for getting the date stamp of a binding link. The variable is bound to external port declared by the last call to the paFbindExternPort function.

PAFBINDVARTIMESTAMP - BINDS A TIME STAMP VARIABLE TO THE LAST DECLARED PORT

FID := paFbindVarTimeStamp (FID, ID, NAME)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenWriteBinding function.
ID : DINT;	ID of the binding link.
NAME : STRING;	Symbol of the bound variable.
OK : BOOL;	TRUE if successful.

This function binds a variable for getting the time stamp of a binding link. The variable is bound to external port declared by the last call to the paFbindExternPort function.

PASFCADDLV2STRING - ADD TEXT STRING TO LEVEL2 PROGRAMMING

OK := paSfcAddLv2String (FID, KIND, TEXT)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
KIND : STRING;	Kind of level 2 information - see notes.
TEXT : STRING;	Text to be appened.
OK : BOOL;	TRUE if successful.

This function appends a string to the level 2 programming of the last created step or transition. It must be called immediately after the step or transition is created. The KIND argument specifies which piece of level 2 information must be written. **The following values are predefined:**

_LV2_NOTE	Description text.
_LV2_COND	Condition (for a transition).
_LV2_ACTION	Default actions of a step.
_LV2_P1	P1 actions of a step.
_LV2_N	N actions of a step.
_LV2_P0	P0 actions of a step.

Generating level 2 programs in LD or FBD is not supported.

PASFCADDLV2LINE - ADD TEXT LINE TO LEVEL2 PROGRAMMING

OK := paSfcAddLv2Line (FID, KIND, TEXT)

Parameters:

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
KIND : STRING;	Kind of level 2 information - see notes.
TEXT : STRING;	Text to be append.
OK : BOOL;	TRUE if successful.

This function appends a string plus end of lines characters ('\$R\$N') to the level 2 programming of the last created step or transition. It must be called immediately after the step or transition is created. The KIND argument specifies which piece of level 2 information must be written. See paSfcAddLv2String for possible values.

PASFCADDLV2EOL - ADD END OF LINE CHARACTERS TO LEVEL2 PROGRAMMING

OK := paSfcAddLv2Eol (FID, KIND)

Parameters :

FID : DINT;	File identifier answered by paFileOpenWriteSrc.
KIND : STRING;	Kind of level 2 information - see notes.
OK : BOOL;	TRUE if successful.

This function appends end of lines characters ('\$R\$N') to the level 2 programming of the last created step or transition. It must be called immediately after the step or transition is created. The KIND argument specifies which piece of level 2 information must be written. See paSfcAddLv2String for possible values.

Example

Below is an example of script that generates a SFC file in the target project:

```
// make LD code
f := paFileOpenWriteProgramSrc ('ProgSFC');
if f <> 0 then
  // initial step
  paSfcIStep (f, 1);
  paSfcAddLv2String (f, _LV2_NOTE, 'Thats S1');
  paSfcAddLv2Line (f, _LV2_ACTION, 'VarXX (N);');
  // an 'OR' divergence
  paSfcDiv (f);
  paSfcTrans (f, 1);
  paSfcAddLv2String (f, _LV2_NOTE, 'Thats T1');
  paSfcAddLv2Line (f, _LV2_COND, 'condition');
  // second branch
  paSfcBranch (f);
  paSfcTrans (f, 101);
  paSfcStep (f, 101);
  paSfcTrans (f, 102);
  // converge
  paSfcCnv (f);
  // jump to the initial step
  paSfcJump (f, 1);
  // close the file
  paFileClose (f);
end_if;
```

3.4.5. Watch window documents - file writing services

PAFILEOPENWRITESPYLIST - OPEN A SPY LIST DOCUMENT FOR WRITING

FID := paFileOpenWriteSpyList (NAME)

Parameters:

NAME : STRING;	Name of the spy list (no suffix!).
FID : DINT;	File identifier or 0 if fail.

This function opens a spy list document file for writing. This functions removes the contents of the spy list if it already exist. When written, the file must be closed by calling the paFileClose function.

PAFSPYADDVARIABLE - ADD A VARIABLE TO A SPY LIST

OK := paFSpyAddVariable (FID, NAME)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenSpyList function.
NAME : STRING;	Symbol of the variable.
OK : BOOL;	TRUE if successful.

This function adds a variable to an open spy list document.

PAFILEOPENWRITERECIPE - OPEN A RECIPE DOCUMENT FOR WRITING

FID := paFileOpenRecipe (NAME)

Parameters:

NAME : STRING;	Name of the recipe (no suffix!).
FID : DINT;	File identifier or 0 if fail.

This function opens a recipe document file for writing. This functions removes the contents of the recipe if it already exist. When written, the file must be closed by calling the paFileClose function.

PAFRCPADDCOLUMN - ADD A COLUMN TO A RECIPE

OK := paFrcpAddColumn (FID, COL, NAME)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenRecipe function.
COL : DINT;	Index of the column (0 based).
NAME : STRING;	Name of the column.
OK : BOOL;	TRUE if successful.

This function adds a column to a recipe. The index of the first column is 0. Column must be declared from the left to the right. Columns must be defined before adding values to the recipe.

PAFRCPADDVARIABLE - ADD A VARIABLE TO A RECIPE

OK := paFrcpAddVariable (FID, NAME)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenRecipe function.
NAME : STRING;	Symbol of the variable.
OK : BOOL;	TRUE if successful.

This function adds a variable (a line) to an open recipe document. Variables must be defined before adding values to the recipe.

PAFRCPADDVALUE - ADD A VALUE TO A RECIPE

OK := paFrcpAddValue (FID, NAME, COL, VALUE)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenRecipe function.
NAME : STRING;	Symbol of the variable.
COL : DINT;	Index of the column (0 based).
VALUE : STRING;	Value in IEC syntax.
OK : BOOL;	TRUE if successful.

This function adds a value (one cell) to a recipe. The corresponding variable (line) and column must be defined before calling this function.

3.4.6. Resource documents - file writing services

PAFILEOPENWRITESTRINGTABLE - OPEN A "STRING TABLE" RESOURCE DOCUMENT FOR WRITING

FID := paFileOpenStringTable (NAME)

Parameters:

NAME : STRING;	Name of the string table (no suffix!).
FID : DINT;	File identifier or 0 if fail.

This function opens a "string table" resource document file for writing. This functions removes the contents of the string table if it already exist. When written, the file must be closed by calling the paFileClose function.

PAFSTBADDCOLUMN - ADD A COLUMN TO A STRING TABLE RESOURCE

OK := paFstbAddColumn (FID, NAME)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenStringTable function.
COL : DINT;	Index of the column (0 based).
NAME : STRING;	Name of the column.
OK : BOOL;	TRUE if successful.

This function adds a column to a "string table" resource. The index of the first column is 0. Column must be declared from the left to the right. Columns must be defined before adding strings to the table.

PAFILEOPENWRITESIGNAL - OPEN A "SIGNAL" RESOURCE DOCUMENT FOR WRITING

FID := paFileOpenSignal (NAME)

Parameters:

NAME : STRING;	Name of the signal (no suffix!).
FID : DINT;	File identifier or 0 if fail.

This function opens an "analog signal" resource document file for writing. This functions removes the contents of the resource if it already exist. When written, the file must be closed by calling the paFileClose function.

PAFSTBADDSTRING - ADD A COLUMN TO A STRING TABLE RESOURCE

OK := paFstbAddString (FID, NAME, COL, VALUE)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenStringTable function.
NAME : STRING;	Identifier of the string in the table.
COL : DINT;	Index of the column (0 based).
VALUE : STRING;	Contents of the string.
OK : BOOL;	TRUE if successful.

This function adds a string to a "string table" resource. The corresponding column must be defined before calling this function.

PAFSIGNALADDCOLUMN

OK := paFSignalAddColumn (FID, NAME)

Parameters:

FID : DINT;	File identifier returned by the paFileOpenSignal function.
COL : DINT;	Index of the column (0 based).
NAME : STRING;	Name of the column.
OK : BOOL;	TRUE if successful.

This function adds a column to an "analog signal" resource. The index of the first column is 0. Column must be declared from the left to the right. Columns must be defined before adding points are added to the signal.

PAFSIGNALADDPPOINT - ADD A POINT TO A SIGNAL RESOURCE

OK := paFSignalAddPoint (FID, TM, COL, VALUE)

This function adds a point to an "analog signal" resource. The corresponding column must be defined before calling this function.

3.5. Templates

3.5.1. Variable keywords for copying templates

Your scripts can contain template documents to be copied in the target project. When copying documents, some special keywords may be variables defined from the script and replaced by the value given by the script. Use the following syntax in your templates for specifying variable keywords:

`$(keyname)`

And use the following functions in your script to set values for keywords.

PASETKEY - DEFINE A VARIABLE KEYWORD FOR COPYING TEMPLATES

OK := paSetKey (NAME, VALUE)

Parameters:

NAME : STRING;	Name of the variable keyword.
VALUE : STRING;	Value of the variable keyword.
OK	TRUE if successful.

This function defines the value of a defined keyword to be replaced when copying a template.

Example:

Template ST program:

```

if ALARM_$(ALM) then
    bMainAlarm := TRUE;
end_if;
Script instructions:
paSetKey ('ALM', 'Engine2')
paTplCopySource ('TargetProgram', 'TemplateName');
Generated ST program:
if ALARM_Engine2 then
    bMainAlarm := TRUE;
end_if;

```

PAREMOVEKEYS - REMOVE ALL VALUES DEFINED FOR TEMPLATE VARIABLE KEYWORDS

OK := paRemoveKeys ()

Parameters:

OK	TRUE if successful.
----	---------------------

This function removes all values defined for variable keywords used for copying templates.

3.5.2. Script parameters

The project automation system enables your script to prompt the user for entering some parameters. Such parameters are then used (as variables) by your script.

Parameters must be variables of your script project declared as global variables. Then you need to create a list of variables in your script project that groups all "parameter" variables. Finally in your script, you must call the `paEditParameterList` function that opens a dialog box where the user must give values for each parameter. Then "parameter" variables can be used in your script.

A parameter must be a variable declared with a single data type (no array, no structure). All basic data types are supported. When the parameter input dialog box is open, current values of variables are shown as default value.

For a string parameter, it is possible to define a drop-list choice. For that you must set possible choices in the string value before editing parameter, separated by '|' characters. e.g.: 'Choice1|Choice2'.

PASETPARAMETERDESC - SET THE DESCRIPTION TEXT TO BE SHOWN WITH A PARAMETER

OK := `paSetParameterDesc` (NAME, DESC)

Parameters:

NAME : STRING;	Name of the parameter, such as declared in your script project.
DESC : STRING;	Description text to be show to the user when entering parameters.
OK : BOOL	TRUE if successful.

This function defines the description text to be shown together with the specified parameter when entered.

PAEDITPARAMETERLIST - PROMPT THE USER TO ENTER PARAMETER VALUES

OK := `paEditParameterList` (LISTID, TITLE)

Parameters:

LISTID : DINT;	ID of the list of parameters (use <code>VLID</code> function).
TITLE : STRING;	Title of the window where the user enters the parameter.

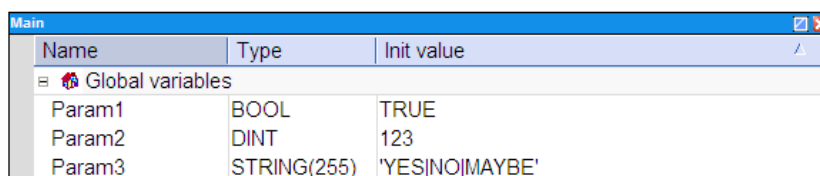
OK : BOOL;	TRUE if the use pressed the ok button.
------------	--

This function opens a dialog box where the user enters values for script parameters.

Example

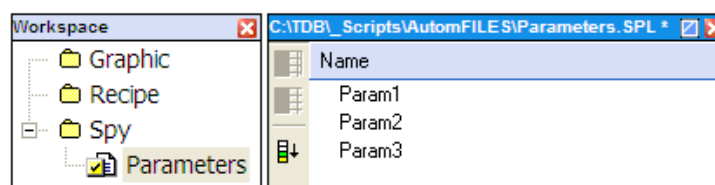
This examples shows the different steps for adding parameters to a script:

- Declare parameters as global variables of your script project:



Name	Type	Init value
Global variables		
Param1	BOOL	TRUE
Param2	DINT	123
Param3	STRING(255)	'YES NO MAYBE'

- Create a list of variables with parameters:

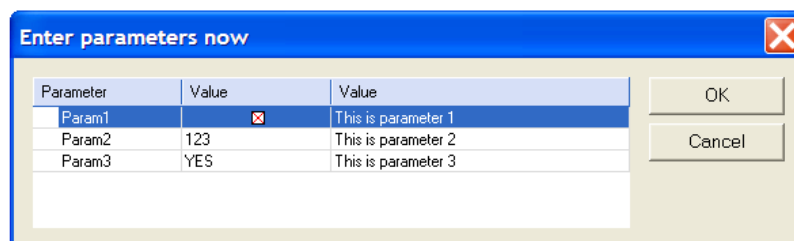


Name
Param1
Param2
Param3

- ST Script:

```
// set decription text for parameters
paSetParameterDesc ('Param1', 'This is parameter 1');
paSetParameterDesc ('Param2', 'This is parameter 2');
paSetParameterDesc ('Param3', 'This is parameter 3');
// prompt the user for parameters
paEditParameterList (VLID ('Parameters'), 'Enter parameters now');
```

When the script is run, the following dialog box is open:



Parameter	Value	Value
Param1	<input checked="" type="checkbox"/>	This is parameter 1
Param2	123	This is parameter 2
Param3	YES	This is parameter 3

OK Cancel

The Param1 parameter is a check box, because it is declared as BOOL. The Param3 is shown as a drop list because the value of Param3 (its initial value in that case) is a list of words separated by '|' characters.

3.5.3. Miscellaneous

PAGETLANGUAGE - GET CURRENT SELECTED LANGUAGE

LGE := paGetLanguage ()

Parameters:

LGE : DINT;	Language identifier (see notes).
-------------	----------------------------------

This function returns the language currently selected for the Workbench. **Possible returned values are:**

_WB_USA	English
_WB_GER	German
_WB_FRA	French
_WB_ITA	Italian
_WB_SPA	Spanish
_MW_KOR	Korean

PATRACE... - OUTPUT A REPORT MESSAGE

paTRACE0 (TEXT)

paTRACE1 (TEXT, ARG1)

paTRACE2 (TEXT, ARG1, ARG2)

paTRACE3 (TEXT, ARG1, ARG2, ARG3)

paTRACE4 (TEXT, ARG1, ARG2, ARG3, ARG3)

Parameters:

TEXT : STRING;	Text to be output.
ARG1..4 : DINT;	Arguments.

The message specified in the TEXT argument is output to the report window when the automation script is run. The message text may include up to 4 integer arguments, specified in the TEXT string with special character sequences:

```
%ld    signed value in decimal.
%lu    unsigned value in decimal.
%lx    value in hexadecimal.
```