User manual PG5.
Contents

1 PCD-Quick start
2 Project Manager
3 Device Configurator
4 PCD – Data Type
5 Symbol Editor
6 Program with Fupla
7 Program structures
8 Graftec programming
9 Programming in IL (Instruction List)
10 Additional tools
11 Saia PCD Networks
12 Profi-S-Bus
13 Ether-S-Bus
14 Profi-S-IO
Preface

This document is intended as an introduction to Saia PCD® programmable controllers, rather than as a detailed commissioning manual. It therefore concentrates on the essential points for users who wish to acquire practical expertise quickly. For more comprehensive information, please refer to the help supplied by the programming tool itself, or to the detailed manuals that will be found on https://www.sbc-support.com

To ensure ideal conditions for your training, we advise you to obtain the following programs, documentation and material:

- CD Saia PG5 version 2.3
- 1 x PCD2.M5540 ¹ controller
- 1 x PCD2.E110 module with 8 digital inputs
- 1 x PCD2.A400 module with 8 digital outputs
- 1 x USB cable

Please also note that all the English names of menus, instructions, options and buttons present in the PG5 program are reproduced in *italics* in this manual.

We wish you every success with your training and with future projects involving Saia PCD® products.

Your partner Saia-Burgess Controls AG.

¹ an other PCD may also be suitable
1 PCD – QUICK-START ...................................................................................................................... 2

1.1 Introduction ..................................................................................................................................... 2

1.2 Preparing the hardware .................................................................................................................. 3
  1.2.1 Example: Stairway lighting ...................................................................................................... 3
  1.2.2 Connection diagram of PCD2.M5540 ...................................................................................... 3
  1.2.3 PCD2.M5540 assembly ............................................................................................................ 4
  1.2.4 Wiring ....................................................................................................................................... 4

1.3 Editing the program ......................................................................................................................... 5
  1.3.1 Software Installation ................................................................................................................. 5
  1.3.2 Starting the PG5 ....................................................................................................................... 5
  1.3.3 Opening a new project .............................................................................................................. 6
  1.3.4 Opening an existing project ...................................................................................................... 6
  1.3.5 Configuration ............................................................................................................................ 7
  1.3.6 Adding a new program file ...................................................................................................... 9
  1.3.7 Opening an existing file .......................................................................................................... 10
  1.3.8 Editing a program ................................................................................................................... 10

1.4 Running and testing the program .................................................................................................. 14
  1.4.1 Building the program ............................................................................................................... 14
  1.4.2 Downloading the program into the PCD .............................................................................. 14

1.5 Finding and correcting errors (Debugging) .................................................................................... 15

1.6 Correcting a program ...................................................................................................................... 16
1 PCD – Quick-start

1.1 Introduction

As your first point of contact with Saia PCD® equipment, we propose a direct approach: tackling the production of a small real-life application. Even without any experience of Saia PCD products, this is easy to do. Everything is set out in detail in this quick-start chapter.

This example shows how to commission a PCD2.M5540. Programming and testing using the Saia PG5® programming tools.

Subsequent chapters in this document repeat in more detail the contents of this quick-start chapter, and provide much more information such as descriptions of available symbols, program structures and instruction list programming.
1.2 Preparing the hardware

1.2.1 Example: Stairway lighting

The commissioning of a Saia PCD® is illustrated using stairway lighting as an example. The building has a ground floor and three upper storeys. Each level has a push-button for switching the lights on. By briefly pressing any of these buttons, all 4 lights in the stairway will be switched on for a period of 5 minutes.

The push-buttons are connected to the 4 inputs of the PCD: I0, I1, I2 and I3.

The 4 lights are switched on/off via a relay. The relay is controlled via a single output (O32) on the PCD.

1.2.2 Connection diagram of PCD2.M5540
1.2.3 PCD2.M5540 assembly

1. Insert the supplied 3.0 V lithium battery.
2. Plug a PCD2.E110 module into socket 1 (addresses 0 to 15).
3. Push the module towards the middle of the device until the end stop and engage latch. This provides 8 digital inputs for 24 VDC with addresses I0 to I7. Only inputs I0 to I4 will be used.
4. Plug a PCD2.A400 module into socket 3 (addresses 32 to 47) as previously described. This provides 8 digital outputs (O32 to O39) for 24 VDC / 0.5A. Only output O32 will be used.

1.2.4 Wiring

1. Connect the 24 VDC supply to screw terminals 20 (+) and 23 (–). The following supply voltages are allowed: 24 VDC ±20% smoothed or 19 VAC ±15% full-wave rectified

2. The four inputs used are connected according to the hardware description of the PCD2.E110 module. Connect the 4 push-button switches to terminals 0 to 3. Terminals 8 and 9 are connected to the power supply negative.

3. Connect terminal 0 to the relay coil, terminal 8 to the 24 VDC supply positive, and terminal 9 to the supply negative.

4. Connect the PC’s USB interface to the PCD

Note: For more detailed information about hardware assembly and wiring, please refer to your PCD hardware manual.
1.3 Editing the program

1.3.1 Software Installation

Install the Saia PG5® programming tool for Saia PCD® on the PC (if it has not already been installed), following the instructions supplied on the CD (CD:\Manuals\English\PG5_InstallationGuide_en.pdf)

1.3.2 Starting the PG5

Start the PG5's Project Manager:

Start → All Programs → PG5 Suite V2.3.160

The Saia PG5 Project Manager window is displayed. The Project Tree window shows the structure of the new project. (If this window is not shown, use the View, Project Tree menu command.)
Folders in the Project Tree window contain project information, which is arranged according to certain criteria:

- The name of the main folder shows the project name and the number of devices used in the project.
- Files which are shared by several of the devices can be stored in the Common Files folder.
- Next are the device folders (each device represents a PCD).

Each device folder contains the following sub-folders:

- Online Settings, Device Configurator and Build Options.
- Program Files, contains the program module files.
- Listing Files, contains files generated during the program build (Build). These are not so interesting to the inexperienced user.

1.3.3 Opening a new project

Before starting to write a new program, a new or existing project must be opened that contains the necessary definitions, a few configuration parameters and the files needed for the user program.

If the project does not yet exist, select Project, New..., define the name of the new project in the Project Name field, check the Create Device option and confirm with the OK button.

1.3.4 Opening an existing project

A project which already exists can be opened using the Project, Open... menu command. This searches for all project files (.saia5pj) in the project directory, and displays them in a list. Double-click on the project in the list, or select the project from the list and press the Open button. Alternatively, you can press the Browse... button and find the project or device file directly.
1.3.5 Configuration

Before you can work with a device in the project, configuration parameters must be defined, so that the programming tools and the generated user program will work with the PCD.

Under Online Settings, parameters can be set for communication between the PC and the PCD. Several possibilities are available. For this exercise the default protocol (S-Bus USB).

Channel S-Bus USB

![Channel S-Bus USB](image)
The Device Configurator defines the device type, memory size, S-Bus station number, communications interfaces, etc., but we won't describe all the options just yet. However, it is important to select the correct device type and memory size. The PCD2.M5540 is always supplied with a standard 1024 Kbyte RAM.

To define a new device, select the correct PCD type from the Change Device Type context menu.

The properties window is shown with the Properties context menu command. Here the memory size of can be defined

The configuration must always be downloaded into the PCD, using the menu command Online, Download Configuration…
1.3.6 Adding a new program file

PCD user programs are stored in one or more files. There are several ways of adding a program file:

In the Project Tree window, select Program Files, click the right-hand mouse button to display the context menu and select New… (new file).

Alternatively, you can click on the New File button on the toolbar, or use the File, New… menu command.

In the New File window the name and type of the module are defined: two very important items of information.

Several editors are available for writing PCD user programs. The user can choose which editor is best suited to the user program. For this example it is Fupla File (*.fup). Fupla is a general-purpose graphical programming language.
1.3.7 Opening an existing file

If the folder already contains a program file, the file can be opened as follows:
In the Project Tree window, open the Program Files folder and double-click on the relevant file. Alternatively, right-click on the file name to open the context menu and select Open.

1.3.8 Editing a program
**Editing symbols**

Symbols reference the data needed by the PCD’s user program, e.g. the stairway lighting switches. We edit symbols in the connectors on the Fupla page. ‘Read’ symbols are on the left, ‘write’ symbols are on the right.

This stairway lighting example has 4 light switches as inputs (I 0, I 1, I 2 and I 3) and one output (O 32) to drive the stairway lights. The required period of 5 minutes, during which stairway lighting will be on, must be entered in the input connector as a multiple of tenths of a second. The value of this constant is therefore 3000 (5 min. x 60 sec. x 10 = 3000).

To add a connector and its symbol to a Fupla page, press the toolbar button Add Connector and position the mouse on the Fupla page. A ‘read’ input connector is added by clicking the left-hand mouse button. A ‘write’ output connector is added by holding down the Shift key and pressing the left-hand mouse button. The connector you have just added is ready to receive a symbol and a cursor is displayed inside the connector. If you do not wish to edit the symbol inside the connector straight away, press the ESC key and place the next connector.

To edit or modify a connector symbol already present on the Fupla page, select the connector by double clicking quickly. A cursor will be displayed inside the connector. It is now possible to enter the address I 0 to I 3, or output O 32, or the constant. Make sure you always leave a space between the letter I and the input address. The same applies for the output.

To edit the input symbols, 4 consecutive cells in the left-hand column of the program screen are marked with the mouse and addresses I 0 to I 3 are entered. The time constant 3000 (left) and output O 32 (right) are entered in the same way. Please note that the address type (I or O) and address value (0 to 3 and 32) must be separated by a space character.

The symbols will immediately appear in the Symbols window of the Symbol Editor. If the symbol editor is not visible it can be displayed using the View, Symbols menu command or by pressing the Show/Hide Symbol Editor toolbar button:

Note: By default, each new page may already provide margins with connectors on the left and right. If you prefer new pages not to appear with these connectors, so that you can place them wherever you want, just turn off the relevant option with menu command: View, Options..., Workspace, New pages with side connectors.

To remove any empty connectors present on the left or right of the page, select menu: Page, Remove Unused Connectors.

To place connectors once again on a blank page, select menu: Page, Add Side Connectors.
Editing program functions

Program functions are inserted in the area between the ‘read’ and ‘write’ connectors. This is done by positioning the graphical symbols of the function boxes (FBoxes) that are used to create user programs.

Function boxes are selected from the FBox Selector window.

The first function required in this example serves to switch on the lighting in response to a short pulse from a stairway switch. This is an OR function, which is found in the Binary family.

The second function Off delay defines the 5 minute period during which the lights are on. It is found in the Timer.

Further information about the FBox can be found by right-clicking on the function in the FBox Selector window, and choosing the FBox Info context menu command.

When a function box has been selected from the Selector, the left-hand mouse button is used to place it in the edit window between symbol columns.

With certain function boxes, such as OR logic, the number of inputs can be selected. This is done by dragging the mouse vertically and clicking the left-hand mouse button when the number of inputs is correct.
Connecting function

Use this method when connection points are aligned horizontally

1. Click on the *Select Mode* button on the toolbar.
2. Point onto the FBox, then press the left mouse button. Without releasing the button, drag the FBox to the adjacent FBox.
3. Release the mouse button once the link is indicated with a yellow point.
4. Drag the FBox again to position it where you want it.

Use this method for the others connections.

1. Click on the *Draw Lines* button on the toolbar.
2. Position the mouse pointer at the depart, a yellow point is displayed when the link is detected, click on the left mouse button.
3. Position mouse pointer on destination point, a yellow point is displayed when the link is detected, click the left-hand mouse button.

Deleting a line, function box, symbol or connector

Press the *Delete Object* toolbar button and click on the line, FBox, Symbol or Connector to be deleted.
1.4 Running and testing the program

1.4.1 Building the program

Before the program can be executed by the PCD, it must be built (compiled, assembled and linked) using the Project Manager's Device, Build Changed Files menu command, or the Rebuild All Files toolbar button.

The results of the build are shown in the Messages window (Compiling, Assembling, Linking etc.). If the program has been correctly edited, the build function completes with the message: Build successful. Total errors 0 Total warnings: 0

Errors are indicated by a red error message. Most errors can be located in the user program by double-clicking on the error message.

1.4.2 Downloading the program into the PCD

The user program is now ready. All that remains is to download it from the PC into the PCD. This is done using Project Manager's Download Program toolbar button or the Online, Download Program menu command.

If any communications problems arise, check the Online Settings and the cable connection between the PC and the PCD.
1.5 Finding and correcting errors (Debugging)

The first version of a program is rarely perfect. A stringent test is always needed. Program testing is done using the same editor that was used to write the program.

1. Press the Go On /Offline button

2. Start program with the Run button

Observe the Run LED on the PCD at the same time.

When the Run button is pressed, the Run LED on the PCD should turn on because the PCD is now executing the user program.

When the Stop button is pressed, the Run LED on the PCD should turn off because the PCD has stopped executing the program.

When the editor is Online and the PCD is in Run mode, the state of each individual symbol can be displayed:

- The logical state of binary data is shown with a heavy or fine line (heavy = 1 and fine = 0)
- Other data values can be displayed by clicking the left-hand mouse button on the connection to show a Probe window: use the mouse to select the Add Probe button and the link.
1.6 Correcting a program

To modify a program, proceed as follows:

1. Go offline (using the **Go On / Offline** button).
2. Modify the program.
3. Build the new program (with the **Build** button).
4. Download program to the PCD with the **Download Program** button.
2 PROJECT MANAGEMENT ................................................................. 2

2.1 Project Organisation ................................................................................. 2
2.1.1 Opening a Project ........................................................................... 3
2.1.2 Creating a new Project ................................................................. 4
2.1.3 How projects are stored on the PC ................................................... 4
2.1.4 Backup and restore of a Project or Device ......................................... 5

2.2 The Project Tree .................................................................................... 6
2.2.1 Project folder .................................................................................. 6
2.2.2 Common Files folder ....................................................................... 7
2.2.3 Libraries folder .............................................................................. 7
2.2.4 Device folder .................................................................................. 9
2.2.5 Online Settings ............................................................................. 9
2.2.6 Connecting the PC to the PCD .......................................................... 10
2.2.7 Device Configurator ...................................................................... 11
2.2.8 Build Options ............................................................................... 11
2.2.9 Program Files folder ....................................................................... 13
2.2.10 File Types ................................................................................... 15

2.3 Building the Program ............................................................................ 17
2.3.1 Build Changed Files, Rebuild All Files, Rebuild All Devices ............... 18

2.4 Messages Window .................................................................................. 18

2.5 Downloading the Program into the PCD ............................................... 19
2.5.1 Download Program ......................................................................... 19

2.6 Commands for All Devices .................................................................... 20

2.7 Self Downloading Files .......................................................................... 21
2.7.1 Creating a self-downloading file ...................................................... 21
2.7.2 Downloading a self-downloading file .............................................. 22

2.8 Flash Backup Memory .......................................................................... 23
2.8.1 Saving the executable program ....................................................... 23
2.8.2 Saving the program's source code .................................................. 25
2.8.3 Backing up data to a file ............................................................... 28

2.9 The View Windows .............................................................................. 29
2.9.1 Block Call Structure ...................................................................... 29
2.9.2 Global Symbols and Data List Views .............................................. 30
2.9.3 Cross-Reference List ..................................................................... 30

2.10 The Online Configurator ..................................................................... 31
2.10.1 Device Configurator ..................................................................... 31
2.10.2 PCD History ................................................................................ 32
2.10.3 Setting the PCD's Clock ............................................................... 32
2.10.4 Saving program and data from RAM ............................................ 33
2.10.5 Create Diagnostic File .................................................................. 34
2.10.6 Firmware Downloader .................................................................. 34
2 Project Management

2.1 Project Organisation

Modern automation processes usually contain several devices connected in a network, where each device carries out a particular function. For example, a building control system might have dedicated devices for handling lighting, heating, ventilation, automatic doors in the underground garage, etc.
The Saia PG5 Project Manager provides a global view of all the devices and files in a single project. All operations are started from here. For example, adding new program files to the project, opening files to write the programs, configuring the hardware, building and downloading the programs into the devices, backup and restore, reporting errors and warnings when building the program, etc.

The Project Tree window shows a hierarchical representation of each device and its configuration and program files. To display this window use the View, Project Tree menu command.

The active device is marked with a green triangle. Many of the menus and Tool Bar buttons work on the active device. To change the active device, just select another device in the Project Tree, or use the Device, Set Active menu command.

The Messages window shows general information messages, and error and warning messages generated when building a program. Display this window with the View, Messages menu command.

2.1.1 Opening a Project

The PG5 is installed with all the examples in this manual. The Project, Open… command displays a list of all projects by finding all the project files (.saia5pj) in the Projects Directory. Double-click on the project to open it, or select the project and press the Open button. Alternatively you can press the Browse… button and find a specific project or device file (.saia5pc) in another directory.
2.1.2 Creating a new Project

To start a new project, use the Project, New… menu command, enter the name of the project in the Project Name field (default = ProjectX) and press OK.

**New Project**

- **Project Name:** The name of the new project.
- **Projects Directory:** Name of folder which will contain the project.
- **Description:** Free text, description of the project.
- **Create Device:** Automatically creates a single device with the same name as the project.

2.1.3 How projects are stored on the PC

By default, all projects are stored in the following project directories, depending on the Windows version:

**Windows 10:** `C:\Users\User_Name\SBC\PG5 V2.3.1xx\Projects`

If necessary, the project (and library) directories can be changed using the Tools, Options… menu command:

- **Directories**
  - Applications Directory <AppsDir>
  - C:\Program Files (x86)\SBC\PG5 V2.3.160

The project is saved in a directory with the same name as the project. Each device is stored in a subdirectory of the project folder.
To store a project, it is necessary to maintain the entire directory and file structure forming the project.

Tools can be used to compress all data relating to a project or device into a *.zip file and to restore it again.

### 2.1.4 Backup and restore of a Project or Device

To backup a project it is necessary to keep the same folder structure and all the source files which comprise the project. The **Project, Backup...** menu command compresses either the entire project or just a single device into a standard *.zip* file, which can be restored using the **Project, Restore...** command.

**Backup what:**
Selects whether the entire project, or just a single device in the project will be backed up. By default it is the entire project.

**To Compressed File**
The path of the compressed *.zip* file which will be created.

**Backup Directory** : destination directory for backup file.

**Backup File Name** : name of backup .zip file

**Comment**: Free text describing the backup.

**Backup What**
Defines which will be saved. It's not necessary to save all the files, only the source ad configuration files are important.
The Project, Restore command can be used to restore the project or device backed up in a *.ZIP file to the PG5 project directory.

### 2.2 The Project Tree

![Project Tree Image](image)

Displayed by the View, Project Tree command, the Project Tree provides a structured view or the project information.

#### 2.2.1 Project folder

The top-level folder represents the project, with its name and the number of devices it contains. Use these commands to manage devices in the project:

**Main menu Device, New… or context menu New Device…**

Creates a new device to the project.

**Main menu Device, Import… or context menu Import Device…**

Imports an existing device from another project. If a device is imported from an old PG4 or PG5 version, it will be converted to the new V2 format.
Main menu File, Properties... or context menu Properties...
Display or modify a project's properties: name, description and read-only attribute.

2.2.2 Common Files folder

This folder contains files that can be shared by more than one device in the project. These files can be copied, pasted or dragged into the Program Files folder of each device which shares them. A common file's name shown in the Program Files folder begins with two dots "..", which indicates that the file is in the parent directory, one level up.

The file can be opened from either the Common Files or the Program Files folders. In both cases the same file can be modified and the changes effect every device which references the file.

These are the main commands you will need for adding common files:

File, New... or context menu New File...
Creates and adds a new file to the Common Files folder.

Context menu Add Files...
Copies one or more existing files into the Common Files directory. Not only PG5 program files, but also commissioning and maintenance documents (Word and Excel files etc.) can be added. These files are stored with the PG5 project and can be opened from the Project Tree by double-clicking on them.

2.2.3 Libraries folder

This folder opens the Library Manager window which shows all the available libraries.

The Installed Libraries list shows all the libraries found in the library files directory, which can be used by all projects. The installation path of these libraries is defined by the Tools, Options, Directories, Libraries Directory menu.

The Libraries in Project Directory list shows the libraries which have been copied to the open project's local libraries subdirectory. Only the open project can use these libraries. Installed libraries can be copied into the project using drag-and-drop or by selecting the library and pressing the Copy To Project button.

User's libraries or versions which are not distributed as standard with the PG5 should always be saved with the project, to guarantee that the project is complete and will always build.
If different versions of the same library are present, the library to be used for the build can be selected with the checkboxes in the *Used* column.
2.2.4 Device folder

Each device folder contains the configuration and files for a single controller.

These are the main commands you will need for managing devices:

Context menu Device, Set Active
Activates the device selected in the Project Tree. The active device is indicated by a green triangle, and many main menu and Tool Bar buttons work on the active device. Note: This command is only shown if the option Tools, Options…, Project Manager Activate Device according to Project Tree location is set to No. If Yes, then the Project Manager will automatically active the device according to the selection in the Project Tree.

File, Properties… or context menu Properties…
Display or modify the properties of a device: name, description, read-only option.

Edit, Copy, Paste, Delete
Copy/Paste allows an entire device to be duplicated in the project, with all its files and configuration. Delete removes a device and all its files to the Recycle Bin.

2.2.5 Online Settings

This branch opens the Online Settings window which defines the communications for connecting to the PCD. Several protocols are available: PGU, S-Bus, S-Bus USB, etc., but only the PGU and S-Bus USB allow the full protocol communications needed by the Device Configurator.

The Add button allows new channels to be created with their own types and parameters. These are then visible in the Online Settings list of Channels.
2.2.6 Connecting the PC to the PCD

Channel S-Bus USB
The USB connection is only available for the new PCD2 and PCD3 devices. Use any standard USB cable.

Verifying the connection
Use the Online Configurator toolbar button or menu command to connect to the PCD and display its details. If the information in red is shown then the connection is working properly (the details will vary according to the connected PCD).

If the connection cannot be made, an error message like this is displayed. Check the PCD has power, and verify the Online Settings and the cable connection.
2.2.7 **Device Configurator**

The *Device Configurator* defines the hardware and physical features of the controller, such as the device type, memory, communications channels, fitted modules and I/Os, etc. It also verifies that the power supply provides enough power, and it can print labels for the I/O modules.

To commission a PCD, it is necessary to configure at least the PCD type and its memory size. Other items such as communications and I/Os can be configured later.

The easiest way to start the configuration is to connect the PCD with the USB cable, and read the actual configuration from the PCD using the *Online, Upload Configuration*… menu command or corresponding toolbar button.

If the PCD's memory has not been configured, it may show default settings, so please always verify that this matches your hardware and application.

2.2.8 **Build Options**

These options are used when the user program is built.

### Media Allocation and Dynamic Addressing

<table>
<thead>
<tr>
<th>Symbol Type</th>
<th>Address Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last Timer</td>
<td>49</td>
</tr>
<tr>
<td>Timer Timebase in milliseconds (10..10000)</td>
<td>100</td>
</tr>
<tr>
<td>Has Volatile Flags</td>
<td>Yes</td>
</tr>
<tr>
<td>Last Volatile Flag</td>
<td>1499</td>
</tr>
<tr>
<td>Dynamic Volatile Flags</td>
<td>1000; 1499</td>
</tr>
<tr>
<td>Dynamic Nonvolatile Flags : Used=23, Free=6169</td>
<td>2000; 8191</td>
</tr>
<tr>
<td>Dynamic Timers : Used=0, Free=40</td>
<td>10; 49</td>
</tr>
<tr>
<td>Dynamic Counters : Used=0, Free=600</td>
<td>1000; 1599</td>
</tr>
<tr>
<td>Dynamic Registers : Used=115, Free=6077</td>
<td>2000; 8191</td>
</tr>
<tr>
<td>First</td>
<td>2000</td>
</tr>
<tr>
<td>Last</td>
<td>8191</td>
</tr>
<tr>
<td>Dynamic Texts : Used=1, Free=499</td>
<td>3000; 3499</td>
</tr>
<tr>
<td>Dynamic Data Blocks : Used=2, Free=498</td>
<td>3500; 3999</td>
</tr>
<tr>
<td>Dynamic RAM Texts : Used=0, Free=1600</td>
<td>5000; 6599</td>
</tr>
<tr>
<td>Dynamic RAM Data Blocks : Used=0, Free=1592</td>
<td>6600; 8191</td>
</tr>
<tr>
<td>First writable Text/DB number</td>
<td>4000</td>
</tr>
</tbody>
</table>

**Media Allocation**

This section reserves address ranges for dynamic Registers, Counters, Timers and Flags. When the program is built, addresses are automatically assigned to dynamic symbols defined in the user program and Fupla FBoxes.

A dynamic symbol is one for which no absolute address has been defined:
It is not always necessary to change the dynamic address ranges. The default settings are usually adequate for most applications.

However, if an error message like this appears during the building of a large program:
**Fatal Error 2368: Dynamic space overflow for type: R**
then it will be necessary to extend the dynamic address range for the media type shown in the error message.

If the controller is fitted with EPROM or Flash for main memory, then the RAM Texts and RAM DB dynamic ranges must be configured from address 4000 upwards, so that these Texts and DBs are in writeable RAM memory.

**Last Timer**
Timers and Counters share the same address ranges. The Last Timer value defines the partition between Timers and Counters (it generates the DEFTC instruction). Dynamic Timer addresses must be below or up to this value, and dynamic Counter addresses must be above. For example, if Last Timer is 49, then Timers are T 0..49 and Counters are C 50..1599.

**Timer Timebase in milliseconds**
The default timebase at which Timers decrement is once every 0.1 seconds (100ms). If necessary this can be set to another value. Note that the timebase has no influence on Fupla programs. Only IL programs are affected by this parameter.

**NOTE:** Do not to define an unnecessarily large number of Timers or an unnecessarily small timebase, because this will slow down your program's cycle times.

**Dynamic Nonvolatile Flags**
By default, all Flags are non-volatile. Volatile Flags are always set to 0 at start-up, nonvolatile Flags retain their values. If necessary, the Last Volatile Flag parameter allows a volatile range to be defined. (The picture above shows Volatile Flags for addresses F 0 to F 1499.)
2.2.9 Program Files folder

This folder holds the files that make up the device's program. These are the main commands you will need for managing program files:

File, New… or context menu New File…
Creates and adds a new file to the folder:

### New File [Device]

- **File Name:** Name of the file to be created.
- **Directory:** Directory of the device, cannot be edited.
- **File Type:** The type of file to be created.
- **Description:** Free text which can be used for a description of the file, history, version information etc.
- **Linked/Built:** If not checked then the file will be ignored by the build. It will not be part of the user program.
- **Open File Now:** Checked by default, the file will be opened immediately in the associated editor.

### Device, Add Files… or context menu Add Files…
Adds one or more files to Program Files list. Files can be copied into the device directory, or can be linked by a path, according to the Copy files into device directory option on the Add Files dialog box.
File, Properties… or context menu Properties…
Display or modify the selected file's properties: name, description, Linked/Built and Read Only and Symbol File options. Check Symbol File if the file contains only the definitions of Global symbols.

Files which comprise the device

Files with an arrow in the icon are processed by the build. These files are part of the PCD's program and their code and data are loaded into the PCD's memory.

Files without an arrow on the icon are not processed by the build. These files are ignored and are not downloaded into PCD memory. This can be useful for test and commissioning code which will not be present in the final program.

**Edit, Copy, Paste, Delete**
Copy/Paste allows a file to be duplicated in the current Program Files list or any device in the project. Delete removes a device and all its files to the Recycle Bin.
2.2.10 File Types

A device can have several program files of different types. Each type of file has a corresponding editor for a specific field of application.

**Instruction List Editor (*.src)**

Allows programming in text form using IL instructions. Suitable for all applications. Code is fast and efficient, but it requires a lot of programming experience.

**Fupla Editor (*.fup)**

Allows programs to be drawn in the form of function plans and contact diagrams. Requires no programming experience. Many libraries are available for the rapid implementation of HEAVAC applications and communications networks (modem, LON, Belimo, EIB, etc.).

**Graftec Editor (*.sfc)**

This is a tool for structuring programs in IL (instruction list) and Fupla. Particularly suitable for sequential applications with waits for internal or external events. It is the ideal tool for programming machines with commands for motors, actuators, etc.

**Web Editor Version 8 project (*.sln)**

Editor of web pages for process monitoring and control. Pages can be stored inside a PCD, a web panel (PCD7.Dxxx) or on the hard drive of the PC. Web panels and PCs display the pages on a standard browser such as Internet Explorer, and can use any network for communications.
S-Net Network Configurator (*.dp, *.lon, *.srio)
For configuring device networks and communications: Profibus DP, LON et SRIO.

HMI Editor (*.hmi)
Allows configuration of simple menus with PCD7.D1xx and PCD7.D2xx terminals (installed in addition to PG5).
2.3 Building the Program

The PCD cannot directly process the text files created by the editors. These files must first be compiled and/or assembled, and then linked to create a binary executable ".pcd" file, as shown in the diagram below.

Sources files:
- Graphical Editors
- IL files
- Object and listing files
- Binary PCD File
- PCD Controller

1. Compilation converts graphical files into Instruction List files (*.fbd, *.src, *.hsr)
2. Assembling produces binary object files (*.obj), and an assembly report (*.lst) which can be printed or used for troubleshooting certain assembler errors.
3. Linking combines object files (*.obj) to form a single binary executable file (*.pcd) for downloading into the PCD.
4. IL documentation files are generated by the Project Manager’s Device, Advanced, Create Documentation command. The resulting files are shown in the Documentation Files folder.
2.3.1 Build Changed Files, Rebuild All Files, Rebuild All Devices

The Device, Rebuild All Files command starts the compilation, assembling and linking of all the Linked/Built files for the active device.

The Device, Build Changed Files command does the same thing but only for the files modified since the last build. This is much faster, especially for big programs.

The Project, Rebuild All Programs… command does a Rebuild All Files for each device in the project. Once complete, the Messages window shows the number of devices built with and without errors. Double-click on the red Build Errors message to move to the error messages for that device.

2.4 Messages Window

The Messages window provides information on the progress of a program build. It notes the different stages of the build: compilation, assembling and linking. If the program has been edited correctly, the build ends with the message:

Build successful. Total errors 0 Total warnings: 0

Errors are indicated with a message in red. Double-clicking on an error message will open the editor and select the location of the error, if possible. Selecting the error message and pressing F1 will display help on the error, if it’s available.
2.5 Downloading the Program into the PCD

2.5.1 Download Program

If the build is successful, the Online, Download Program command or toolbar button will download the executable program into the PCD's memory.

**Program File**
Specify the name of the program file downloaded in the active device.

**Destination Device**
Specify the type of destination device and communication channel.

**Before Download, After Download.**
The status of the PCD before and after downloading the program is defined here. If the options Stay in Run and Run the program are selected, the process will undergo minimal disruption by downloading the program. The new program version is downloaded in parallel to the version still being processed. When the program download is complete, the PCD executes a cold start and launches the new version of the program.

**Options**

**Download First-time Initialisation Data**
With this option, some data is only initialised when the program is downloaded. This data initialisation is supported in the programs as follows:

```plaintext
symbol EQU type [address] := initialisation_value
```

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Type</th>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>symbol0</td>
<td>R</td>
<td>10, 314</td>
<td>First-time initialisation value = 324</td>
</tr>
<tr>
<td>symbol1</td>
<td>R</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

Data not initialised on downloading the program can be initialised each time there is a cold restart with an XOB16 block.

**Clear media**
This option clears all PCD data before the program is downloaded: (Registers, Flags, ...)

**Don’t clear outputs**
The program download keeps the current status of the outputs until the end of the program download.

**Backup to Flash**
If the PCD battery is faulty, a power outage may lead to the loss of the program. To prevent this from happening, the program is copied to the internal Flash memory (**Onboard Flash**) or external Flash memory (**Flash card**). If the program in RAM is lost, the copy is automatically restored to the RAM from the Flash memory.

Take care to copy the program to the Flash memory or clear the Flash memory. Otherwise, a different program to the one you want may be restored.

### 2.6 Commands for All Devices

If the PC and all the project's devices are connected together on a network, then **Project, Online Commands** dialog box provides some useful commands for controlling or downloading to all devices on the network.

The dialog box shows a list of all devices in the project, and has checkboxes for selecting the devices to receive the commands. By default all devices are checked. The device list's context menu contains some commands for manipulating these checkboxes.

**Options, Device Sequence**
By default, commands are sent to the devices in the order in which they are defined in the **Project Tree** (alphabetical order). This option displays a dialog box which allows the order to be changed.

**Commands, Test**
This command verifies that each selected device is online and able to receive commands. If it fails, check the **Online Settings** for the device and make sure it is connected and power is on.

**Commands, Download Programs**
Downloads the programs into all the selected devices.

**Commands, Set Clock**
Copies the PC's date/time to every selected device.

**Commands, Run/Stop**
Puts all selected devices into **Run** or **Stop**.
2.7 **Self Downloading Files**

The self-downloading files feature makes it much easier to download programs and configurations into PCDs on site.

A self-downloading `.saiasd5` file is created, which contains all the information needed to update the PCD's program and configuration. The PG5 programmer then simply sends this file by e-mail to the person in charge at the PCD job site.

When you open a `.saiasd5` file, the **Download Self Downloading File** dialog box is displayed. Certain parameters and options on it will match predefined those in the PG5 project. The person present at the job site can either leave these options as they are, or modify them before downloading to the PCD.

This means that no special knowledge of the PG5 is needed to download programs and configurations into the PCDs. The utility works without having to install the PG5 or user licences. However, the **Stand Alone Online Tools** package must still be installed on the PC.

### 2.7.1 Creating a self-downloading file

![Create Self-Downloading File Dialog Box](image.png)

<table>
<thead>
<tr>
<th>File To Create: <code>.saiasd5</code></th>
<th>C:\Users\814826\Sample\PG5\V23.1\text\Projects\Sample\PG5 User Manual\Chapter 6 - Fundamentals</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Logical settings</strong></td>
<td></td>
</tr>
<tr>
<td>User: SautiBurgers Controls AG</td>
<td></td>
</tr>
<tr>
<td>Project: Chapter 6 - Fupla examples</td>
<td></td>
</tr>
<tr>
<td>Device: Daily timer, PCD3 M 862</td>
<td></td>
</tr>
<tr>
<td>Created: 22.06.2018 15:43</td>
<td></td>
</tr>
</tbody>
</table>

**Online Settings:**

- S.Bus USB: PGU

**End User Dialog Box Options:**

- No Dialog Box (show Progress only)
- Show "Advanced ..." button

**Old Firmware, below 1.16.17 [see Help]**

**Actions:**

- Download Program
- Download Configuration
- Update Communications and Options
- Update Password

**Options:**

- Clear Media (F.T.C.F.)
- Don't Clear Outputs
- Clear History
- Set Clock from PC

- Delete old backups from all Flash Cards
- Stay in Stop after download
- Check IP Address won't change
- Verify Serial Number:
  - Fail if different Serial Number
Use the *Device, Advanced, Create Self-Downloading File* command to create a self-downloading file for the *active device*.

The dialog box allows you to configure parameters and options for self downloading at the job site. The options are the same as those we already know from other commands: *Download Configuration and Download Program*, but a few new options have been added.

It is advisable to check that the *Online Settings* and *Device Configuration* are correct, and to perform a successful *build* before creating the *.saiasd5* file.

**File To Create (*.saiasd5)**
Enter the path of the file to be created. Use the browse button > to select a path.

**No Dialog box (Progress only)**
Downloads the *.saiasd5* file without displaying the *Download Self-Downloading File* dialog box. The download starts immediately and only a progress dialog box is shown.

**Show “Advanced >>” button**
Hides the advanced settings from the end-user, so none of the settings can be changed before downloading the file.

**Verify Serial Number**
The downloader checks that the PCD’s serial number matches the one defined in the *Serial Number* field. This serial number is unique to each PCD and can therefore be used to ensure that the download goes to the intended PCD.

**NOTE:** The serial number is only supported by the latest PCD3 systems. The *Online Configurator* can be used to read it online, using the menu *Online, Information command*.

### 2.7.2 Downloading a self-downloading file

![Screenshot of the Self-Downloading File dialog box](image-url)

- **File To Download (.saiasd5)**
Enter the path of the file to be created. Use the browse button > to select a path.

- **Project and Device:**
  - **Project:** Chapter 6 - Fuels examples
  - **Device:** Daily timer
  - **Created:** 22/08/2018 15:46:04

- **Comment:**
  - User: Sais Burgess Controls AG
  - Project: Chapter 6 - Fuels examples
  - Device: Daily timer - PCD3 M950
  - Created: 22/08/2018 15:46:3

- **Online Settings:**
  - 5-DU USB: PGU

- **OK**
- **Cancel**
The Stand-alone Online Tools package must be installed on the PC. For more details please refer to the PG5 installation guide.

Simply open the `.saiasd5` file from Windows Explorer by double-clicking on it. A dialog box like the one shown below will be displayed, so the destination and details can be verified before starting the download. If the 'Advanced >>' button is shown, additional options can be configured before the download, but that's not usually necessary. Start the download process by pressing with the OK button.

### 2.8 Flash Backup Memory

All PCD models have internal Flash memory, and can also be fitted with removable memory with a much greater capacity, using dedicated or I/O slots. Unlike RAM, Flash memory has the advantage that data is not lost when power is switched off. This Flash memory can be used to store a backup copy of the user program, a copy of the PG5 source code for the program, and/or to save data in a file accessible for reading and writing by the PCD's user program.

#### 2.8.1 Saving the executable program

The PCD program is stored in RAM memory. If power is lost and the backup battery is flat then the contents of memory can be lost and the program will not run when power is restored.

To backup the program *(Code/Text/Extension)*, it can be copied to Flash memory which is not altered when power is lost.

The menu Online, Flash Memory, Copy Program To Flash… copies the program to the Flash card, and the command Copy Program From Flash… restores it.

This copying process is automatically supported if the Backup to Flash option is changed in the dialog box menu for downloading the program.

If the user program in RAM is lost, the PCD will automatically restore the program from Flash backup into RAM at start-up.

We recommend using Flash memory on your PCD to protect you from undesirable data loss.
NOTE: The program’s source files must be backed up separately, because only the executable file is loaded into the PCD. See the next section.
2.8.2 Saving the program's source code

The program's source files are stored on the hard drive of the PC. Only the executable code created by the build is downloaded into the PCD's memory. Without the source files it is not possible to modify the programs for updates or maintenance. It is therefore important to create backups, and to always have the latest version available for the maintenance technicians.

There are two ways to save the source files:

- The Project, Backup… command saves all the directories and files into a ".zip" file. This single file makes it easy to retain the directory structure and file layouts. The project is restored from the ".zip" file using the Project, Restore… command. See section 2.1.4 for details.

- The command Online, Flash Memory, Backup Source To Flash… creates the ".zip" backup file and downloads it to the PCD's Flash memory using FTP via an Ethernet connection. The source is restored by uploading and restoring the file using Online, Flash Memory, Restore Source From Flash…. This procedure is described below.

Backup Source to Flash

The command Online, Flash Memory, Backup Source To Flash… starts by compressing the entire project or single device into a standard ".zip" file:
Backup Project or single Device
By default the active device is selected, but this can be changed to create a backup of the entire project.

Name of Backup File
The name of the backup file to be created. To be stored in the PCD's Flash file system it must not be longer than 23 characters, including the ".zip".

Destination PCD with Flash Memory
Shows the PCD to which the ZIP file will be downloaded.

Comment
Free text, but by default it is filled in with a comment containing the user name, project and device names and the date/time. You can also add a revision number and other details here.

Backup What
Selects which files will be backed up. It is not necessary to backup all files, only the source files are important.

When OK is pressed, the ZIP file is created, then the FTP Downloader is started.

FTP downloader

User Name
Name which identifies the user to the FTP server. If no user is defined, use the default: root

Password
Access to the FTP server is protected by a password. If no password has been defined, use the default: rootpasswd
NOTE: The FTP server password and the PCD password may not be the same. The FTP server password is defined in a configuration file in Flash memory called FTPConfig.txt. The password to access the PCD is defined by the Device Configurator.

Save Password
When True, the password is remembered for the next use of this function.

IP Address
IP address of the destination PCD.

PC File Name
The name of the file to be downloaded. By default this is the path of the ".zip" file created in the previous step.

PCD Directory Name
Select this entry and press the button on the right. This obtains a list of Flash memory cards from the FTP server. Select the desired destination for the file.

It may take a little time to load and display the above information. If there are several Flash memories fitted, each Flash is shown with its directory structure. Select the Flash and directory. If necessary, directories can be created and deleted with the associated buttons.

M1_FLASH and M2_FLASH
Flash memory cards fitted in slots M1 and M2 of the PCD.
SL0FLASH, SL1FLASH, etc
Flash memory modules fitted in slots 0..3 of the PCD.

NOTE: This feature is supported only by new PCD models fitted with Flash memory that has the new Flash File System.
Restoring a project or device from Flash

This is done with the command Online, Flash Memory, Restore Source From Flash..... This uploads the "zip" file from Flash memory and decompresses it.

Pressing OK shows the FTP uploader dialog box with the same parameters as show previously for the download operation. Use the PCD Directory Name button to establish communications with the FTP server and to select the Flash memory and directory containing the "zip" file to be restored. Then confirm the upload a restoration of the project or device.

2.8.3 Backing up data to a file

The FBox library called File System supports access to data files in Flash memory. These files can be read and written and can be up and downloaded via FTP.

For more information about the number and types of Flash memory which are available, the configuration of the FTP server, and the possibilities for creating data files using Fupla or IL programs, please refer to the following manuals:

SBC FTP Server and SBC Flash File System
Hardware manual for the PCD3 series
Hardware manual for the PCD2.M5
Hardware manual for the PCD1.Mxxx0
2.9 The View Windows

The information displayed in these windows may not be completely accurate if build errors have occurred, but in PG5 V2 they will show all the available information. In V1.x they were always empty if a build error had occurred.

2.9.1 Block Call Structure

A Saia PCD program is a tree-like structure of organization blocks which contain the application's code. Each block provides a particular service: cyclical programming (COB), sequential programming (SB), sub-programs (PB), functions with parameters (FB), and exception routines (XOB).

The overall structure of the blocks that make up the program can be seen by pressing the Block Call Structure toolbar button or selecting the View, Block Call Structure menu command.

The example below shows a program comprising blocks: COB 0, COB 1, XOB16, PB 10, PB11 and FB 156. Note that COB 0 conditionally calls three sub-blocks (PB 10, PB 11 and FB 156). The call condition is indicated in brackets.

```
[COBs - Cyclic Organization Blocks]
[ ] COB 0 - Regulation
  [ ] PB 10 [H] - Summer_Control ;Program for summer time
  [ ] PB 11 [L] - Winter_Control ;Program for winter time
 [ ] FB 156 [H] - Maintenance_Control ;Program for maintenance time
  [ ] FB 12 - Test.Install ;Test your installation
  [ ] FB 13 - Diag.Instal ;Installation diagnose
 [ ] COB 1 - Light_Control

[XOBs - Exception Organization Blocks]
[ ] XOB 16 - Init ;Initialise your parameters
```

Call condition:
H: Accu = 1
L: Accu = 0
2.9.2 **Global Symbols and Data List Views**

The **View All Symbols** and **View Data List** menus display all symbols for the active device. These symbols can only be edited in the files defining them. Unused symbols are not displayed.

2.9.3 **Cross-Reference List**

The **All Publics Symbols** and **Data List** views offer the possibility of selecting a symbol and showing its cross-reference list, which is a list of all program locations where the symbol is used.

Each entry shows the file name and block in which the symbol selected is used, with a line or page number too. It also shows if the could be changed at that location with the word Written.

The **Definitions** list shows where the symbol is defined, e.g. where its IL EQU statement can be found. The **References** list shows where the symbol is used in the program.

For blocks, ‘>>’ indicates where the block itself can be found.

To view the program where the symbol is used, select the definition or reference and press the **Goto** button.
2.10 The Online Configurator

### Online Settings
Communications parameters for PCD connection

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PCD Type</strong></td>
<td>The connected PCD type</td>
</tr>
<tr>
<td><strong>Version</strong></td>
<td>Version of firmware in the PCD</td>
</tr>
<tr>
<td><strong>Program Name</strong></td>
<td>Name of the program in the PCD (device name)</td>
</tr>
<tr>
<td><strong>Date</strong></td>
<td>Date from PCD's clock, if present</td>
</tr>
<tr>
<td><strong>Time</strong></td>
<td>Time from the PCD's clock, if fitted</td>
</tr>
<tr>
<td><strong>Day</strong></td>
<td>Day of the week: 1 = Monday, ... 7 = Sunday</td>
</tr>
<tr>
<td><strong>Week</strong></td>
<td>Week of year 1..52</td>
</tr>
<tr>
<td><strong>Status</strong></td>
<td>Operating mode: Run, Stop, Halt, Conditional Run</td>
</tr>
</tbody>
</table>

If the information shown in red is not present, or an error message dialog box appears, it means that the *Saia PG5 Online Configurator* could not communicate with the PCD.

Please check the following:
- That the PCD is correctly connected with the USB cable, or to the network, and it's powered up.
- That the communications parameters are correct by pressing the Online Settings… button.

### 2.10.1 Device Configurator

Uploads the configuration and opens the Device Configurator. Refer to the Device Configurator documentation.
### 2.10.2 PCD History

The History window shows all the hardware or software errors which have occurred while the PCD was operating. History messages are always added to this list, even if the associated XOB handler is programmed. Examine this list if the PCD's Error lamp is lit.

![History Window](image)

**NOTES:**
- If the error occurred on a program line, the Address shows the line number. If not, it will show a hexadecimal reference.
- XOB 0 (power off) messages are only shown if XOB 0 is programmed.

### 2.10.3 Setting the PCD's Clock

Most PCDs have a built-in real-time clock which provides the date and time. The Clock... button displays the above dialog box which allows the date/time to be viewed and adjusted. The Copy to PCD >> button immediately copies the date/time of the PC into the PCD's clock. Alternatively, the date and time can be entered manually and are transferred when OK is pressed.
2.10.4 Saving program and data from RAM

This is an interesting command which allows the save and restore of the user program and configuration, and also all the Registers, Flags, Timers, DB, Texts etc. present in the PCD's RAM memory. This is very useful for copying programs to other PCDs, when duplicating an installation, changing the PCD, or simply to restore the PCD to a previously saved state.

**Tools, Upload All…**
Uploads and saves all of RAM into a file of type ".im5" (PG5 memory image file).

**Tools, Download All…**
Downloads a ".im5" file into the PCD's RAM.
2.10.5 **Create Diagnostic File**

This useful feature creates a file containing all the information needed when requesting help from Saia Burgess Controls technical support team. The file contains details about the PCD type, firmware version etc.

Use the **Tools, Create Diagnostic File**... command and note the path of the file which was created.

2.10.6 **Firmware Downloader**

For a PCD to be able to use the latest functionality it may be necessary to update the firmware in the PCD. This is easy to do because all PCDs store the firmware in Flash memory. The current firmware version can be seen on the main window of the **Online Configurator**.

Firmware is downloaded using the **Tools, Firmware Downloader command**, available from Project Manager or the Online Configurator.

The **Add** button adds a new firmware file (.blk) to the **Files to download** list. The list remembers the last file which was downloaded. For special applications the list could contain several files, but for normal use please ensure that only the firmware file for the connected PCD type is present. The most recent firmware files are available in a directory on the PG5 distribution CD.

The **Options, Online Settings**... command defines the communications parameters, usually using **S-Bus USB** or **PGU mode**.

Pressing the **Start** button to begin downloading the firmware. After a few moments, a progress dialog box will appear.

When completed, the **Run, Halt and Error** LEDs will flash while the PCD does some memory management. Please wait for about 1 minute after they stop flashing before powering off the PCD or continuing to work.
3 DEVICE CONFIGURATOR ................................................................. 3

3.1 Scanning the Device parameters to the configurator ............................................. 3

3.2 The main view on the configurator ..................................................................... 4

3.3 Loading the configurator parameters to the Device .......................................... 5

3.4 Device properties ............................................................................................... 6
   3.4.1 Memory ........................................................................................................... 6
   3.4.2 Password ......................................................................................................... 7
   3.4.3 S-Bus ............................................................................................................... 8
   3.4.4 Power Supply .................................................................................................. 8

3.5 Serial S-Bus communication properties ............................................................. 8
   3.5.1 Full protocol (PGU) Serial-S-Bus (PGU slave for series line) ...................... 9
   3.5.2 Public Line S-Bus Modem (PGU slave for a modem line) ......................... 9
   3.5.3 Serial S-Bus Master Gateway ....................................................................... 9
   3.5.4 S-Bus Mode and Timings ............................................................................. 10

3.6 Profi S-Bus communication properties .............................................................. 11
   3.6.1 Profi-S-Bus (slave) .......................................................................................... 11
   3.6.2 Profi S-Bus Master Gateway ........................................................................ 12
   3.6.3 Bus Parameters: user defined ...................................................................... 13

3.7 Ether-S-Bus communication properties .............................................................. 14
   3.7.1 Ether-S-Bus (slave) ........................................................................................ 14
   3.7.2 Profi S-Bus Master Gateway ........................................................................ 15

3.8 On board slots properties, configuration of the media mapping ....................... 16
   3.8.1 Device properties, necessary configurations. .............................................. 16
   3.8.2 Onboard slots, configuration of the E/S modules ...................................... 16
   3.8.3 Properties of binary I/O ............................................................................. 17
   3.8.4 Properties of analogue I/O ........................................................................ 18

3.9 Printing of labels for the I/O modules ............................................................... 19

3.10 Extension of the Device Configurator by means of new devices and I/O modules .. 20
3 Device configurator

The Saia PG5 Device Configurator enables configuration of the Device’s equipment parameters: Device type, memory, communications channels, input/output modules, but it also checks the power consumption by the input/output modules on the internal power supply of the PCD and prints labels for sticking on the I/O modules.

In order to place a PCD device into service it is necessary to define at least the type of PCD and its memory configuration. The other configurations can then be added according to the required services, such as communication networks or I/O handling.

For those familiar with versions PG5 1.4 and older, the Device Configurator is an entirely new program, presenting configuration parameters with a new human-machine interface organized in a very different way. Although new configuration options are available, both Devices and their configuration parameters remain the same as before.

3.1 Scanning the Device parameters to the configurator

The simplest method to realize the configuration consists in linking the computer to the PCD using a USB cable and scanning the configuration already present in the PCD’s memory using the menu Online, Upload Configuration… or the corresponding button in the tool bar. If the memory turns out to contain no configuration, PCD firmware undertakes to return the appropriate information, then check the configurations corresponding to your application.
3.2 The main view on the configurator

The main view on the configurator represents the different equipment features making up the Device.

**Device slot**
Indicates the current Device type. If it is selected, it displays the properties of the Device such as RAM/EPROM/Flash internal memory size, the password, use of the S-Bus, of input/output handling, options and power consumption on the I/O bus.
To change the type of the device, point the mouse on the current Device and select the context menu Change Device Type … The Properties context menu enables the relevant properties window to be displayed.

**Memory slots**
Memory slots available in order to receive Flash memory extensions corresponding to the type of the selected Device. To configure the memory slots, select and drag the corresponding memory modules from the I/O Selector window to their slots. To open the E/S Selector window select the menu View, Selector Window.
Onboard Communication slots
The internal communication slots represent the available communication interfaces. Depending on the PCD types, the number of communication interfaces is different, they are only configured if they are utilized by the PCD program.

Some slots are pre-defined, others are freely configurable and require definition of the corresponding communication modules by selecting them in the E/S Selector window and dragging them to a slot.

Selecting a communication slot displays the parameters of the communication interface in the properties window and enables them to be defined.

Onboard slots
Represents the input/output slots available on the Device. The available E/S modules are configured by selecting them in the E/S Selector window in order to drag them to an E/S slot. As is the case with the other slots, selecting one of the E/S slots displays the relevant configuration parameters.

Expansion slots
The expansion slots designated by means of ‘+’ can receive a bus expansion module in order to add extra E/S modules. Configuration of an expansion module is carried out by selecting the corresponding module in the E/S Selector window in order to drag it to the extension slot indicated with ‘+’.

3.3 Loading the configurator parameters to the Device
Equipment configuration of the Device is necessary on first utilization but also in the event of all changes such as memory extension, setting up a communication module, etc.

These parameters are not only configured in the device configurator but also loaded in the Device’s memory by means of the Download Configuration menu or the corresponding button.

Attention, loading the program from the Project Manager does not load the configurations in the Device. The configurations must be loaded from the Device configurator.
3.4 Device properties

3.4.1 Memory

Code/Text/Extension memory

 Represents the RAM available on the Device in order to receive the program as well as texts and data blocks included between addresses 0 …3999, but also texts and data blocks with an address higher than 3999.

This memory may be organized differently on some older PCD’s. The extension memory is separate from the Code/Text memory. We therefore have two parameters:
**Code/Text**

Represents the RAM or EPROM available on the Device in order to receive the program as well as texts and data blocks included between addresses 0 …3999

This memory may also be EPROM on some older PCD’s. Whereas the new ones provide the opportunity to carry out a program backup on a Flash memory.

**Extension Memory**

 Represents the RAM available for data blocks with addresses higher than 3999.

**User Program Memory Backup size (Flash)**

Indicates the size of the internal Flash memory of the Device, if it is insufficient, some types of devices support a Flash extension memory on one or a number of external slots. See configurations under Memory Slots.

The Flash memory is used in order to realize program backup after loading on the Device. Backup must be activated in the menu Tools, Options of the Project Manager or realized on request by the Project Manager by means of selecting the menu Online, Flash Backup/Restore.

If a power outage causes a loss of the program in RAM, when power is restored the Device Firmware automatically restores the program present on the Backup memory.

The Flash memory also supports other applications such as data backup or the source files of the project.

### 3.4.2 Password

The Devices have a protection mechanism by means of a password. If there is password protection, only the reduced communication protocol can be used. It only authorizes access to the registers, timers, counters, indicators, inputs, outputs, data blocks and the system clock. The other data, such as the user program, S-Bus configuration and the historical table, cannot be accessed. The orders Run, Stop, Step etc. are placed out of operation.

Setting up communication on a password protected Device displays a dialogue for password entry before enabling use of the full protocol. Selecting the Cancel button sets up communication in any case, with the restrictions of the reduced protocol.

In the event of forgetting the password, the PCD memory must be wiped and reprogrammed in order to configure it without a password or with a new password.

**Wiping the memory**

This is very simple for RAM. It is simply necessary to cut the Device’s power supply and remove the battery for a while.

For EPROM, the memory chip must be removed from the Device and wiped by means of the UV lamp, then reprogrammed with an EPROM programmer by means of the user program and a known password.

For Flash memories, the memory chip must be removed from the Device and wiped by means of an EPROM programmer which supports Flash memory chips.

On reading the hardware settings, the PCD password is always absent.
Otherwise, this would enable unknown passwords to be read in order to then load them into another PCD which would become inaccessible until the memory was removed and wiped.

3.4.3 S-Bus

If one of the communication interfaces defined under Onboard Communication or Onboard Slots is used, start by activating the S-Bus support and defining the S-Bus station number.
The S-Bus station number is common to all of the Device’s communication channels.

3.4.4 Power Supply.

The 5 and 24 Volts power supply required for the functioning of I/O modules is provided by the Device's bus. It is advisable to check that the configuration of the I/O defined under Onboard Slots of the configurator is not higher than the values for the maximum available current.

3.5 Serial S-Bus communication properties

Serial S-Bus is a masters/slave network enabling the Devices to be networked on a RS 485/232 series line in order to exchange data between PCDs and supervise the process. It also supports all service functionalities by means of the programming tool PG5 (PGU) and an analogue or ISDN modem.

Depending on the series communication slot selected, the properties window enables configuration of the Serial-S-Bus channel as a PGU slave for a series line, PGU slave for a modem line or master gateway. Start by activating the desired configuration then add the active parameters.

<table>
<thead>
<tr>
<th>Port Number</th>
<th>Serial S-Bus Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Port Number

Number of the communication channel, this number may be used by program instructions in order to designate the assignment channel (SASI) or to transmit data exchange telegrams.
3.5.1 *Full protocol (PGU) Serial-S-Bus* (PGU slave for series line)

Extra *Serial-S-Bus* slave communication slots can be configured using a reduced protocol (without the PGU function). Only the PGU slot is configured from the configurator, the slave function, without the PGU function, is configured from the program Fupla/IL with the assistance of the instruction SASI.

Note, the S-Bus address is defined in the device’s properties.

3.5.2 *Public Line S-Bus Modem* (PGU slave for a modem line)

This configuration is only available for series communications slots which provide all control lines required for linking with a modem. Provides the same services as *Full protocol (PGU) Serial-S-Bus* but through a telephone line and an analogue or ISDN modem. To enjoy simplified operation, we recommend using SBC modems.

*Modem Name*

Enables selection of the type of modem used. It is recommended to use Saia PCD modems, thus they are already present in this list, the parameters *Modem Init* and *Modem Reset* are already configured and tested.

Note, the S-Bus address is defined in the device’s properties.

3.5.3 *Serial S-Bus Master Gateway.*

The *Gateway* function is commonly used to link two different communication networks, adapt a Saia PG5 programming tool, for VisiPlus supervision, a modem line on the Serial-S-Bus network or for creating a multi-master network.

All telegrams received from external masters and which are not meant for the *gateway* station are automatically redirected to the master *gateway* channel.

*First/Last S-Bus Station*

Enables filtration of telegrams to be sent on the *Gateway* according to the target station numbers.
3.5.4 S-Bus Mode and Timings

<table>
<thead>
<tr>
<th>S-Bus Mode And Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-Bus Mode</td>
</tr>
<tr>
<td>Baud Rate</td>
</tr>
<tr>
<td>Response Timeout [ms]</td>
</tr>
<tr>
<td>Training Sequence Delay [ms]</td>
</tr>
<tr>
<td>Turnaround Delay [ms]</td>
</tr>
</tbody>
</table>

**Mode**
Choice of the method applied by S-Bus in order to indicate the start and finish of the messages, must be identical on all stations making up the network. Preferably choose the data mode, this method is suitable for all applications, in particular for the use of modems or standardized network equipment.

**Baud rate**
Speed of communication, must be identical on all stations in the network.

**Response Timeout**
Waiting time in milliseconds by the master station in order to receive the response to a telegram. This timing is particularly important when the slave does not respond. If it is too long this will severely slow down communication. If it is too short the responses arrive too late and cause the telegram sent by the master to fail.

By default the value of the timings is zero, this means that the default timings are applied. In practice we only change them when Gateways or modems are used: generally we restrict ourselves to stretching the Timeout on the external masters.

**Training Sequence Delay (TS)**
Timing in milliseconds between activation of the RTS (Request To Send) signal and sending of the message. With S-Bus, the activation of RTS enables drivers RS-485 or RS-422 to be selected.

**Turnaround Delay (TN)**
Minimum timing in milliseconds between the end of a response and transmission of the next telegram. This gives the outlying station time to switch the line from emission mode to reception after having sent a message. The TN is particularly important when PCD7.T100 receptors or modems for private lines are used.
3.6 **Profi S-Bus communication properties**

Profi-S-Bus is a multi-master field bus based on the Profibus FDL standards and the SBC S-Bus protocol. It enables the Devices to be networked in order to exchange data between PCDs and supervise the process. It also supports all service functionalities by means of the Saia PG5 programming tool. (PGU)

The properties window corresponding to the S-Net communication slot enables a Profi-S-Bus slave or master gateway channel to be configured. Start by activating the desired configuration then add the active parameters.

### 3.6.1 **Profi-S-Bus** (slave)

Defines Profi-S-Bus channel as slave or PGU slave. This definition may be complemented by a master function by adding a SASI master instruction in the Fupla or IL program.

**Full Protocol PGU**

PGU Slave (Yes)
Supports the exchange of data with master stations, supervision systems and terminals, but also supports the programming tool and tool for setting up PG5.

Slave (No)
Only supports the exchange of data with the master stations, supervision systems and terminals.
Address
Address of the station on the Profibus FDL network. In order to fully designate the Device with which telegram exchange takes place, two addresses are required, the Profibus FDL address and the S-Bus address defined in the device’s properties.

Baud rate Profi-S-Bus
Speed of communication, must be identical on all stations in the network.

Bus Profile
The timings for transmission are grouped into three profiles:

- **User-defined**: the parameters are defined by the user
- **S-Net**: uses the values which are most suitable for the Saia PG5 network Configurator.
- **DP**: uses the values which are most suitable for the Profibus DP network.

The Profile must be identical on all stations in the network. The S-Net profile is required when RIO PCD3.T76x is used on the networks.

3.6.2 Profi S-Bus Master Gateway.
The *Gateway* function is commonly used to link two different communication networks, adapt a Saia PG5 programming tool, for VisiPlus supervision, a modem line on the Profi-S-Bus network.

All telegrams received from external masters and which are not meant for the gateway station are automatically redirected to the master gateway channel.

First/Last S-Bus Station
Enables filtration of telegrams to be sent on the Gateway according to the target station numbers.

Response Timeout
By default the value of the *Timeout* is zero, this means that the default timing is applied. In practice we only change it when the Gateways or modems are used: generally we restrict ourselves to stretching the *Timeout* on the external masters and never on the gateway channel.
3.6.3 Bus Parameters: user defined

<table>
<thead>
<tr>
<th>Bus Profile</th>
<th>User defined</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bus parameters</strong></td>
<td></td>
</tr>
<tr>
<td>Slot Time</td>
<td>300</td>
</tr>
<tr>
<td>Min. Tsd</td>
<td>11</td>
</tr>
<tr>
<td>Max. Tsd</td>
<td>150</td>
</tr>
<tr>
<td>Quiet time</td>
<td>0</td>
</tr>
<tr>
<td>Setup time</td>
<td>1</td>
</tr>
<tr>
<td>Gap update factor</td>
<td>10</td>
</tr>
<tr>
<td>Highest station address</td>
<td>126</td>
</tr>
<tr>
<td>Max. retry limit</td>
<td>1</td>
</tr>
</tbody>
</table>

The properties of the bus contain all of the Profi-S-Bus communication parameters. These values can only be edited if the **User-defined** bus profile is selected. The parameters must be the same for all stations on the network.

All timings are indicated in bit (number of bits).

**Slot Time**
Maximum amount of time the data or token emitter waits for the addressee's response.

**Min. Tsd**
Minimum waiting time by a slave after having received a telegram until emission of the response to the master.

**Max. Tsd**
Maximum waiting time by a slave after having received a telegram until emission of the response to the master.

**Quiet Time**
Waiting time by the emitter after the end of a frame before activating its reception.

**Setup Time**
Time elapsed between an event and the corresponding reaction.

**Gap Update Factor**
Number of rotations by the token between two GAP (update) cycle maintenances.

**Highest Station Address**
The highest address (HSA) on the network

**Max. Retry Limit**
Number of repetitions without acknowledgement of receipt before sending a negative acknowledgement of receipt.
3.7 Ether-S-Bus communication properties

Ether-S-Bus is a multi-master network based on Ethernet standards and the SBC S-Bus protocol. It enables the Devices to be networked in order to exchange data between PCDs and supervise the process. It also supports all service functionalities by means of the Saia PG5 programming tool. (PGU)

![Diagram of Ether-S-Bus communication properties](image)

Ether-S-Bus slave PGU channel

The properties window corresponding to the S-Net communication slot enables configuration of an Ether-S-Bus slave or master gateway channel. Start by activating the desired configuration then add the active parameters.

3.7.1 Ether-S-Bus (slave)

Defines Profi-S-Bus channel as slave or PGU slave. This definition may be complemented by the master function by adding a SASI master instruction in the Fupla or IL program.

<table>
<thead>
<tr>
<th>TCP/IP</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel Number</td>
<td>9</td>
</tr>
<tr>
<td>TCP/IP Enabled</td>
<td>Yes</td>
</tr>
<tr>
<td>Ethernet R10 Network</td>
<td>None</td>
</tr>
<tr>
<td>IF Address</td>
<td>0.0.0.0</td>
</tr>
<tr>
<td>Subnet Mask</td>
<td>255.255.255.0</td>
</tr>
<tr>
<td>Default Gateway</td>
<td>0.0.0.0</td>
</tr>
<tr>
<td>+Access Control List</td>
<td>Show</td>
</tr>
<tr>
<td>IF Filtering Enabled</td>
<td>No</td>
</tr>
<tr>
<td>IF Filtering Policy</td>
<td>White List</td>
</tr>
<tr>
<td>IF Filtering List</td>
<td>Configure</td>
</tr>
</tbody>
</table>

**IP Node**

Number of the TCP/IP node. The node is used in the Device’s application program in order to reference the IP address of the slave station with which data is to be exchanged.

**IP Address**

Address of the station on the Ethernet network. In order to fully designate the Device with which telegram exchange takes place, two addresses are required, the Ethernet address and the S-Bus address defined in the device’s properties.

**Subnet mask**

Defines the part of the IP address which belongs to identification on the network and identification of the host. All bits corresponding to the network are at 1 while the bits...
corresponding to the host are at zero. The network identifier is the result of a logical AND combination between the IP address and the subnet mask. Each TCP/IP host requires a subnet mask, even on a network segment. The subnet mask by default is therefore 255.255.255.0 (ClassC)

**Default Router.**
Address of the default router.

**PGU port**

PGU slave (Yes)
Supports the exchange of data with master stations, supervision systems and terminals, but also supports the programming tool and tool for setting up PG5.

Slave (No)
Only supports the exchange of data with the master stations, supervision systems and terminals.

**Network Groups**
Displays the dialogue window in order to configure the Device as client or server on the network group.

### 3.7.2 Profi S-Bus Master Gateway.

The Gateway function is commonly used to link two different communication networks, adapt a Saia PG5 programming tool, for VisiPlus supervision, a modem line on the Ether-S-Bus network.

All telegrams received from external masters and which are not meant for the gateway station are automatically redirected to the master gateway channel.

**First/Last S-Bus Station**
Enables filtration of telegrams to be sent on the Gateway according to the target station numbers.

**Response Timeout**
By default the value of the Timeout is zero, this means that the default timing is applied. In practice we only change it when the Gateways or modems are used: generally we restrict ourselves to stretching the Timeout on the external masters and never on the gateway channel.
3.8 On board slots properties, configuration of the media mapping

The Device’s input/output handling can be managed with the assistance of media mapping made up of indicators/registers updated by the Device’s firmware. The Fupla or IL program no longer directly accesses the inputs/outputs for the read/write operations but works with the media mapping.

The media mapping is configurable with PCD1.Mxxx0, PCD2.Mxxx0 and PCD3 devices. However, it is still possible to access the inputs/outputs directly without configuring the media mapping. This makes it possible to guarantee compatibility with former projects by previous versions of PG5 and devices which do not yet support it: PCD1, PCD2.M1xx, PCD2.M480.

3.8.1 Device properties, necessary configurations.

If the Device’s firmware supports the media mapping, I/O handling can be activated or deactivated by means of the parameters below.

<table>
<thead>
<tr>
<th>Input/output handling</th>
<th>Input/Output Handling Enabled</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Peripheral Addresses Definition</td>
<td>Auto (recommended)</td>
</tr>
</tbody>
</table>

Input/Output Handling Enabled
Yes, all E/S parameters on the module are available to support configuration of the media mapping.

No, all of the parameters defined in Onboard slots are without influence on the user program.

Peripheral Address Definition
Automatic or manual definition of peripheral addresses for the media mapping of each module.

3.8.2 Onboard slots, configuration of the E/S modules.

Enables configuration of the I/O modules present in the device by selecting them in the E/S Selector window in order to drag them to an I/O slot.
Selecting one of the I/O slots displays the relevant configuration parameters in a properties window.

Configuration of the I/O slots is not necessary if the media mapping is not supported by the PCD or not activated in the Device’s properties.

### 3.8.3 Properties of binary I/O.

Enables definition of the configurations required to elaborate the media mapping. If the Media mapping selection is not displayed, this means that the Device’s firmware selected under Device does not support this functionality.

![Properties of binary I/O.](image)

**Base Address**

*Base address of the module: 0, 16, 32, etc.*

**Enabled Media Mapping**

Cyclical update of the media mapping (register or indicators) with the input, output values present on the Device’s slots.

**Media Type**

Type of medium used to backup the input or output values. For analogue I/O, the type is always register. For binary I/O, the type is indicator by default but can also be register.

**Number of Media**

Number of media required to backup the values. For example, a PCD3.E110 digital input module requires 8 indicators.

**Media Mapping**

The new Media Mapping view shows the symbol names for all the I/Os which are mapped to media (Registers and Flags). This view can be shown or hidden using the "Media Mapping" command under the “View”.

The addresses for the input and output symbols can be dynamic or absolute. It is no longer necessary to define consecutive addresses in an array of symbols (except for RIO media or Firmware compatibility).

The media mapping supports external symbols which are defined in other files with Public scope (Fupla, IL or symbol files). These can be filled in quickly using drag-and-
drop from the Symbol Editor. If you have defined a Fupla application where some public symbols need to be linked to an input or output, just drag-and-drop them into the Media Mapping view and onto the correct I/O.

According to the build of the program, these symbols are available in order to elaborate the Fupla et IL programs, we find them among the All Publics symbols under: S.IO.Slot0, etc.

3.8.4 Properties of analogue I/O.

Configured as for the binary I/O modules, we find the same parameters in order to define the Media mapping of the analogue module on registers.

On the other hand, a new section provides the parameters required for configuration of the I/O magnitudes of each channel.
3.9 Printing of labels for the I/O modules.

The Tools, Label menu enables display of the window below in order to prepare labels to be placed on the PCD3 I/O modules and in the PCD1.Mxxx0 and PCD2.Mxxx0 cover. Select the label and edit the properties window.

![Label Editor Window]

Width and colour of lines printed on the labels.

**Border Visible**
Traces the edge of the label in order to facilitate cutting. Note, pre-cut sheets are available for printing the labels.

The size of the printed labels can appear twice the width because the intention is to fold them in two lengthwise. They are therefore more stable in their holders.
3.10 Extension of the Device Configurator by means of new *devices* and I/O modules

When new modules or devices become available, it is no longer necessary to install a new version of the Device Configurator or of PG5 in order to support them. It is sufficient to install the XML file which describes the new hardware in a PG5 installation sub-directory and restart the software.

C:\Program Files (x86)\SBC\PG5 V2.3.160\DeviceTemplates
### PCD DATA

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>4.1 Hardware Data</strong></td>
<td>4</td>
</tr>
<tr>
<td>4.1.1 Digital inputs and outputs</td>
<td>4</td>
</tr>
<tr>
<td>4.1.2 Date and time</td>
<td>5</td>
</tr>
<tr>
<td>4.1.3 Interrupt inputs</td>
<td>6</td>
</tr>
<tr>
<td><strong>4.2 Internal Data</strong></td>
<td>7</td>
</tr>
<tr>
<td>4.2.1 Flags</td>
<td>7</td>
</tr>
<tr>
<td>4.2.2 Registers</td>
<td>8</td>
</tr>
<tr>
<td>4.2.3 Constants</td>
<td>9</td>
</tr>
<tr>
<td>4.2.4 Timers and counters</td>
<td>10</td>
</tr>
<tr>
<td>4.2.5 Texts and data blocks</td>
<td>13</td>
</tr>
</tbody>
</table>
PCD Data

This section gives an overview of the types of data which can be used in the PCD programs (Inputs, Outputs, Flags, Registers, Counters, Timers etc.), their uses and address ranges.
4.1 Hardware Data

Programs contain many instructions to read, write and modify many different kinds of data. The data which allows the program to interact with the outside world is accessed via the controller's hardware.

4.1.1 Digital inputs and outputs

Inputs and outputs represent binary values (one bit) from the outside world. Inputs show the states of switches, pushbuttons, proximity detectors, sensors, etc. Outputs can activate valves, lamps, motors, etc.

Outputs can be written and read. Inputs can only be read. Inputs and outputs are fitted by plugging I/O cards into the PCD's module slots. The start address of the I/Os on the card is defined by its slot position.

Example

The following example turns on output $O_{64}$ if inputs $I_1$ and $I_2$ are both high. Another way to show such functions is by using a boolean equation:

$$O_{64} = I_1 \text{ AND } I_2$$

Instruction list program: 

```
COB 0
  0
STH I 1
ANH I 2
OUT O 64
ECOB
```

Fupla program:

```
Fbox: Binary, Arithmetic, And
```

![Diagram](image)
4.1.2 Date and time

A real-time clock (RTC) is built into the PCD. The date and time can be loaded into a Register with an IL instruction or an FBox.

Example

The following example shows how to read the date/time.

Instruction list program:  

Fupla program:

```
COB 0
0
RTIME R 1
ECOB
```

FBox: System information, Read time

This program reads the time from the clock and copies the value into registers R 1 and R 2 as decimal numbers, for example:

\[ R_1 = 093510 \quad \text{hhmms} = 09:35 \text{ and } 10 \text{ seconds} \]
\[ R_2 = 74081231 \quad \text{wdymmd} = \text{week 07, day 4 (Thursday), 31/12/08} \]
4.1.3 Interrupt inputs

Some PCDs have two interrupt inputs called INB1 and INB2. Whenever there is a rising edge on one of these inputs, the normal program cycle is interrupted and the PCD executes a special program block called XOB 20 or XOB 25 (XOB 20 for INB1 and XOB 25 for INB2). These inputs are capable of a frequency up to 1000 times per second.

Example

The example demonstrates how to count pulses from INB1.

Instruction list program:

```
COB 0 ;main program
0...
ECOB

XOB 20 ;interrupt INB1
INC R2 ;increments ;register R2
EXOB

...120
```

Fupla program:

```
COB 0

```

Limits imposed by the input filtering (protecting a normal digital input against interference and bounce from mechanical contacts) prevent the input from counting pulses with a frequency higher than 50 Hz. Interrupt inputs therefore represent an interesting alternative solution for this kind of application. They bypass the need to use PCD2/3.Hxxxx counting modules, which have a maximum counting frequency between 10 to 160 kHz, depending on the module type.

For more details, see the PCD hardware manuals

FBox: Integer, Arithmetic, Add
4.2 Internal Data

4.2.1 Flags

A flag contains a single binary bit of data, either 0 or 1. By default, all the flags are *non volatile*. This means that if you turn off the PCD when the flag is a 1, it will still be a 1 when you turn the PCD back on (assuming the RAM backup battery isn't flat). All *volatile* flags are reset to 0 when the PCD is turned off. The number of *nonvolatile* flags is defined from the device's *Build Options*, see below.

**Example**

The following example writes a high (1) to flag number 11 as soon as input 1 or input 3 is high. Boolean equation: \( F_{11} = I_1 \ OR \ I_3 \)

Instruction list program:

```
COB 0
  0
STH I 1 ; if I 1 is 1
ORH I 3 ; and I 3 is 1
OUT F 11 ; then set F 11
ECOB
```

Fupla program:

```
[11] >= 1
```

FBox: *Binary, Arithmetic, Or*

**Configuring the number of nonvolatile flags**

The partition between volatile and non volatile flags is defined in the device's *Build Options* in the Project Manager. The ranges for *dynamic addressing* can also be defined there. Dynamic addressing is described in the next section.
4.2.2 Registers

Registers can contain signed 32-bit integer or floating point values.

Registers are useful for arithmetic operations, such as processing analogue values, particularly in measurement and regulation applications.

All registers are non-volatile, which means that their values are not lost when the PCD is powered off.

In Fupla, the “wires” or lines connected to registers have different colours depending on the data type: yellow lines for floating point values and green lines for integer values. An integer value cannot interact directly with a floating point value, one of the values must first be converted into the format of the other.

Example

The following example adds the integer 113 to the contents of register 12, and puts the result into workspace register 54:

\[ R_{54} = R_{12} + 113 \]

Instruction list program:  

```
COB 0
   0
ADD R 12
   K 113
   R 54
ECOB
```

Fupla program:  

```
R 12 + 113 R 54
```

FBox: Integer, Arithmetic, Add

The 'K' indicates that '113' is a constant, not another register. 'K' values are unsigned values from 0..16383. To use a negative or larger value, first load it into another register with the LD instruction or Integer/Constant FBox, and add the registers.

Dynamic register addressing

Dynamic address allocation is a powerful feature introduced to free you from having to specify a fixed address for every workspace register and flag in the program. Addresses are allocated if you define a symbol name and type, without specifying the address, this lets the Build assign an unused address for the symbol. The address range used for dynamic address allocation is configured from the device's Build Options. If a build error like this occurs: Dynamic address overflow for type: R, increase the number of dynamic addresses for that data type.

![Device configuration options](image)

Double mouse-click
4.2.3 Constants

Constants are fixed values that can be used directly, or loaded into registers, timers or counters. Constants can be integer or floating point numbers, e.g. 100, 0.25, or Pi 3.1415. Timers and counters accept only unsigned integer values 0..2,147,483,647.

Example

The following example loads register R 4 with an integer value (100), which is divided by floating point value (0.25). Because register R 4 contains an integer and we want to divide it by a floating point value, R 4 must first be converted from integer to floating point. First, copy R 4 to R 35 (a work register which we know is not being used) and convert R 35 to a floating point value, then divide R 35 by 0.25 which has been loaded into R 36. The floating point result of the division is put in R 5. R 5 is then copied to R 6 to be converted back to an integer value (some of these steps are not really necessary, they could use the same workspace registers.)

Instruction list program:

```
COB 0       ; cyclic block
    0
LD R 4     ; load 100 into R 4
    100
COPY R 4   ; convert integer R 4 to floating point
    R 35
IFP R 35   ; in R 35 0 = 10^0
    0
LD R 36    ; load 0.25 into R 36
    0.25
FDIV R 35  ; divide R 35 by 0.25
    R 36
    R 5
    ; put result in R 5
COPY R 5   ; convert result back
    R 6
    0
FPI R 6    ; to integer in R 6
ECOB
```

Fupla program:

```
Load an integer constant into a register
100

Convert an integer value to floating
1.0000000E+02

Int Fp Div RF 5
100
2.5000000E-01

Convert a floating value to integer
400

Fp Int Err RS
RF 5
4.0000000E+02

FBox:
- Integer, Arithmetic, Move
- Conversion, Integer to, Floating points, Int to float
- Floating point, Arithmetic, Divide
- Conversion, Floating point to, Integer, Float to Int
```
4.2.4 Timers and counters

Timers and counters can hold values between 0 and 2,147,483,648 (31 bits), and they share the same address range: 0 to 1599. The number of timers is defined by the device's Build Options, for example 0..31 are timers and 32 to 1499 are counters. Timers have a default "timebase" of 100ms, which means that the timer is decremented by 1 every 100ms, and so counts down 10 times a second.

Usually addresses 0 to 31 are dedicated to the timers, and addresses 32 to 1599 are dedicated to the counters. The user can, of course, configure personal settings. Timers have a default time-base of 100ms (i.e. the system decrements each timer by one every 100ms). The timebase can be changed in the Build Options, where timer / counter addresses can also be configured. Timers are volatile, counters are not.

Timers and counters can only contain positive values. Their value can be changed by loading a new value with the LD instruction. Timer values decrease only. Counters can count up or down, using the INC/DEC instructions. (INC: ↑, DEC: ↓).

Timers and counters can also be used with binary instructions. When a timer or counter contains a non-zero value, its state is High (1). When its content is zero, its state is Low (0).

Configuring timers and counters

The distribution of the address range between timers and counters can be altered in the Build Options. This is also where you can change the time-base of 100 ms.

Technical information

The more timers you declare, the greater the load on the PCD. This is also true if you lower the time-base. Take this into consideration before you change the number of timers or lower the time-base.

Example: 100 timers will take about 2% of the PCD's capacity.
Example: Timer

Input 4 goes to 1. On the rising edge of this input, Output 65 is also set to 1. It remains at 1 for duration of 2.5 seconds.

Instruction list program:

```
COB 0 ;Cyclic block 0
  0
STH I 4 ;If input 4
DYN F 12 ; sees a rising edge
LD T 1 ;Load timer 1
  25 ; with 2.5 seconds
STH T 1 ;Copy timer state
OUT 0 65 ; to output 65
ECOB
```

Fupla program:

```
XPluse FBox:
  Timer, Pulse, Exclusive pulse

14 Mn Q O 65
25 TV t
```

Technical information

Timers in the Saia PCD are decremented at a rate defined by the Build Options, Timer Timebase. (normally 100ms). The actual time defined by a constant which is loaded into a timer changes if the time-base is changed. This means that if the time-base setting is changed, then all timer load values must also be changed. To overcome this problem, the time data type can be used to declare timer load values. If a time value is used, then the linker calculates the actual timer load value according to the time-base settings.

Format: \text{T#}nnnS|MS
Example: Counter

A counter will be programmed to count up each time input 5 turns on, and down each time input 6 turns on. These counts will be activated on the rising edge of the input. The counter can be zeroed via input 2. The initial count will be loaded with 3.

Instruction list program:

```
COB 0 ;Cyclic block

STH I 1 ;If input 1 is 1
LD C 35 ;then load counter 35
  3 ; with 3
STH I 2 ;If input 2 is 1
LD C 35 ; then load counter 35
  0 ; with zero
STH I 5 ;On rising edge of
DYN F 13 ; input 5
INC C 35 ;increment counter 35
STH I 6 ;On rising edge of
DYN F 14 ; input 6
DEC C 35 ;decrement counter 35
```

Fupla program:

```
FBox:
  Counter, Pulses counter,
  Up down with preset and clear
```

Counter value

Counter state:

A counter's state is 1 if it contains a non-zero value:

```
STH C 35
```
4.2.5 Texts and data blocks

Texts (character strings) and data blocks (DBs, arrays of 32-bit values) are non-volatile. Texts are used for: messages for displays, transmissions to pagers, command strings for modems, etc. DBs are used for data logging, tables etc.

Texts and DBs share the same addresses. If Text 1 is defined, there cannot be a DB 1. Texts and DBs 0..3999 are in main program memory, which may be read-only. Texts and DBs 4000 and above are in extension memory, which is writable RAM.

The maximum number of Texts and DBs is 16383.

Technical information

Registers, flags, timers and counters are handled by the system and stored in separate RAM memory. DBs and Texts are stored in the main memory, together with the user program. If you want to use a Flash or EPROM for your main memory, your program can read from this memory but not write to it. Therefore you cannot alter the content of your DBs (for example your data logging). In most cases you will not bother with this, but if you know that you are going to read and write your DBs, then make sure to use DBs which are stored in the extension (data) memory => Texts/DBs 4000 and above. Extension (data) memory is always RAM, which means you can read and write to it.

Examples of Text and DB declarations

TEXT 10 "Bonjour!" ;Text 10 contains the string Bonjour!

TEXT 11[7] "Hello" ;Text 11 is 7 chars long, the first 5 are Hello and the last two are spaces

DB 12 45,46,78,999,0 ;DB 12 has the 5 integer values: 45,46,78,999,0

DB 13 [10] ;DB 13 is 10 values, which are all zero

DB 14 [4] 2,3 ;DB 14 is four values, the first two are 2 and 3, the last two are 0
Text definition with the symbol editor

Data bloc definition with the symbol editor
Summary table

<table>
<thead>
<tr>
<th>Description</th>
<th>Media</th>
<th>Operand</th>
<th>Binary</th>
<th>Numeric</th>
<th>Volatile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>I</td>
<td>1) 0…8191</td>
<td>0,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Outputs</td>
<td>O</td>
<td>1) 0…8191</td>
<td>0,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flags</td>
<td>F</td>
<td>0…16383</td>
<td>0,1</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>Registers</td>
<td>R</td>
<td>0…16383</td>
<td>-2 147 483 648…+2 147 483 647</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>Constantes</td>
<td>K</td>
<td></td>
<td>-2 147 483 648 à +2 147 483 647</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timers</td>
<td>T</td>
<td>2) 0…31</td>
<td>0,1</td>
<td>0 … 2 147 483 648</td>
<td>Yes</td>
</tr>
<tr>
<td>Counters</td>
<td>C</td>
<td>2) 32…1599</td>
<td>0,1</td>
<td>0 … 2 147 483 648</td>
<td>No</td>
</tr>
<tr>
<td>Texts</td>
<td>X</td>
<td>3) 0…3999</td>
<td>String of max. 3072 characters</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4) 4000 …</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Blocks</td>
<td>DB</td>
<td>3) 0…3999</td>
<td>Max. 382 values (slow access)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4) 4000 …</td>
<td>Max.16 383 values (fast access)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1)  Depends on PCD model and its fitted I/O cards
2)  The actual number of timers is configured from the *Build Options*.
3)  Texts/DBs 0..3999 are stored in program memory (RAM / EPROM / FLASH)
4)  Texts/DBs 4000 and above are stored in extension (data) memory (RAM)
5 SYMBOL EDITOR .........................................................................................................................2

5.1 Elements of a symbol..................................................................................................................2
  5.1.1 Symbol Name .......................................................................................................................2
  5.1.2 Syntax for the name of a symbol .........................................................................................3
  5.1.3 Type ..................................................................................................................................3
  5.1.4 Address/Value ....................................................................................................................3
  5.1.5 Comment ...........................................................................................................................4
  5.1.6 Actual Address ..................................................................................................................4
  5.1.7 Tags ..................................................................................................................................4
  5.1.8 Scope ................................................................................................................................4

5.2 Editing the symbols ......................................................................................................................4
  5.2.1 Absolute addressing..........................................................................................................5
  5.2.2 Symbolic addressing.........................................................................................................5
  5.2.3 Dynamic addressing .........................................................................................................5
  5.2.4 Symbol Editor, quick editing of symbols ........................................................................6
  5.2.5 IL or Fupla editor, quick editing of symbols ......................................................................7
  5.2.6 Symbol groups ..................................................................................................................7
  5.2.7 Symbol tables ...................................................................................................................8
  5.2.8 All Publics, System, ... views .........................................................................................8
  5.2.9 Visibility of symbols, Local, Public and External scope ....................................................9
  5.2.10 Searching for symbols ..................................................................................................10
  5.2.11 Using a symbol in the program ......................................................................................10

5.3 Initialising the symbols ..............................................................................................................11
  5.3.1 Initialising on cold restart ...............................................................................................11
  5.3.2 Initialising on downloading the program .........................................................................11
  5.3.3 Initialising texts and DBs ...............................................................................................12

5.4 Predefined symbols .....................................................................................................................13
  5.4.1 Symbols predefined by FBoxes .........................................................................................13
  5.4.2 System symbols ...............................................................................................................15

5.5 Cross-references of symbols .....................................................................................................16

5.6 Sharing symbols between the program files ..............................................................................16
  5.6.1 ‘.sy5’ symbol files ............................................................................................................17
  5.6.2 ‘.xls’ symbol files ............................................................................................................18
  5.6.3 ‘.rxp’ symbol files ............................................................................................................18
  5.6.4 Defining symbols or programs in a common file .............................................................19

5.7 Importing/exporting symbols .....................................................................................................20
  5.7.1 Importing & merging symbols ..........................................................................................20
  5.7.2 Exporting symbols ...........................................................................................................21
  5.7.3 Importing ‘EQU’ declarations .........................................................................................21
  5.7.4 Renumbering Addresses ...............................................................................................21
5 Symbol Editor

5.1 Elements of a symbol

The symbol editor can be accessed from the program editors, Fupla, instruction list (IL) and Graftec, by pressing the Show or Hide the Symbols Editor button or using the View/Symbol Editor menu.

The Symbol Editor supports the list of all variables manipulated using the programs: symbol name, type, address/value, comment, scope, etc.

There is always a blank row at the end of the list or a group to define new symbols.

5.1.1 Symbol Name

A symbol is a name representing the address of an input, output, flag, register, constant, text, DB or structure block (COB, PB, FB, XOB, SB, TR, ST). We recommend using symbols in the programs and not directly using the addresses of registers, flags, etc.

Assigning meaningful comments and names to symbols makes the programs easy to read and understand. For example, we can assign symbol 'pump_boiler1 to O 32 and use this symbol to represent this output in the programs. The comment can be used to expand the information relating to a symbol: 'feed pump for boiler 1'.

Symbols can be used to correct the address or type of data via the Symbol Editor without having to copy the correction to each program instruction/connector. The corrections made to a symbol definition are automatically propagated to all program instructions/connectors that use this symbol. This approach significantly restricts the risk of errors that could then be difficult to trace.
5.1.2 Syntax for the name of a symbol

The first character is always a letter followed by further letters, digits or the underline character.

The underline character must never be used as the first character of a symbol. Symbols with an underline as the first character are reserved for internal symbols within PG5 and FBox libraries.

The first characters of a symbol name must not start with the character S followed by a full stop. These are the first characters reserved for system symbols within PG5 and FBox libraries.

It is recommended that special characters are avoided, such as ö, è, ç, etc.

Upper or lower case letters do not affect the functioning of the program. The symbols Motor_On and MOTOR_ON are two identical symbols.

Symbols corresponding to reserved words must not be used:
- Assembler instructions: PUBL, EXTN, EQU, DEF, LEQU, LDEF, MACRO, ENDM, EXTM
- Short notations and command codes of the various types of data in the PCD: I, O, F, R, C, T, K, M, COB, FB, TEXT, X, SEMA, DB
- Special instructions: MOV: N, Q, B, W, L, D
- Conditional codes: H, L, P, N, Z, E
- All mnemonic instructions
- Predefined symbols
- Internal symbols reserved for automatically assigning resources
- Starting with an underline character. For example: ______TEXT, _________F
- Internal symbols __CSTART__, used with $$.

5.1.3 Type

This defines the type of symbol: Input (I), output (O), register (R), counter (C), timer (T), text (X), DB, ...

The program only supports the writing of texts and DBs of type Text Ram, DB Ram. If such texts and DBs still cannot be written, check whether the dynamic address range corresponding to these types of symbols is greater than 3999. See under build options in the project manager.

5.1.4 Address/Value

Each type of symbol has various available addresses:

Inputs and outputs: According to the I/O modules inserted in the machine
Flags: F 0, ..., F 8191, 16383
Registers: R 0, ..., R 4095, 16383
Timers, counters: T/C 0, ..., T/C 1599
...
5.1.5 **Comment**

The comment is linked to its symbol and can be displayed in the program.

Using the Fupla editor, hover the mouse cursor over a connector to display the full definition in an info-bubble.

It is also possible to select the symbol in a connector from the Fupla page, in this case, the Symbol Editor presents the corresponding symbol definition, with its comment.

If the IL editor is used, the comments are shown in green and start with a semi-colon. They are displayed to the right of the mnemonic and operand, but may also take up a whole row.

```
$SKIP
******************************************************************************
* AUTEUR:    Dupond Fred
* DATE:      10.06.2003
* FICHER:    logic.SRC
******************************************************************************
$ENDSKIP
```

A comment over several rows does not necessarily have to start with a semi-colon at the start of each row; it is also possible to write the entire text between two assembly directives ($skip and $endskip). The assembler does not take into account any of the text within these two instructions.

Alternatively, the **View User or Auto Comment** button displays the comment of the instruction line or symbol.

```
STM Flag ; Copie l'état du flag dans l'accumulateur
STM Flag ;Active incrémentation
```

5.1.6 **Actual Address**

If the symbol address is not specified by the user, this column displays the dynamic address assigned for **build**.

5.1.7 **Tags**

The tags are used to associate symbols sharing a common characteristic, symbols exchanged over the communication network, symbols for the web or supervision system, etc. Using the filtering function, the Symbol Editor can display the symbols marked with a tag in the same view.

5.1.8 **Scope**

The scope defines the visibility of the symbols: Local symbols are only visible by the program file defining them. The **Public** symbols are visible by all files present in the **device**. The **External** symbols are defined outside the opened file.

5.2 **Editing the symbols**

There is always a blank row at the end of the list or a group, so that new symbols can be defined by completing the **Symbol Name, Type, Address/Value, Comment and Scope**
fields. Defining a symbol does not necessarily include all the information previously presented. If necessary, a row can also be inserted using the Insert context menu or by pressing the Ins key.

5.2.1 Absolute addressing

<table>
<thead>
<tr>
<th>Symbol Name</th>
<th>Type</th>
<th>Address/Value</th>
<th>Comment</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Untitled2.fup</td>
<td>ROOT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Daily Timer</td>
<td></td>
<td>0</td>
<td>Daily Timer</td>
<td>Local</td>
</tr>
</tbody>
</table>

The symbol is defined with a type, address and optionally a comment. Using absolute addresses can be problematic when correcting the type or address of the symbol. The correction cannot be propagated from the Symbol Editor. The correction has to be manually copied over for each program instruction/connector. To avoid having to do this, it is preferable to use symbolic or dynamic addressing.

5.2.2 Symbolic addressing

<table>
<thead>
<tr>
<th>Symbol Name</th>
<th>Type</th>
<th>Address/Value</th>
<th>Comment</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Untitled2.fup</td>
<td>ROOT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Daily Timer</td>
<td></td>
<td>0</td>
<td>Daily Timer</td>
<td>Local</td>
</tr>
</tbody>
</table>

The symbol is defined with a name, type, address and optionally a comment. Any corrections to the symbol made in the Symbol Editor are automatically propagated to each program instruction/connector using this symbol.

5.2.3 Dynamic addressing

<table>
<thead>
<tr>
<th>Symbol Name</th>
<th>Type</th>
<th>Address/Value</th>
<th>Comment</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Daily Timer.fup</td>
<td>ROOT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HMS</td>
<td></td>
<td>R</td>
<td>PCD Clock w...</td>
<td>Local</td>
</tr>
</tbody>
</table>

The symbol is defined with a name, type and optionally a comment. The programming tool automatically assigns an address corresponding to the defined type when the program is build. The address is taken from a range of addresses reserved by the Build Options (see project manager). Note: Dynamic addressing is available with the flags, counters, timers, registers, texts DB, COB, PB, FB, SB. However, the addresses still have to be defined for the inputs, outputs and XOB.
5.2.4 Symbol Editor, quick editing of symbols

It is possible to define the new symbol from the Symbol Name field. If the new symbol is defined according to the syntax `symbol_name type address ;comment`, just press the enter key to automatically fill the corresponding cells with the necessary data.

The Symbol Name field can also be used to define a whole list of symbols by pressing just one key.

If necessary, the various definition fields can be corrected using the Stretch function. For example, correct the address of register 9 with address 100, then click on the bottom right corner of the cell to drag it downwards. All the expanded cells will then be corrected in line with the original cell.
5.2.5 IL or Fupla editor, quick editing of symbols

New symbols can also be added from the Fupla connectors or IL instructions based on the same syntax as the previous method.

Press the enter key to automatically add the new symbol to the Symbol Editor view (only if the symbol has been correctly defined).

5.2.6 Symbol groups

If a program calls a function relating to different symbols a number of times, it is not necessary to find the different symbol names for each of the functions or to index them. Symbols support the notion of groups. The group is a symbol inserted as a prefix to the symbol name. Within the programs, the group and symbol are separated by a full stop. To insert a group in the Symbol Editor, use the New Group context menu or shortcut CTRL + G. The group names can be concatenated.

5.2.7 Symbol tables

If the address of a symbol is supplemented with a whole value in square brackets, this symbol is a table. Each type of symbol supports tables. The symbol address can be static or dynamic and corresponds to the address of the first element in the table. The other elements in the table are consecutive addresses. The whole value in square brackets is the size of the table. It is possible to access the various elements of the table using indexed or referenced symbols.

Indexed symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR_Datas1[3]</td>
<td>Is an equivalent symbol to AR_Datas1[3].</td>
</tr>
</tbody>
</table>

Symbol AR_Datas1 corresponds to register 10.
Symbol AR_Datas1[2] corresponds to the second register in the table (R 10 + 2).
Symbol AR_Datas1[3] is an equivalent symbol to AR_Datas1[3], (R 10 + 3).
It is another way of indexing a table of symbols.

The advantage of these notations is to create a build error, if the index is outside the table:
Symbol AR_Datas1[10] creates a build error, the size of this table is 5 elements and it only supports symbols AR_Datas1[0] to AR_Datas1[4].

Referenced symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR_Datas</td>
<td>Array of 5 statics registers (Base address register = 10)</td>
</tr>
<tr>
<td>Value1</td>
<td>Array element 1</td>
</tr>
<tr>
<td>Value2</td>
<td>Array element 2</td>
</tr>
<tr>
<td>Value3</td>
<td>Array element 3</td>
</tr>
<tr>
<td>Value4</td>
<td>Array element 4</td>
</tr>
</tbody>
</table>

A symbol is created to access each of the elements in a table. The address of these symbols refers to the address of the first element in the table. If the address of the table is corrected with a new address, the referenced symbols are automatically assigned the new address.

5.2.8 All Publics, System, ... views

The Symbol Editor displays a number of views:

The view with the name of the open file supports the list of all symbols defined and used by this file. If several files are opened simultaneously, the symbols relating to each of the files are presented with independent views with the name of each of the files.

The ALL Publics view supports the list of all shared symbols. Symbols defined with Public visibility are available to all files of the device.

The System view displays the list of all system symbols. Symbols predefined by PG5 or by certain FBoxes within the Fupla program.

The symbols of the All Publics and System views are updated if the file is saved or when the program is build.
5.2.9 Visibility of symbols, *Local*, *Public* and *External* scope

The user freely shares the definition of the symbols between the various program files of a *device*. For example, the *Daily Timer.fup* and *Ventillation.src* files.

Some of the symbols are shared with all the files of the *Device* and other symbols are not.

Visibility of a symbol among the files of the *Program Files* directory is adjusted by assigning *Local*, *Public* or *External* scope.

A symbol with *Local* scope is only visible by the file defining it.

A symbol with *Public* scope is visible by all files of the *device*.

A symbol with *External* scope is a reference to a *Public* symbol defined outside the open file, but included in the *Program Files* directory.

There is only one file in the *Program Files* directory defining a symbol with *Public* scope. All other files using this symbol only define a reference with *External* scope.

Symbols with *External* scope are automatically added to the view of the symbols with the name of the open file when we drag the symbol of the *All Publics* view towards the program.

A *Public* symbol or *External* reference has to be corrected in the file defining the symbol with *Public* scope.

The *All Publics* view and symbols with *External* scope cannot be edited.

To add or correct such a symbol, select the *Public* symbol in the *All Publics* view, or the *External* symbol, then use the *Goto Definition* context menu to open the file containing the definition of the symbol with *Public* scope or select the symbol with the mouse.
5.2.10 Searching for symbols

If there is a large number of symbols in the editor, the Find tool can be used to locate a character string in the view. Enter the character string to be found in the Find field. Use the Next and Previous buttons to navigate to the next or previous symbol. Advanced options are available to select columns in which the search is to be carried out.

5.2.11 Using a symbol in the program

The symbols already defined in the Symbol Editor window can be used in different ways:

Various options are available to place the symbols in the program.

**Keying in the symbol**

The full name of the symbol is keyed in for each connector/instruction using it. This approach allows for the possibility of editing the symbol name if there is a typographical error, which only becomes apparent when the program is built.

**Keying in by clicking and dragging**

This way of using a symbol leaves no scope for typographical errors.

In the Symbol Editor view, position the mouse cursor over the button at the start of the row defining a symbol, click and hold down the left-hand mouse button. Drag the mouse cursor onto a Fupla connector or an IL instruction line and release the mouse button. The symbol selected is automatically inserted in the target position.

It is also possible to drag the symbol to a free space on the Fupla page, this will automatically add the connector and symbol in one operation.

Several consecutive symbols can be dragged into the program at the same time. Click on the first symbol, press the Shift key, click on the last symbol, then drag the selection to the program. If the symbols to be dragged are not consecutive, press the ALT key instead of the Shift key.
5.3 **Initialising the symbols**

During power outages, data such as registers, flags and counters are saved by the battery or capacitor for the next time the PCD is started.

The symbols relating to this data are therefore in an undefined state the first time the PCD is started after downloading a program.

This may sometimes even cause a malfunction in the process to be automated.

To protect against this, such symbols are initialised with the services available.

5.3.1 **Initialising on cold restart**

The symbols are generally initialised, by automatically running a program in cold start block XOB 16, which is called each time the PCD is powered up again, cold restart.

For example: Initialising a flag and register when the PCD is given a cold restart

```plaintext
XOB 16 ; Cold start
LD R 5 ; R 5 = 256
256
SET F 10 ; F 10 = 1
EXOB
```

5.3.2 **Initialising on downloading the program**

To initialise a symbol on downloading a program to the PCD, follow the symbol address with := then the initialisation value.

For example:

<table>
<thead>
<tr>
<th>Group/Symbol</th>
<th>Type</th>
<th>Address/Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>BL_3E315C55</td>
<td>COB</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SymbolA</td>
<td>R</td>
<td>5:= 256</td>
<td></td>
</tr>
<tr>
<td>SymbolB</td>
<td>F</td>
<td>10:= 1</td>
<td></td>
</tr>
</tbody>
</table>

**NB:**

Don't forget to initiate the next option on downloading the program:
5.3.3 Initialising texts and DBs

The Symbol Editor can also be used to define symbols for texts and DB data blocks; the *Address/Value* cell shows a button that opens a view to define the size of the text or DB as well as the initial text or data, which is applied each time the program is downloaded.

<table>
<thead>
<tr>
<th>Symbol Name</th>
<th>Type</th>
<th>Address/Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Untitled1.fup</td>
<td>ROOT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMS_Message</td>
<td>TEXT</td>
<td>100</td>
<td>...</td>
</tr>
<tr>
<td>MeasuresList</td>
<td>DB</td>
<td>200</td>
<td></td>
</tr>
</tbody>
</table>

With DBs, start by defining the size of the block, then click on *Create* to create the table of indexes, values and comments for the block.

It is not necessary to define the values for each index; indexes without values are initialised by default with the value of zero.

If the DB is large, it is not necessary to create the grid with all the indexes. You only have to enter the required indexes. Click on *Clear* to edit the index and its value in the available cell. A new row will automatically appear when we edit the blank row.
5.4 Predefined symbols

5.4.1 Symbols predefined by FBoxes

Some libraries, for example E-Line and DDC, insert predefined symbols in the file during editing. The predefined symbols are available if the library uses them and the option View, Options, Display predefined symbols = Yes.

The predefined symbols can be used by the other FBoxes in the program, for example supervision system or Web panels.

For example:

Insert the following FBoxes in a Fupla file:

- E-Line, Initialization Channel, EL + S-Bus Master
- E-Line, Mixed RIO, EL-PCD1.B1000

Each FBox displays the dialog box to insert the predefined symbols in the Symbol Editor.

FBox EL-B1000 has inserted a group of symbols (EChannel.PCD1_B1000_0) in the Symbol Editor. If necessary, the user can change the name of the root group using the dialog box or directly in the Symbol Editor.

This mini program is a quick example of using a remote E-Line input/output module. It can be used to configure the E-Line PCD1.B1000 device, read and write inputs and outputs, and support manual control over the outputs.
FBox EL+ MS-Bus Master assigns the serial channel RS 485 of the master PCD. The address of the Channel has to be adjusted with the hardware used and network cabling.

FBox EL-B1000 represents the device with remote outputs and inputs; this device must be powered and cabled via the network, two small switches can be used to define the S-Bus address. This address must match the S-Bus address parameter of the Adjust window.

To read and write the remote inputs and outputs, the program of the master station uses the predefined symbols when the FBox is inserted into the Fupla page.

<table>
<thead>
<tr>
<th>Symbol Name</th>
<th>Type</th>
<th>Address/Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>EChannel</td>
<td>GROUP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCD1_B1000_0</td>
<td>GROUP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parameters</td>
<td>GROUP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manual</td>
<td>GROUP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DigitalOutput0</td>
<td>GROUP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DigitalOutput1</td>
<td>GROUP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DigitalOutput2</td>
<td>GROUP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DigitalOutput3</td>
<td>GROUP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>F</td>
<td>(EChannel.PCD1_B1000_0.DigitalOutput1)</td>
<td>Output value</td>
</tr>
<tr>
<td>ManMode</td>
<td>F</td>
<td>(EChannel.PCD1_B1000_0.DigitalOutput1)</td>
<td>State of the software manual mode</td>
</tr>
<tr>
<td>ManModeHw</td>
<td>F</td>
<td>(EChannel.PCD1_B1000_0.DigitalOutput1)</td>
<td>State of the hardware manual mode</td>
</tr>
<tr>
<td>ToggleManMode</td>
<td>F</td>
<td>(EChannel.PCD1_B1000_0.DigitalOutput1)</td>
<td>Enable/disable the manual model</td>
</tr>
<tr>
<td>ToggleManValue</td>
<td>F</td>
<td>(EChannel.PCD1_B1000_0.DigitalOutput1)</td>
<td>Toggle/edit the manual value</td>
</tr>
</tbody>
</table>

Each remote output has a number of predefined symbols dedicated to manual control. These symbols are used to deactivate the programmed control of an output and replace it with manual control, managed from the supervision system or Web panels and the symbols described below.

Force the ToggleManMode symbol in high state to change from programmed control to manual control of the remote output. The symbol automatically drops to low state once the control mode has been changed.

The ManMode symbol indicates the control mode of the remote output. Low for programmed control and high for manual control.

The ToggleManValue symbol is the manual control of the remote output; it toggles the output state when a rising edge is detected.

Whatever control mode is in effect, the Value symbol indicates the state of the output.

To restore programmed control over the remote output, again force the ToggleManMode symbol to high state.
5.4.2 System symbols

The Symbol Editor has a System view displaying the list of all system symbols. Symbols predefined by PG5 or by certain FBoxes within the Fupla program.

Symbols predefined by PG5

The symbols within PG5 are made available to the user program via system symbols.

This system symbol defines the version of the program edited by the user with the properties of the device.

These symbols define the firmware version configured with the properties of the configurator device.

Other system symbols define the address of the PGU port, S-Bus address...

Symbols predefined by FBoxes

Some FBox libraries, such as the CVC library, sometimes create system symbols to allow the editing of change parameters via the supervision system or web panels.
5.5 Cross-references of symbols

It is sometimes useful to find the definition of a symbol, Fupla page or IL program line using a symbol. All this information is quickly available via the cross-reference.

After the program has been built, select a symbol in the Symbol Editor, a Fupla connector or an IL instruction. The Cross-reference List context menu displays the data found.

![Cross-reference List](image)

Each row of this dialog box represents a Fupla connector or an IL instruction line using this symbol. The symbol is modified by the program for each reference to which the word Written is added. Select the various rows of the cross-reference list to navigate to the Fupla pages or IL instruction using this symbol.

To find the definition of a symbol, select the symbol in the connector of the Fupla page or an IL instruction line, then use the Goto Definition context menu to open the file containing the definition of the symbol.

5.6 Sharing symbols between the program files

The symbol definitions are integrated in the Fupla or IL program files. The Public and External scopes enable these symbols to be shared with programs of other device files.

Where necessary, it is also possible to define symbols in a file outside the program. A file without a program and only with the symbol definitions. The defined symbols in such files are always Public and are displayed in the All Publics view, or a new view of the Symbol Editor which is called the name of the file defining the external symbols.

It is not necessary to import or merge the external symbol files with the program symbols. The external symbol files in the project simply have to be updated with the files created by another service, planning office, CAD, etc.

The symbols defined using external files are displayed in the All Publics view or are updated by running the program build.

These files may have different formats:

- *sy5* File to be edited using the PG5 Symbol Editor.
- *xls* File to be edited using Excel.
- *txt* File to be edited using a text editor
- *rxp* File created by the VisiPlus supervision system

External symbol files with extension ‘sy5’, ‘xls’ and ‘rxp’ are edited using the program corresponding to the file extension and are saved in the original format.
5.6.1 `.sy5` symbol files

The .sy5 file is inserted into the project in the same way as a Fupla or IL program file.

Select the Program Files directory in the Project Manager, click the right-hand mouse button, then go to the New… menu or press the New File button in the toolbar. Edit the file name in the File Name field, then the type of file (*.sy5)

The file is added to the project manager and opened using the Symbol Editor. It is possible to create new symbols or modify existing ones.

On opening the program files present in this device, the symbols of this file are displayed in the All Publics view of the Symbol Editor. The symbols can only be edited in the source file, but they can be freely dragged to the program.
Global symbols with a PG5 project Version 1.4 or older.

If a PG5 1.4 project is imported into version PG5 2.x, the definitions of global symbols in PG5 1.4 are saved in file `Globals.sy5`. These symbols are displayed in the All publics view of Symbol Editor version 2.x

### 5.6.2 ‘.xls’ symbol files

Excel files or files of another format can be inserted following the same method as for .sy5 files.

The first row of the EXCEL file must be completed with the following information: `SYMBOLNAME, TYPE, …`

The EXCEL sheet must be called `symbol`.

The symbols are edited following the example of `Symbol0`.

An example of the format required to edit a file can easily be provided by exporting the symbols from any program file.

Double-click on the `Supervision.xls` file to open the file by default with the Symbol Editor; it is now possible to correct or insert new symbols. The file is always saved in the original format (.xls).

To open the file in EXCEL, click on the file, then open the Open With… context menu to select the editor.

If we open the FUPLA file, the symbols of the EXCEL file are displayed in the All Publics view with the symbols from file `Symbols.sy5`.

### 5.6.3 ‘.rxp’ symbol files

Take the same steps as with the examples shown for .sy5 and .xls files.
5.6.4 Defining symbols or programs in a common file

The program and symbol files in the Common Files folder can be shared by several devices in the project. A common file is referenced by dragging the common directory to the devices that use it or vice versa. The device indicates the reference by preceding the file name with two dots and one backslash: ..\CommonProgram.fup or ..\CommonSymbols.sy5.

The file can be opened and edited from the Common directory or the device referring to it. There is no copy of the file, it is a unique file, corrections are applied to all devices referring to it.
5.7 Importing/exporting symbols

5.7.1 Importing & merging symbols

Symbol files outside PG5 (.sy5, .xls, .txt, .rxp) can be imported into the FUPLA or IL program. This is done with the Advanced, Import… context menu of the Symbol Editor. Importing the external symbol files is not without problems, as the symbols of the external file are merged with the symbols of the IL or Fupla file. Any conflicts between the symbols of the two files have to be resolved by the user. To prevent this from happening, we recommend not importing the external symbol files, but inserting them into the project as shown in the section "Sharing symbols between the program files".
5.7.2 Exporting symbols

The symbols of the Fupla or IL program can likewise be exported to a file in .sy5, .xls, .txt, .rxp format. This is done by selecting a number of consecutive or non-consecutive symbols and then using the Advanced, Export Selected Symbols… context menu of the Symbol Editor.

5.7.3 Importing ‘EQU’ declarations

If we still have old PG3/4 files with ‘EQU’ or ‘DOC’ declarations, it is possible to import these declarations to the Symbol Editor. This is done by selecting the zone containing the declarations, then going to the ‘Tools’ menu and selecting ‘Move Symbols to Symbol Editor’.

5.7.4 Renumbering Addresses

The old "Excel drag" feature for renumbering addresses has been replaced by a new "Renumber Addresses" command.

First, select the symbols to be renumbered. These can be the same types, or mixed types. You can even select all the symbols (Ctrl+A) and renumber all of them. A different base address can be defined for each type.

Tip: You can filter and/or sort the symbols first to make it easy to select the symbols you need.

In the example below, Register and Flag types have been selected:
Now run the context menu's the "Tools > Renumber Addresses..." command to open the «Renumber Addresses» dialog box. This shows a list of the selected symbol types, e.g. "R" and "F", and allows the types to be selected with the checkbox, and a new base address to be defined for each type.

You can also select the "Renumbering Mode" which defines how the addresses will be created.

- **from base address**: Selected symbols are renumbered consecutively starting from this address. The value column title shows "Base address" when this option is selected.
- **add offset**: An offset is added to the addresses of the selected symbols. The value column title shows "Offset to add".
- **clear address**: This converts the selected symbols from an "absolute address" to a "dynamic address". The addresses of the selected symbols are removed, so the addresses are generated dynamically when the program is built. The value column is hidden when this option is selected.

**Tip**: Dynamic addresses are renumbered using Project Manager's "Device >
Advanced > Clean Files" command to delete the "Symbol information files".

In the example, we will renumber the selected Registers starting from base address 1000. The Flags, even though some flags are selected, are not renumbered because the checkbox is clear.

Below is the result. Symbols which were modified are selected.

Notes

- Symbol array sizes are taken into account, but only if the size can be evaluated (maybe it's an external symbol or expression). In the above example, Releve_Mensuel_1 R 1003[14] skips 14 addresses, so Report gets address R 1017.

- If a symbol already uses a symbol for a base address then it is not renumbered. See Val_Max R Report+1 above. "Report" is renumbered, so the Address/Value of Val_Max does not need to be changed.

- Renumbering addresses may cause conflicts - other symbols may have the same address. This can have unpredictable effects. **Tip**: Check for conflicts by enabling the Build Option "Warn on symbols with the same type and value". Or use the "All Symbols" view or Project Manager's "Data List view", sort by type/address and examine the assigned addresses.

- Dynamic addresses are renumbered using Project Manager's "Device > Advanced CleanFiles" command to delete the "Symbol information files".

- Addresses inside the dynamic address ranges defined in the Build Options should not be used.

- If you are unhappy with the result, use 'Undo' to restore all the addresses to their original values.
6 PROGRAMMING WITH FUPLA ......................................................... 3

6.1 Preparing a Fupla project................................................................. 4

6.2 Layout of the Fupla window ............................................................. 5

6.3 Editing connectors ........................................................................... 6
  6.3.1 Placing connectors................................................................. 6
  6.3.2 Editing a symbol inside connector........................................... 6
  6.3.3 Quick way to place a symbol and its connector ....................... 6
  6.3.4 Drag, Copy/Paste, Delete symbol .......................................... 6
  6.3.5 Copy/Paste, Delete connector................................................. 6
  6.3.6 Stretch connectors ............................................................... 7
  6.3.7 Move connector vertically ....................................................... 7

6.4 Placing a Fupla function box .......................................................... 8
  6.4.1 FBox selector...................................................................... 8
  6.4.2 Adding an FBox ................................................................. 9
  6.4.3 Edit stretchable FBox ........................................................ 9
  6.4.4 Edit logical inversion ......................................................... 9
  6.4.5 Triggering on a rising edge .................................................. 10
  6.4.6 Comments ....................................................................... 10
  6.4.7 FBox Help ....................................................................... 10

6.5 Links between FBoxes and connectors .......................................... 11
  6.5.1 Link by shifting FBox......................................................... 11
  6.5.2 Link with automatic routing ................................................. 11
  6.5.3 Multiple link with automatic routing ..................................... 11
  6.5.4 Link all inputs/outputs on an FBox to connectors .................. 11
  6.5.5 Delete lines, FBoxes, connectors or symbols ....................... 12
  6.5.6 Move FBox/connector vertically without undoing links ......... 12
  6.5.7 Insert FBox without undoing link ........................................ 12
  6.5.8 Rules to follow ................................................................ 12

6.6 Editing Fupla pages ......................................................................... 13
  6.6.1 Insert page ....................................................................... 13
  6.6.2 Delete a page ..................................................................... 13
  6.6.3 Page navigation ............................................................... 13
  6.6.4 Page documentation ....................................................... 14
  6.6.5 Processing of program by the PCD ..................................... 14

6.7 Copy and paste ............................................................................... 15
  6.7.1 Copy/paste part of a program .............................................. 15
  6.7.2 Copy and paste symbols .................................................... 15

6.8 Templates ....................................................................................... 16
  6.8.1 Creating a template............................................................. 16
  6.8.2 Importing templates .......................................................... 17

6.9 Editing your first Fupla program ..................................................... 19
  6.9.1 Objectives........................................................................ 19
  6.9.2 Solution.......................................................................... 19
  6.9.3 Programming ................................................................. 21
6.10 Building the program ................................................................. 22

6.11 Downloading the program into the PCD ...................................... 23

6.12 Finding and correcting errors (Debugging) .................................... 23
   6.12.2 Breakpoints ....................................................................................... 24
   6.12.3 Display symbols or addresses .............................................................. 25
   6.12.4 Display symbol state with Fupla .......................................................... 25
   6.12.5 Editing symbols online ......................................................................... 26
   6.12.6 Display/edit symbol state with Watch window ..................................... 26
   6.12.7 Setting the PCD's clock ....................................................................... 28

6.13 Adjust parameters .......................................................................... 29
   6.13.1 Initialization of HEAVAC FBoxes ......................................................... 30
   6.13.2 HEAVAC FBox with adjust parameters ................................................ 31
   6.13.3 Mini HEAVAC application .................................................................... 31
   6.13.4 Modifying Adjust Parameters when online .......................................... 32
   6.13.5 Restoring the original parameters from the Fupla file ....................... 32
   6.13.6 Saving the online parameters into the Fupla file .................................. 33
   6.13.7 Defining symbol names for Adjust Parameters ..................................... 34
   6.13.8 Defining Adjust Parameter addresses .................................................. 35

6.14 Commissioning an analogue module ............................................. 36
   6.14.1 Acquisition of an analogue measurements .......................................... 36
   6.14.2 Example for PCD2.W340 analogue input modules ............................. 37
   6.14.3 Example for PCD2.W610 analogue output modules ............................ 38
6 Programming with Fupla

The Saia PG5 Fupla Editor is the simplest, fastest introduction to programming Saia PCD controllers. The name "Fupla" means "FUnction PLAn", a graphical programming environment in which the user draws programs with the aid of hundreds of pre-programmed function boxes. These functions are organized into libraries covering the basic applications, through to more specialized functions for certain professional applications. The special libraries include: a HEAVAC library for heating, ventilation and air conditioning; a modem library for networking PLCs to exchange data via telephone lines (analog, ISDN, GSM, GPRS); SMS messaging; paging and DTMF. Other libraries for communications networks LON, EIB and Belimo products are also available.

The great advantage of Fupla lies in the fact that the user can put a PCD into service without having to write a single line of code, and without any particular programming knowledge.
6.1 Preparing a Fupla project

In the Saia PG5 Project Manager window, select the menu command Project, New... and create a new project.

Next, create a new Fupla program file in the project by clicking on the New File button on the toolbar, or click the right-hand mouse button on the Program Files branch and use the New File command from the context menu:

Select the file type Fupla File, and press OK to open the Fupla editor (S-Fup).
6.2 Layout of the Fupla window

The PCD reads the data from the input connectors, evaluates it according to the program and writes the results to the output connectors. The symbols used by the program are all shown in the Symbol Editor window. Most data types are allowed in the input and output connectors, except input and constant types which cannot be used for outputs. Digital inputs and constants provide read-only data, and can therefore only be used at the input connectors.

In the middle of the page we have the program, made up of different graphical function boxes selected from the FBox Selector window. The "wires" represent the exchange of data between the different functions. The colour of these wires defines the type of data: purple for binary (Boolean) information, blue for integers and yellow for floating-point numbers. Data which is different in type or colour cannot be linked together without first being converted to a common type. (FBox group: Conversion)

If the program uses several pages, the Page Navigator window allows pages to be deleted and helps you move around the program structure quickly.
6.3 Editing connectors

Input and output connectors can be placed anywhere on Fupla pages, and used to hold the necessary symbols for program functions described by FBoxes.

As a default, each new page may already provide margins with connectors on the left and right. If you prefer new pages not to appear with these connectors, so that you can place them yourself at your own convenience, please deactivate the corresponding option with menu: View, Options…, Workspace, New pages with side connectors.

To remove any connectors present on the left or right of the page, select menu: Page, Remove Unused Connectors.

To restore connectors to a blank page use: Page, Add Side Connectors.

6.3.1 Placing connectors

To add a connector and its symbol to a Fupla page, select the toolbar button Add Connector and position the mouse on the Fupla page. A ‘read’ input connector is added by clicking with the left-hand mouse button. A ‘write’ output connector is added by pressing the Shift key while clicking the left-hand mouse button. The connector you have just added is ready for entering a symbol, and a cursor is displayed inside the connector. If you don't want to edit the symbol inside the connector straight away, press the Esc key and place the next connector.

6.3.2 Editing a symbol inside connector

To edit or modify a connector symbol already present on the Fupla page, click on the connector once to select it, and a second time to open the field for editing. A cursor is displayed inside the connector, and it's now possible to enter the symbol definition. Note that newly entered symbols are automatically added to the Symbol Editor window if they are not already there.

6.3.3 Quick way to place a symbol and its connector

Symbols already present in the Symbol Editor window can be dragged onto a free space on the Fupla page. This will create a new connector containing the symbol. If the symbol is dragged onto an FBox input or output, an input or output connector will be created and linked directly to the FBox.

6.3.4 Drag, Copy/Paste, Delete symbol

Selecting the area shown in red will only affect the symbol. It is possible to select the symbol with the mouse and drag, copy/paste it to another connector, or delete it. The right-hand mouse button will display a context menu with all available operations.

6.3.5 Copy/Paste, Delete connector

Selecting the area shown in white affects the connector and the symbol it contains. The right-hand mouse button will display a context menu with all available operations.
6.3.6 **Stretch connectors**

Connectors are stretchable. This means that the number of connectors can be defined by vertical movement of the mouse.

Press toolbar button: *Select Mode*
Select connector on area shown in red.
Display context menu by right-clicking mouse.
Selection menu: *Stretch*
Move the mouse vertically to create the
Desired number of connectors
Press the left-hand mouse button.

6.3.7 **Move connector vertically**

To move the connector, place the mouse in the red circle.
Press and hold down the *shift*-key.
Press and hold down the left-hand mouse button.
Drag the mouse vertically onto a free space on the page.
Release mouse button and *shift*-key.
6.4 Placing a Fupla function box

6.4.1 FBox selector

The FBox Selector window shows all FBoxes defined in the available FBox libraries. They are organized into Families with similar application domains. For example, here are some of the main families:

<table>
<thead>
<tr>
<th>Family</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>FBoxes for solving logical equations</td>
</tr>
<tr>
<td>Integer</td>
<td>Arithmetic with integer numbers</td>
</tr>
<tr>
<td>Floating Point</td>
<td>Floating point arithmetic</td>
</tr>
<tr>
<td>Counter</td>
<td>Counting tasks</td>
</tr>
<tr>
<td>Timer</td>
<td>Time-related tasks</td>
</tr>
<tr>
<td>Analog I/O</td>
<td>Controlling analogue modules</td>
</tr>
<tr>
<td>Communication S-Bus</td>
<td>Exchanging data on S-Bus or Ethernet networks</td>
</tr>
<tr>
<td>Conversion</td>
<td>Converting binary to integer, integer to floating point, etc.</td>
</tr>
</tbody>
</table>

The FBox libraries provide almost all the operations ever needed by a program. There are a many Families and FBoxes, and it may be difficult to find the right one. Various search facilities have been provided to help you.

If an FBox family is selected, pressing a letter key scrolls to the next family name which begins with that letter. If a family branch is open, pressing a letter key scrolls to the next FBox name in that family which begins with that letter.

The Selector window’s toolbar has a Filter field where a filter string can be entered. For example, type ADD and press the Enter key, the Selector window will now show only the FBoxes which contain the ADD keyword, which is Floating Point and Integer. To see all the FBoxes again, press the Clear Filter button.
Project Manager's Libraries branch shows all the PG5's installed libraries, and libraries local to the current Project. Libraries which you don't want to use can be unchecked, which reduces the number of libraries shown in the Selector window.

FBoxes which are regularly used can be added to a Favorites list. Select the FBox, open the context menu and use the Add To Favorites command.

To show the Favorites, press the button at the bottom of the Selector window.

### 6.4.2 Adding an FBox

The functions needed for writing a program are selected from the FBox Selector, then inserted into the Fupla program using Drag&Drop.

### 6.4.3 Edit stretchable FBox

Some FBoxes are stretchable, which means that the number of links can be defined by dragging vertically with the mouse.

Select a stretchable FBox.
Drag&Drop onto the Fupla page.
Drag the mouse vertically until the correct number of inputs or outputs is shown.
Press the left mouse button to finish.

### 6.4.4 Edit logical inversion

Click on the Invert Connector button.
Position the mouse pointer on the input or output link of a logic function and press the left mouse button.
6.4.5 Triggering on a rising edge

The inputs of some binary FBoxes have been 'dynamized'. They only trigger on the rising edge of the signal. These are indicated by a little black triangle.

For example, a pulse counter cannot be incremented when its Up input is one.

Otherwise, what would happen if the Up signal remained at one for any amount of time? The counter would be continuously incrementing itself for as long as the Up signal remained one. It is for this type of application that certain digital inputs have been dynamized. Therefore, only the positive edge of a Up signal will increment the counter.

It is sometimes necessary to add dynamization to the input or output of an FBox. We then use the Binary, Arithmetic Dynamize function.

6.4.6 Comments

Comments can be inserted anywhere on the page:

1. Click on the Place comment button
2. Position the comment field on the Fupla page, then press the left mouse button.
3. Write the comment.
4. Press the Enter key.

Note:
To write a comment with more than one line use Ctrl + Enter keys to create the next line.

6.4.7 FBox Help

To obtain a full description of any function, select the FBox in the Selector window or on the Fupla page and then press the F1 key.

For rapid identification of an unknown FBox found in a program, open the Selector window, position the mouse pointer on the unknown FBox and single-click on the left mouse button. The Selector window will select the FBox.
6.5 Links between FBoxes and connectors

6.5.1 Link by shifting FBox
1. Click on the Select Mode button on the toolbar.
2. Point onto the FBox, then press the left mouse button.
3. Without releasing the button, drag the FBox to the adjacent FBox. Release the mouse button once the link is indicated with a yellow point.
4. Drag the FBox again to position it where you want it.

6.5.2 Link with automatic routing
1. Click on the Draw Lines button on the toolbar.
2. Position the mouse pointer at the depart, a yellow point is displayed when the link is detected, click on the left mouse button.
3. Position mouse pointer on destination point, a yellow point is displayed when the link is detected, click the left-hand mouse button.

Note:
Intermediate points of passage can also be selected.
To interrupt link editing, press the right-hand mouse button.

6.5.3 Multiple link with automatic routing
1. Select menu Mode, Connect Bus or (CTRL+B).
2. Select a starting point with the mouse.
3. Then select the destination point

6.5.4 Link all inputs/outputs on an FBox to connectors
Place the mouse pointer over an FBox. Right-click to display the context menu: Connections, Connect to Side Connectors.
6.5.5 Delete lines, FBoxes, connectors or symbols

Select the *Delete Object* button on the toolbar, then select the links, FBoxes or symbols to delete.

Another, faster solution is to mark a space and delete it.

1. Press the mouse button.
2. Without releasing the button, drag the mouse.
3. Release the mouse button.
4. Select menu *Edit, Delete*.

6.5.6 Move FBox/connector vertically without undoing links

Position the mouse pointer over the FBox. Press and hold down the left-hand mouse button. Drag the mouse vertically onto a free area on the page. Release mouse button.

To move the connector, position the mouse pointer in the red circle and repeat the sequence.

6.5.7 Insert FBox without undoing link

Select the FBox to insert in the window *Selector*. Place it above the link.

6.5.8 Rules to follow

Loops are not allowed.

No direct links are allowed between input connectors and outputs connectors. An FBox must be used: *Binary, Direct transfer* or *Integer, Direct transfer*. 
6.6 Editing Fupla pages

The Page Navigator window shows the program's blocks and pages. Each Fupla file can hold up to 200 pages grouped into blocks: COBs, PBs, FBs, or SBs. But Fupla is faster if you don't have too many pages in a single file. By default, pages are put into a COB type block. For more detailed information about blocks and their use, please refer to the Program Structure section in this document.

6.6.1 Insert page

Open the Page Navigator window, mark the reference page and select Insert Page from the context menu.

It is also possible to insert a page after the current page with the Insert Page button or the menu item: Page Insert After (Page Insert Before)

6.6.2 Delete a page

Open the Page Navigator window, mark the page to be deleted and select Delete from the context menu.

6.6.3 Page navigation

It is also possible to navigate with the Go to Previous Page and Go to Next Page buttons, allowing movement from page to page in a Fupla block. If either of the buttons is grey, you are already on the first or last page of the block.
6.6.4 Page documentation

You are strongly advised to document each of your Fupla pages. This is very useful when navigating through the pages of your program, because page names and comments will be displayed in the Page Navigator window. The description is a way of leaving some useful information about the program that will make it easier to maintain.

To display the Properties page, select the page in the Page Navigator, open the context menu and use the Properties command.

6.6.5 Processing of program by the PCD

The PCD processes the pages of each block from the top left of the first page to the bottom right of the last page. For more precise details on the order in which FBoxes are processed by the PCD, select menu path: Page, Show FBox priorities
6.7 Copy and paste

Certain parts of a program may be repetitive. It is not necessary to edit them again in full. It is much faster to duplicate them by copying and pasting, and then adapt them as required.

6.7.1 Copy/paste part of a program

1. Click on the Select Mode button.
2. Mark the area to be copied:
   - Press the left mouse button.
   - With button still pressed, slide mouse.
   - Release left mouse button.
3. Add an FBox or connection to the selection:
   - Press the Ctrl key.
   - Keeping the Ctrl key down, select the connectors and FBoxes to add.
4. Copy the selection with the Edit Copy menu path, or with the Ctrl+C keys.
5. Paste a copy of the selection with the Edit Paste menu path, or the Ctrl+V keys.
6. Position the copy on the Fupla page:
   - Position mouse pointer in middle of copy.
   - Press left mouse button.
   - With button still pressed, slide mouse.

6.7.2 Copy and paste symbols

1. Click on the Select Mode button.
2. Mark a list of symbols:
   - Position mouse pointer on first symbol.
   - Left-click with mouse.
   - Position mouse pointer on last symbol.
   - Press Shift key. *)
   - Keeping Shift key down, left-click with mouse.
3. Copy the selection with the Edit Copy menu path, or with the Ctrl+C keys.
4. Position the mouse pointer on a free part of the margin.
5. Paste a copy of the selection using the Edit Paste menu path or the Ctrl+V keys..

*) The Ctrl key allows non-consecutive symbols to be selected.
6.8 Templates

Fupla pages can be saved as a Template which can be used like a library of pages.

6.8.1 Creating a template

It is easy to create a template. Use the Page Navigator window to select one or more pages, the execute the context menu command Add To Templates. A dialog box will ask for a group, a name and a comment for the template.

The template's Group is similar to the FBox Family. The group organizes the templates according to the classification defined by the author. The group name defines the tree structure of the Templates selector window.

The template comment and group are changed in the Properties window: Select the group or template, then select the Properties context menu.
6.8.2 Importing templates

Templates can be re-used in any projects. Select the template in the Template Selector window and drag it onto a Fupla page to insert the sequence of pages with their FBoxes, links, symbols, adjust parameters etc., into the file.

A dialog box is displayed which allows the changing of names and addresses of imported symbols and other parameters. This functionality is similar to a macro or function with parameters which can be used like a library.

![Import dialog box]

**Number of Copies, Base Index**
If several copies of the same template are needed, define the number of times the template will be inserted and the base index which will be appended to the symbol and group names.

**Before/After current page**
Imports the sequence before or after the page currently active in Fupla.

**Set Internal Variables to system defined**
Some FBoxes have Adjust Variables whose name is defined by the user or by the system (Static symbols). This setting allows you to keep the addresses and static symbols defined by the author, or to restore dynamic addressing and internal symbols by default.

**Reset page conditions**
A Fupla page’s Properties window allows the definition of an execution Condition for each page. This setting allows the conditions to be removed when the pages are imported.

**Page Range**
Allows the selection of individual pages to be imported, or all pages in the template.
The Symbol List page shows all the symbol definitions and references which are imported with the template. It allows the symbols to be redefined with new names, addresses, comments and scopes.

Marking symbols for putting in a group is the fastest way to change the names of all the imported symbols. The context menu command Insert Pre-group puts the selected symbols into a group with the chosen name.

To change the addresses of symbols, sort them by type by pressing the column's header button Type. Select and edit the first address, and then drag the tiny square at the bottom right of the selected cell downwards until all the desired addresses are selected and release the mouse button. All selected addresses are renumbered starting from the first address.

If importing several copies of the same template, examine the parameters on the General page. It is useful to insert an index into the names of symbols or groups using the # character. This character is automatically replaced by the base index incremented by 1 for each copy of the template. It is also possible to select the symbols using the context menu command Indexing.

The FBox List page shows a list of all the FBoxes whose symbols are defined with a name. These names can also be modified using the # character.
6.9 Editing your first Fupla program

6.9.1 Objectives

Now that the working environment is known, the next step is to create a more complex program than the logic structures presented up to this point. We propose creating a daily timer to control a digital output (O 32) that comes on at 06.00 hrs and goes off at 19.00 hrs. Although this function is available with the HEAVAC library, we are going to reproduce it ourselves using standard FBoxes.

6.9.2 Solution

Before starting to program, a method must be found that will behave according to our specification document and that can be implemented with the most elementary functions possible.

For this timer example, we propose making two comparisons. The first will determine whether the current time in HMS (i.e. the time by our watches or PCD time) is greater than or equal to the turn-on time: ONTIME. The second will determine whether current time is smaller than or equal to the turn-off time: OFFTIME. If both comparisons are verified by an expression – an exclusive OR logic function – the timer's digital output 32 DailyTimer must be switched on.

This algorithm offers one solution, but it may leave some gaps. What happens if the turn-on and turn-off time instructions overlap? The following drawing demonstrates that the PCD output will be in the opposite state to that desired.
It is therefore necessary to complete our algorithm by adding a third comparison to determine whether the turn-on time is greater than or equal to the turn-off time. The final solution is therefore as follows.

Outputs active by day:

Outputs active by night:
6.9.3 Programming

It is now time to move on to programming. At the beginning of this chapter we created a project with a file in it called: `DailyTimer.fup`. This is the file to which you will now write the present programming example.

Start by creating the symbol list. Note that the current PCD time is saved in a dynamic HMS register. The address of this register has not been defined. The PG5 will automatically assign its address when the program is built. The same applies for the turn-on and turn-off times (`ONTIME, OFFTIME`), except that «:=60000» is not a register address, but the value with which it will be initialised when the program is downloaded to the PCD (:=60000 means 6 hours 00 minutes 00 seconds).

N.B.: A PCD cold start will not reinitialise these registers. They can only be reinitialised by downloading the program!

All the necessary FBoxes can be found in the Selector window:
- System information, Read time
- Integer, Comparison, Is greater or equal to
- Binary, Arithmetic, Xor
### 6.10 Building the program

Before the fully edited program can be read and executed by the PCD, it must be "built" (or converted) using the command **Device, Build Changed Files** or **Rebuild All Files** from in the Project Manager or the Fupla editor.

The **Messages** window shows the results of various stages of the program build (Compiling, Assembling, Linking etc.). If the program has been correctly edited, the build function concludes with the message:

*Build successful. Total errors: 0 Total warnings: 0*

Any errors are indicated with an error message in red. By double-clicking the mouse button, the error can easily be located in the user program.
6.11 Downloading the program into the PCD

The user program is now ready. All that remains is to download it from the PC into the PCD. This is done with Project Manager's Download Program toolbar button or with menu command Online, Download Program.

If any communications problems arise, check the Online Settings, Device Configurator and the cable connection between PC and PCD. Make sure the PCD is powered up.

6.12 Finding and correcting errors (Debugging)

The first version of a program is not always perfect. A stringent test is always needed. The program can be tested using the same editor that was used to write the program.

6.12.1 Go On/Offline – Run – Stop - Step-by-step

1. Press the Go On /Offline button
2. Start program with the Run button

At the same time, observe the Run lamp on the front-panel of the PCD. The Run command turns on the Run LED and PCD starts executing the program.

3. When the Stop button is clicked, the Run lamp goes off and the PCD stops execution of the user program.
4. The PCD executes one FBox each time the Step-by-step button or F11 key is pressed.

Observe the Stop marker which indicates the step-by-step progress of the program.
6.12.2 Breakpoints

Breakpoints let you stop a program at an event linked to one of its FBoxes, or to a symbol:
- Low or high state of input, output, flag or status flag
- Value in a register or counter

**Breakpoint on a symbol's value**

The stop condition can be defined via the menu *Online Breakpoints*.

The above window is used to define symbol type and address/number. A symbol can simply be dragged from the Symbol Editor into the *Symbol Name* field, then the breakpoint condition and status/value are defined.

Pressing the *Set & Run* button will put the PCD into Conditional Run mode. The PCD's *Run* LED will flash and its *Run* toolbar button alternates between green and red.

The PCD will automatically put itself in stop mode when the breakpoint condition is reached. For example, when an instruction modifies the output value, the status of 32 is high. The last FBox processed by the PCD is shown with a red *Stop*. It is possible to continue processing the program in step-by-step mode, or with another breakpoint condition.

If necessary, conditional Run mode can be interrupted:
- The *Clear-Run* button forces the PCD into RUN mode. The PCD's *Run* LED will come on and its *Run* button will be green.
- The *Clear-Stop* button forces the PCD into Stop mode. The PCD's *Run* LED will go out and its *Run* button will be red.

If a number of conditional breakpoints are defined, they will all be recorded in the *History* field. Any of them can then be selected with the mouse and activated with the *Set & Run* button.

**Breakpoint on a program FBox**

Select any FBox within the program, followed by the menu or button *Online, Run to, Fbox*, to make the program stop at the chosen FBox, and then continue in step-by-step mode.
6.12.3 Display symbols or addresses

The Change Symbol/Resource view button allows information from the connectors to be displayed with their symbols or addresses. If pressing it does not replace a symbol with its corresponding address, that symbol's address is assigned by the build. If the connector always displays the address, it means that this address is not linked to a symbol.

It is also possible to display symbols via the Symbol Editor. Selecting a connector will highlight the symbol present in the Symbol Editor view.

6.12.4 Display symbol state with Fupla

When the editor is Online and the PCD is in Run mode, each individual symbol used by the program can be displayed:

The logical state of a binary value is shown with a heavy or fine line (heavy = 1 and fine = 0). Other data values can be seen by pressing the Add Probe button to connect a probe to a line.

Double-clicking on a probe opens the Probe Display Format window, allowing a choice of format for values consulted: integer, hexadecimal, binary, floating point, boolean or ASCII.
6.12.5 Editing symbols online

When checking program behaviour under certain conditions of use, it is sometimes helpful to change the states/values of symbols present in the input connectors.

Select an input connector with the mouse and right-click to display the context menu.

The *Edit Data* context menu lets you modify the state/value of a symbol inside a connector or the *Symbol Editor*.

6.12.6 Display/edit symbol state with *Watch window*

Another useful way to test and display the symbol states in our example is in the *Watch Window*. Press the *Watch Window* button. Then drag one or several symbols from the symbol editor to the *Watch Window* or use the *Add to Watch Window* context menu.

To select and drag several symbols at the same time, click on the first symbol in the Symbol Editor, press the *Shift* key, then click on the last symbol; the *Ctrl* key can be used to individually add additional symbols.

Press the *Start/Stop Monitoring* button to display the values presented in the PCD in the *Value* column, for each of the symbols shown in the *Watch Window*. Check that the status bar of the *Watch Window* indicates *RUN* mode. If necessary, force the PCD into *RUN* or *STOP* mode using the *Online* menu.
To check that our daily timer example is functioning properly, we are going to change the switch on and off settings \((ONTIME\) and \(OFFTIME\)) and look at the status of the \(DailyTimer\) output.

The \textit{Modify Value} column is used to define new values for several symbols and simultaneously download them to the PCD by pressing the \textit{Download Values} button.

If necessary, change the display format by clicking on a symbol in the \textit{Watch Window} and opening the \textit{Display Format} context menu.

It is also possible to select a symbol from the table and change its status/value using the toolbar.

To change the value of a register, edit the value in the field, then click the button to its right.

To change a binary status, click directly on the \textit{Reset value}, \textit{Set binary value}, \textit{Toggle binary value} buttons.

On exiting the \textit{Watch Window}, a dialog box asks if you want to save the symbols in the view with a name defined by default. If the user accepts, the symbols are kept for the next time the \textit{Watch Window} is opened.

It is also possible to force the view to be saved using the \textit{File, Save} menu or pressing the \textit{Save} button.

It is also possible to save different symbol lists and display them simultaneously. Use the \textit{Files, Save as..} menu to save the views and \textit{File, Open} to display them.
6.12.7 Setting the PCD's clock

When a PCD is commissioned, its internal clock may not have the correct time. To adjust it, proceed as follows:

Press the **Online Configurator** button on the **Project Manager** window. Then press **Clock**.

Copy the PC's time to the PCD with the **Copy to PCD >>>** button, or enter clock settings in the **PCD Clock** fields and press **OK**.

Comment:
It is also possible to automatically synchronise the PCD clock with the Ethernet network. This functionality requires the Ethernet port and SNTP server to be configured using the **Device Configurator**.
6.13 Adjust parameters

Some FBoxes, which are indicated by a black triangle in the bottom left-hand corner, have additional Adjust Parameters. These parameters configure particular features of the FBox, and they can also be modified online.

If the Properties window is already open, just click on the FBox to see its properties, which shows the Adjust Parameters. If not open, right-click on the FBox to display the context menu, and select the Properties command.

Editing Adjust Parameters when offline

<table>
<thead>
<tr>
<th>Adjust Variables</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objet pour éditeur HMI</td>
<td>Non</td>
</tr>
<tr>
<td>Enclenchement</td>
<td>12:00</td>
</tr>
<tr>
<td>Déclochement</td>
<td>13:00</td>
</tr>
</tbody>
</table>

The offline editing of adjust parameters is supported by the Properties window. The parameter's values are saved in the Fupla file. The program must be built and downloaded for the new parameter values to be used by the PCD.

Editing Adjust parameters when online

The online editing of adjust parameters is supported by the View, Adjust Window command which opens the online Adjust Window showing both the actual online values and the modified values. The modified values are written directly into the PCD's memory and are not updated in the original Fupla source file.

The online Adjust Window is automatically displayed instead of the Properties window when Fupla is online.

FBox: HVC, clocks, daily clock
6.13.1 Initialization of HEAVAC FBoxes

When using certain FBox libraries, such as the HEAVAC library, an initialization FBox must always be positioned on the first page of the Fupla file. It allows some of the library’s common tasks to be managed, such as initialisation of the library after the program has been downloaded or after a PCD cold start (power-up).

After any program download and PCD cold start, the Res input of this FBox and the adjust parameters shown below have an important influence on initialisation of the adjust parameters for all the other HEAVAC FBoxes in the program.

<table>
<thead>
<tr>
<th>Reset</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Automatic Reset  …</td>
<td>Activated</td>
</tr>
<tr>
<td></td>
<td>Evaluate Reset Input</td>
<td>At startup</td>
</tr>
</tbody>
</table>

Downloading the program and the automatic Reset parameter

With the Active option, the adjust parameters of all HEAVAC FBoxes will be initialised with the values defined by the program.

With the Not active option, all existing parameters in the PCD will be preserved.

Res input and the Evaluate Reset input parameter

If the status of the reset input is high, the adjust parameters of all HEAVAC FBoxes will be initialised with values defined during programming.

Depending on the option selected for the Evaluate Reset input parameter, the Res input will only be taken into account in case of a PCD coldstart or during runtime (always).

Green/red LED

Some FBoxes have a simulated LED that can display three different colours: grey when the controller is off-line, green or red when the controller is on-line. Green signifies that everything is functioning properly, red indicates an error (generally caused by information at FBox inputs or by the selection of unsuitable adjust parameters. For more detailed information, please consult the guides regarding FBox errors).

Note:

Within the HEAVAC library you will find different versions of the initialisation function (Initialisation HEAVAC 4, …7). Version 7 is the most recent. We recommend the use of function 6 for all new applications.
6.13.2 HEAVAC FBox with adjust parameters

The `Clk_D` FBox allows a daily clock to be produced just like the one created earlier in this chapter, but with a single FBox available in the HEAVAC library. The FBox output can be switched on or off according to times defined in the adjust window.

The parameter *Objet pour HMI editor* is only used in the presence of HMI terminals. If this option is not used, keep the proposed standard parameter. Input `En` allows the clock function to be disabled. If `En` is low, output `Ch` will remain inactive.

6.13.3 Mini HEAVAC application

To try out the operation of adjust window parameters we can once again use the daily clock program presented at the beginning of this chapter. However, this time we will achieve it with the help of the HEAVAC library.

The two FBoxes described above are the only ones we need. Create the program as set out below, then execute *Build All, Download Program and Go Online*.

If the program is extended with several other HEAVAC FBoxes, the *Initialisation* HEAVAC 7 FBox must be positioned once only at the top of the first Fupla page.
6.13.4 Modifying Adjust Parameters when online

The online modification of Adjust Parameters is supported by the View, Adjust Parameters window. It shows the parameters of the selected FBox in a window which works a bit like the Watch Window.

The Description column describes the adjust parameter. The Online Value column shows the value of the parameter read from the memory of the PCD. The Modify Value column allows new values to be entered and written individually or simultaneously into the PCD.

- Writes a single parameter to the PCD.
- Writes all changed parameters simultaneously.

It's also possible to select a parameter and modify it in the Edit Data field on the tool bar.

The values of modified parameters are written directly into the PCD's memory, they do not change the contents of the original Fupla file.

6.13.5 Restoring the original parameters from the Fupla file

After online changes to the adjust parameters, it's possible to restore the original values from the Fupla file. The Show Source Value button fills the Source Value column with the original values from the Fupla file. See the FBox Properties window.

- Restores a single parameter
- Restores all the parameters for the current FBox from the Fupla file

You can also use the menu command Online, Write FBox Adjust Parameters download Adjust Parameters from the Fupla file.
6.13.6  Saving the online parameters into the Fupla file

If the parameters which have been changed online are suitable, they can also be saved into the Fupla file.

Save a single parameter

Save all the parameters the current FBox

You can also use the menu command Online, Read FBox Adjust Parameters to upload Adjust Parameters from the PCD and save them in the Fupla file.
### 6.13.7 Defining symbol names for Adjust Parameters

Sometimes it is necessary to read or write adjust window parameters from the Fupla program, the communications network, or the supervisory system.

This is possible if symbols have been defined for flags or registers corresponding to parameters displayed in the FBox adjust window.

To define these symbols, right-click on the FBox to display the context menu. Select menu item *FBox Properties* → Define a symbol name for a group of parameters linked to the selected FBox.

To define these symbols, open the FBox's *Properties* window and fill in the *Name* in the *General* section.

Build the program and open the Symbol Editor. Open the *System* symbols page.

With the HEAVAC library, all system symbols corresponding to adjust window parameters are grouped under `A.HVC.name` (where *name* is the symbol name of the FBox).
6.13.8 Defining Adjust Parameter addresses

Define the system symbol for the Adjust Parameters as described above, and add the address from the FBox Properties. Select the System Defined line and press the button at the end of the line.

Build the program and open the Symbol Editor. System symbols have been assigned the register addresses shown below.
6.14 Commissioning an analogue module

The reading or writing of an analogue value requires a small program for each analogue module. This controls the multiplexing of channels and the A/D or D/A conversion. This program is provided either by an FBox or by the media mapping created by the Device Configurator.

6.14.1 Acquisition of an analogue measurements

The sample programs presented up until now make use of digital inputs and outputs, putting their addresses or symbols in the connectors of the FUPLA page.

With analogue input or output modules, an FBox must be used to acquire the analogue value. These FBoxes are available with libraries: Analogue modules, or HEAVAC-Analogue.

These libraries offer a wide variety of FBoxes, each corresponding to an analogue module. The name that appears in the FBox Selector matches the module item number.

Analogue FBoxes are expandable. The user can define the number of measurement channels required by an application. If some measurement channels are not used, or if an extra channel is added, the context menu Resize FBox can be used to adjust its dimensions. However, an FBox can also be defined with the maximum number of channels, even if they are not all used.

The Add field allows the base address of the analogue module to be defined. This address indicates where the module has been inserted in the PCD: 0, 16, 32, ...

Analogue measurements are available at FBox inputs I 0 to I 7. They can be connected directly to other FBoxes, or the values can be saved to a register. Saving a value to a register is a good solution, particularly when the value is used on several different pages of the program or by Graftec Steps and Transitions.

Attention:

Be careful never to define more than one FBox per analogue module, and never to insert the analogue module at the PCD watchdog address (255). Otherwise the value supplied by the module may be incorrect.
6.14.2 Example for PCD2.W340 analogue input modules

If the PCD is equipped with a PCD2.W340 module, which has 8 universal input channels, the user can take one of the following FUPLA FBoxes and define the required number of measurement channels.

FBoxes: Analog I/O, PCD2/3, PCD2.W3, PCD2.W34, PCD2.W34 with error

Units of measurement depend on the module, FBox, and adjust parameters selected.

The PCD2.W340 is a universal module. It supports measurement of ranges 0..10V, 0..2.5V, 0..20mA and Pt/Ni 1000 temperature sensors. A bridge must be selected on the module to define the measurement range. Resolution is 12 bits, equating to 4096 distinct measured states. (For more detailed information about these modules, please refer to your PCD hardware manual.)

The PCD2.W3 FBox supplies a raw measurement. For this module with a resolution of 12 bits, that corresponds to a measured value between 0 and 4095. The user then has the task of converting the measurement into a standard physical unit.

The PCD2.W34 FBox is more elaborate. An adjust window allows units of measurement to be defined for each channel. The FBox LED turns red if one of the measurements exceeds the valid range: short-circuit or break in sensor cable. The error can be acknowledged with the Acknowledge button in the adjust window.

The PCD2.W34 with error FBox offers the same services for converting units, but also has an error output indicating which channel has the error, plus an additional adjust parameter to define a default value in case of error.
6.14.3 Example for PCD2.W610 analogue output modules

The same principle applies as for inputs: the user puts an FBox corresponding to the analogue output module on the FUPLA page, drags it to select the number of output channels and defines the module base address.

Unlike input FBoxes, the setpoints of analogue outputs are displayed on the left side of the FBox.

These inputs can be linked directly to other FBoxes, or to registers defined in the connectors of the FUPLA page.

If the PCD is equipped with a PCD2.W610 module, which has 4 universal analogue outputs, the FBox below may be used to output a current of 0..20mA, or a voltage of 0..10V.

<table>
<thead>
<tr>
<th>Input value at FBox</th>
<th>Output voltage [V]</th>
<th>Output current [mA]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2047</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>4095</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

Other FBoxes have an adjust window for adapting the range of setpoint values applied to the FBox input (e.g. the FBox for the PCD2.W605 module, which has 6 electrically isolated outputs of 0..10V:

The parameters User scaling 0 and 100% allow values to be defined for the minimum and maximum channel voltages applied to the FBox input. The Reset value parameter corresponds to the value applied to the channel when the PCD is powered up.
7 PROGRAM STRUCTURE........................................................................................................... 3

7.1 Cyclic Organisation Blocks (COB 0 to 31)............................................................................. 4
  7.1.1 Definition.......................................................................................................................... 4
  7.1.2 Creating a block.................................................................................................................. 5
  7.1.3 Example.................................................................................................................................. 5

7.2 Program Blocks (PB) and Function Blocks (FB)..................................................................... 6
  7.2.1 Definition.......................................................................................................................... 6
  7.2.2 Program Block with conditional call.................................................................................... 7
  7.2.3 Function Block with parameters............................................................................................ 9

7.3 View Block Call Structure......................................................................................................... 10

7.4 Exception Organization Blocks (XOB).................................................................................. 11
  7.4.1 Definition.......................................................................................................................... 11
  7.4.2 XOB numbers and descriptions.......................................................................................... 12
  7.4.3 Use of XOBs....................................................................................................................... 13
  7.4.4 History.................................................................................................................................. 16
  7.4.5 Description of XOBs............................................................................................................ 17

7.5 Sequential Blocks (SB 0 to 96)............................................................................................. 21

7.6 Summary..................................................................................................................................... 21
7 Program structure

A good program needs a well-designed structure. It simplifies the program and makes it easier to develop and maintain. The Saia PCD programming language is a language which uses "organisation blocks" to structure the program. Each block encapsulates a task, program or function. These are the block types: Cyclic Organisation Blocks (COBs), Program Blocks (PB), Function Blocks (FB), Exception Organisation Blocks (XOB) and Sequential Blocks (SB).
7.1 **Cyclic Organisation Blocks** (COB 0 to 31)

7.1.1 **Definition**

Cyclic Organisation Blocks (COBs) are the program's "tasks", which run continuously, without program loops or waiting for events. When the PCD starts up, it executes the instructions in each COB in turn in a continuous loop. A program must have at least one COB. Because each COB is executed cyclically, it can regularly check for significant events, such as input signals, end-stop switches, emergency stop buttons, etc.

It is important to understand the concept of COBs. Since all COBs must run continuously, they should not contain wait loops or delays because these would prevent the regular handling of events.

If you use the Fupla editor (S-Fup), it creates a COB by default. Fupla programs are "continuous function chart" programs, which are executed cyclically and are therefore well-suited to COBs.

If the program is written in the Instruction List language, the block begins with a COB instruction and ends with ECOB (End COB). The block's code is written between these two instructions. At the start of every COB, the Accumulator (ACCU) is always High (1), you will see that this is very important for cyclic programs.

The COB instruction has two operands. The first is the COB number, the second is the COB's **supervision time**. If the supervision time is 0, then the COB's execution time is not monitored. If the supervision time is not 0, then it represents a timeout in 10ths of a second (e.g. 10 = 100ms, 100 = 1s). If the COB has not finished within that time, the Exception Organization Block XOB 11 is called. At the end of XOB 11, the COB which timed out is resumed from where it was interrupted, and the supervision time is restarted. The Error LED is not lit because the error has been handled by the program.

If XOB 11 is not programmed, the PCD's Error LED is lit, and execution continues with the next COB, which starts its own supervision time. On the next cycle of the program, the COB which timed out continued from where is was interrupted, and the supervision time is restarted.

In Fupla programs, the supervision time is configured using the **Block, Properties** command.

**Note:**
Each COB has its own **Index Register**.
### 7.1.2 Creating a block

A Fupla file can contain several blocks which can be added, deleted or edited using the **Block** menu.

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol name of the block</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Block's type: COB, PB, FB, XOB</td>
</tr>
<tr>
<td>Comment</td>
<td>Free comment text</td>
</tr>
<tr>
<td>Number</td>
<td>The block's number. For example, COBs are numbered 0..31, PBs are 0..999. By default the block's number is empty (dynamically allocated), except for XOBs. If the block's number is dynamically allocated, then the actual block number is assigned by the build.</td>
</tr>
<tr>
<td>Scope</td>
<td>Scope of the symbol (Local or Global). Global means that the symbol is accessible for other files.</td>
</tr>
<tr>
<td>COB Supervision Time</td>
<td>COB timeout period, in 100ths of a second.</td>
</tr>
</tbody>
</table>

### 7.1.3 Example

This example, in both IL and FUPLA, makes Output 64 blink at a rate of 1.5 seconds. The program is written in COB 0. Other COBs, if present, are executed consecutively.

**IL program**

```plaintext
COB 0 ;start COB 0
    0 ; supervision time=0
STL T 1 ; if Timer 1 = 0,
LD T 1 ; load it with 1.5s 15
COM O 64 ; and toggle O 64
ECOB ; end of COB 0

COB 1 ; next COB 0
...
ECOB
```

**Fupla program**

START UP

FBox: Timer, Pulse, Blink delay T
7.2 Program Blocks (PB) and Function Blocks (FB)

7.2.1 Definition

The programming language also allows working with Program Blocks (PB 0..999), and Function Blocks (FB 0..1999). These provide a good way to organize the structure and hierarchy of the program.

The only difference between PBs and FBs is that an FB can be called with parameters, and a PB cannot.

FBs provide an ideal solution for developing libraries which are usable in many projects. They help to reduce the development time.

PBs and FBs must be called from other blocks (COB, PB, FB, SB or XOB).

There are two type of call, conditional and unconditional. The first depends on the result of a logical operation; the second does not depend on a condition. It is possible to call the same PBs and FBs several times in a program.

A PB/ FB can also call another PB/ FB, and another, up to a maximum call depth of 31. If the maximum call depth is exceeded, then Exception Organization Block XOB 10 is called, if programmed.
7.2.2 Program Block with conditional call

Example of a two-speed blinker controlled by the state of input *Condition*.

**Fupla program**

If the state of digital input *Condition* is low (0), then PB *ShortTime* is called, which loads timer *WaitTime* with constant *ShortTimeValue* (5). If input *Condition* is high (1), then PB *LongTime* is called, which loads timer *WaitTime* with the constant *LongTimeValue* (15). The *WaitTime* timer defines the length of the delay between the changes of the blinker state (FBox "Blink"). To ensure that the *WaitTime* timer is initialised after a cold start, the *Blink* FBox must be positioned after both the PB calls.

If the program is edited with the Fupla editor, create a new block with the menu command *Block, New* and enter the block name from the block's *Properties* window. The *Call PB* FBoxes can be found in the *Block Call* family in the FBox *Selector* window.
IL program

; Two-speed Blinker
LongTime EQU PB 1
ShortTime EQU PB 2
ShortTimeValue EQU K 5 ; 0,5s
LongTimeValue EQU K 15 ; 1,5s
Condition EQU I 1
Output EQU O 32
WaitTime EQU T

COB 0
0
STH Condition ; IF Condition = High
CPB L ShortTime ; THEN Call PB ShortTime
CPB H LongTime ; ELSE Call PB LongTime
ECOB

PB ShortTime
STL WaitTime ; IF WaitTime = Low
LD WaitTime ; load it with a short value
ShortTimeValue
COM Output ; Invert the output
EPB

PB LongTime
STL WaitTime ; IF WaitTime = Low
LD WaitTime ; load it with a long value
LongTimeValue
COM Output ; Invert the output
EPB

If the program is written in Instruction List, the start of a block is begins with the instruction PB or FB. with the block name or number as the operand. The end of the block is defined with EPB or EFB. The program code is inserted between the two instructions.

At the start of every block, the ACCU is always High. When a PB or FB is called, the contents of the ACCU is saved, set High as the start of the called block, and is restored when the call returns. This allows the programming of logical IF..THEN..ELSE structures where the blocks are called conditionally.

Notes:
The status flags (Error, Negative, Positive and Zero) are not saved and restored when a PB of FB is called, only the ACCU state is saved. If the status flags need to be saved, they must be copied into normal Flags.
As with COBs, PBs and FBs should not contain wait loops and delays, and jumps to out of the block are not allowed. As a general rule, backward jumps should not be used.
### 7.2.3 Function Block with parameters

The following example shows an FB that makes an output flash. The FB is called twice. Its first call makes output 64 flash at a rate of 1.5 seconds. Its second call makes output 65 flash at a rate of 3 seconds.

```
;Blinker FB
FB  1    ;start of FB

;FB parameters
  tempo  LDEF =1   ;[T] timer
  delay  LDEF =2   ;[W] delay between flashes
  blinker LDEF =3  ;[O|F] output or flag to be controlled
STL = tempo ;if timer is 0
LDL = tempo ;initialise timer with parameter 'delay'
  = delay
COM = blinker ;toggle parameter 3
EFB ;end of FB

COB 0
  0

CFB 1    ;call the FB for the first time
  T 1
    15
    0  64

CFB 1    ;call the FB for the second time
  T 2
    30
    0  65
ECOB
```

As already stated, the only difference between PBs and FBs is that FBs can be called with parameters. The CFB instruction is followed by the list of parameters, numbered from 1 to a maximum of 255. Inside the black, parameter numbers can optionally be defined with symbol names, which are local to the block.

Symbols for FB parameters are indicated by '=' followed by the parameter number. For example, \texttt{STL = 1}. Or you can define a symbol with a value "= 1", as shown in the example above.

Note: Fupla programs do not support calling FBs with parameters.

Almost all instructions can reference FB parameters. One exception is the LD (Load) instruction, because it needs a 32-bit operand, and FB parameters are 16 bits. A 32-bit value can be transferred using two instructions \texttt{LDH} and \texttt{LDL} (Load High Word and Load Low Word), which each transfer 16 bits.
When making nested FB calls, it is possible to pass directly from one call to the next:

```
COB 1
  0
CFB 1
  R 1  ;parameter 1 = R 1
EFB

FB 1
CFB 2
  =1 ;call FB 2 with FB 1's parameter 1
EFB

FB 2
INC =1 ;FB 1's parameter 1
CFB 3
  =1 ;call FB 3 with FB 1's parameter 1
EFB

FB 3
DEC =1 ;FB 1's parameter 1 = R 1
EFB
```

7.3 View Block Call Structure

Once you have built your program, you can view its block structure. Click on the toolbar button called *Block Call Structure*. This displays a window showing which COB calls which PB, FB, SB etc. The display below is for the PB example in chapter 7.6.6. It shows that COB 0 calls PB 2 and PB 1.
7.4 Exception Organization Blocks (XOB)

7.4.1 Definition

Exception Organization blocks are called automatically when a particular event or error occurs. Some hardware and software events are handled by XOBs. The events themselves cannot be modified by the user, but the code inside the XOB, which handles the event, can be programmed.

Example 1:
When the PCD is powered, it zeros a register which is used to count pulses INB1 at a maximum frequency of 1kHz. XOB 20 is called whenever INB1 goes from low to high.

Example 2:
Turn on the PCD and take out the battery, the Error LED will light up. If your program contained XOB 2 (see table), the LED would not have come on and XOB 2 would have been executed instead.
### 7.4.2 XOB numbers and descriptions

<table>
<thead>
<tr>
<th>XOB</th>
<th>Description</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Power problem in the main rack (PCD6) or Watchdog (PCD1/2)</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>Power problem in the extension rack (PCD 6 )</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Battery low</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Parity error on the I/O bus (PCD6)</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>No response on a module I/O (PCD4/6)</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>Overload of the system due to multiple events.</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>Instruction not valid</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>To many active tasks (Graftec)</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>To many PB/FB levels</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>Watchdog COB</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>To many index registers used</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>Error flag is set</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>Interruption cyclic</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>Interruption cyclic</td>
<td>3</td>
</tr>
<tr>
<td>16</td>
<td>PCD cold start</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>S-Bus telegram</td>
<td>3</td>
</tr>
<tr>
<td>18</td>
<td>S-Bus telegram</td>
<td>3</td>
</tr>
<tr>
<td>19</td>
<td>S-Bus telegram</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>Interrupt input INB1</td>
<td>3</td>
</tr>
<tr>
<td>21</td>
<td>Interrupt input</td>
<td>3</td>
</tr>
<tr>
<td>22</td>
<td>Interrupt input</td>
<td>3</td>
</tr>
<tr>
<td>23</td>
<td>Interrupt input</td>
<td>3</td>
</tr>
<tr>
<td>25</td>
<td>Interrupt input INB2</td>
<td>3</td>
</tr>
<tr>
<td>26</td>
<td>Cyclic interrupt</td>
<td>2</td>
</tr>
<tr>
<td>27</td>
<td>Cyclic interrupt</td>
<td>2</td>
</tr>
<tr>
<td>28</td>
<td>Cyclic interrupt</td>
<td>2</td>
</tr>
<tr>
<td>29</td>
<td>Cyclic interrupt</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>RIO connection lost</td>
<td>1</td>
</tr>
</tbody>
</table>

If an error occurs and the handler XOB has not been programmed, the Error LED on the front-panel of the PCD is lit, and the program keeps running.

If an error occurs and the handler XOB has been programmed, the Error LED stays off, and the XOB is called. After executing the XOB, the program returns to where it was when the XPB was called.

XOBs have different priorities to ensure that the most important XOBs are processed first. Priority 4 is the highest. Only XOBs 0 and 8 can interrupt the execution of another XOB. If an XOB event occurs during the execution of an XOB with a lower priority, it will be handled as soon as the current XOB is finished.

Error diagnostic XOBs and their programming is described in the following chapters. XOBs cannot be called directly by the user program.
7.4.3 Use of XOBs

Diagnose of errors

XOBs detect the following types of error:
- Incorrect hardware configuration
- Invalid jump addresses
- More than 7 (or 31) program call levels
- More than 32 active transitions in a Graftec structure
- Never-ending loops (COB supervision timeout)
- Mathematical errors (overflow, divide by 0)
- Communications errors

Fupla example:
Using features to detect to location of programming errors.
With Fupla it is easy to use XOBs. They are added automatically by the Fbox: Blocks Controls, Diagnostic XOB
Diagnostic information is available on the function outputs, error counter, XOB number, program line number,....

IL example:
The IL program's diagnostics supply the same information as above in registers Rdiag + 0 ... +12.
Maintenance of your PLC.

- Monitoring of batteries (need to be changed every 3 to 4 years)

Fupla example:
XOB 2 Battery Failure also has its own FBox, System information, Battery. The Err output, which is connected to symbol Battery_Failure goes high if the battery needs replacing.

IL Example:
If there is a PCD battery failure, the Batt LED on the front-panel of the PCD is turned on and XOB 2 will then be called at regular intervals.

In the example below, XOB 2 loads a timer with a delay of 1 second. As the exception block is called regularly, the timer will be initialised frequently and will not have the chance to count down to zero. The binary state of this timer will therefore be high for a battery failure, and going low approximately 1 second after the battery has been replaced.
Monitoring special events or very fast reactions to external signals.

- interrupt inputs
- interruption of the program every once in a while
- interruption of the program when a telegram arrives
- cold start. Initial values

**Fupla example:**
Output pulses to a digital output. Use functions Blocks Controls, Install cyclic task and Binary, Arithmetic, Move.

**IL example:**

```plaintext
;Start-up XOB 16
XOB 16
SYSWR 4014 ;Initialise XOB 14
     1000 ;with a 1000ms interrupt
EXOB

COB 0
     0
... ;user program goes here
ECOB

;Cyclic Interrupt
XOB 14
COM 0 32 ;toggles output 0 32
EXOB
```
7.4.4 History

The History window shows all the hardware or software errors which have occurred while the PCD was operating. History messages are always added to this list, even if the associated XOB handler is programmed. Examine this list if the PCD’s Error lamp is lit. To see the history table, start the Online Configurator and press the History… button.
7.4.5 Description of XOBs

**XOB 0: Power failure in main rack**
The voltage monitor in the supply module of the main rack has detected an excessive drop in voltage. All outputs are reset, XOB 0 is called and the PCD is put into the Halt state.
From the moment when XOB 0 is invoked until CPU HALT is an interval of approx. 5 ms. During this time, XOB 0 continues processing, so that data can still be saved.

**XOB 1: Power failure in extension rack**
The voltage monitor in the supply module of an extension rack detected an excessive drop in voltage. In this case all outputs of the extension rack are set low within 2ms and XOB 1 is called.
If outputs from this "dead" extension rack continue to be handled (set, reset or polled) by the user program, XOB 4 and/or XOB 5 are also called.

**XOB 2: Battery failure or low battery**
The battery is low, has failed or is missing. Information in non-volatile flags, registers or the user program in RAM as well as the hardware clock may be altered. The battery can fail even if the PCD is not in use for a long time. Even a new PCD which has never been used can show the same symptoms.

**XOB 4: Parity error on address bus**
XOB 4 can only be called if the PCD has extension modules. The monitor circuit of the address bus has noticed a parity error. This can either arise from a faulty extension cable, a defective extension rack or a bus extension module, or else it is simply because the extension rack addressed is not present. If there is a fault, the wrong element could be addressed.

**XOB 5: No response from I/O module**
The PCD's input and output modules return a signal when they are accessed. If this signal is not returned, XOB 5 is called. Generally occurs if the module is not present, but it can also happen in case of faulty address decoding on the module.
On the PCD1/2/3, this mechanism is not implemented.
XOB 7: System overload
The waiting mechanism for XOBs with priority levels 2 or 3 is overloaded. If a level 2 or 3 XOB is processed at the same instant as an XOB with a higher priority (level 4), the lower priority XOB is put on hold until the XOB with priority has finished. XOB 7 is called when the queue is full.

XOB 8: Invalid opcode
The PCD tried to execute an invalid instruction code. This can occur if the firmware is old and does not support the instruction, or if the program has been incorrectly edited using the debugger S-Bug. Other errors are: calling non-existent blocks; missing end of block instructions; program jumps to the second line of multi-line instructions; jumps from one block into another, etc.

XOB 9: Too many active Graftec branches
More than 32 Graftec branches were simultaneously activate in a Sequential Block (SB). More than 32 parallel branches can be programmed in a single SB, however, only a maximum of 32 are allowed to run simultaneously.

XOB 10: More than 7 (31) nested PB/FB calls
PBs and FBs can be nested to a depth of 7 (or 31) levels. An additional call (calling the 8th or 32nd level) causes XOB 10 to be called. The 8th/32nd level call is not executed.

XOB 11: COB monitoring time exceeded
If the second line of the COB instruction is a non-zero supervision time (in 1/100ths of a second), and if COB processing time exceeds this duration, XOB 11 is called. COB processing time is the time which elapses between the COB and ECOB instructions. The original purpose of this monitoring time was the immediate discovery and subsequent eradication of any long delays in the user program resulting from bad programming (wait loops, over-long count loops). It is in fact, a "software watchdog". As already mentioned, wait and count loops (program jumps) are not encouraged. This minimizes the possibility of blocking user programs. However, even in properly structured programs, one or more COBs may be programmed with very lengthy mathematical calculations etc. which cause a long execution time, and other COBs with only monitoring and control functions may be delayed. If a monitoring time defined for this lengthy calculation program elapses, the COB will be abandoned to continue from the start of the next COB. The "release point" is automatically stored in memory together with the ACCU status. When the original COB is next invoked, it will continue from the release address+1. If this technique is used, XOB 11 should not be programmed as otherwise time is wasted calling XOB 11 when the timeout is not actually an error. A further programming technique (timeslice) is explained in "Other programming techniques".
XOB 12: Index Register overflow
The size of the Index Register is 13 bits (0 to 8191). This is sufficient to reference all element addresses. If a program contains an indexed element which falls outside its address range, then XOB 12 is called.
For example, the indexed Flag 8000 is referenced and the Index Register contains 500, such that flag 8500 would be referenced, which lies outside the Flag's address range of 0..8191.

XOB 13: Error flag set
Many instructions can set the Error flag, see the "Reference Guide", status flags.
If an error occurs, the Error flag is set and XOB 13 is called, so that error can be handled (alarm, error message to a printer, etc). XOB 13 is always called when the Error flag is set, irrespective of whether the cause is a calculation, data transfer or communications error.
If a more detailed diagnostics is needed for the Error flag, a PB (or FB) can be conditionally called after every instruction which could set the Error flag.

Example:
...
DIV R 500 ;value 1
R 520 ;value 2
R 550 ;result
R 551 ;remainder
CPB E 73 ;if error then call PB 73 ....
...
PB 73 ;divide by zero
SET O 99
INC C 1591
EPB
...

PB 73 is called after a divide by zero, which turns on Output 99. Counter C 1591 counts how often this event occurs. An overflow from multiplication could, for example, activate output 98 and increment counter 1590.

XOB 13 should also be programmed, but can be empty.
If it is not programmed, the Error LED on the PCD's front-panel is turned on when the Error flag is set, which may not be satisfactory.

IMPORTANT:
The Error flag and other arithmetic status flags (Positive, Negative, Zero) are set in case of a particular event or state, and, if they are of interest, must be processed immediately, as these status flags always refer to the last executed instruction which can affect them.
For example, if a correct addition had followed the division by zero example above, the Error flag would be reset.
XOB 14, 15: Cyclic interrupt XOBs
XOBs 14 and 15 are called periodically with a frequency of between 10ms and 1000s. This frequency is defined by the SYSWR instruction.

XOB 16: Coldstart
XOB 16 is a start-up block. It is processed when the PCD is powered up or when a "restart cold" command is received from a programming tool. XOB 16 is used to initialise data before starting the program. Once XOB 16 is finished, the program will process COBs in ascending number order, but will never return to XOB 16. XOB 16 cannot be restarted by the user program. If a particular action has to be executable both by a COB and during initialisation, this action must be written in a PB or FB which can be called from XOB 16 and from the COB. XOB 16 has its own Index register which is separate from the Index registers used by the COBs.

XOB 17, 18, 19: Request to call an XOB via S-Bus
These three XOBs can be used as interrupt routines. They can be called when a particular S-Bus message is received. The SYSWR instruction or Fupla FBox Special, Execute XOB can also be used to call them.

XOB 20, 21, 22, 23, 25: Interrupt input change detected
XOB 20, ..., 25 is called when interrupt input INB1 or INB2 of the PCD has detected a rising edge (see Saia PCD hardware manual for further details).

XOB 25, 26, 27, 28, 29: Cyclic interrupt XOBs
XOBs 25, ..., 29 can be called periodically with a period between 10ms and 1000s. The period is defined by the SYSWR instruction.

XOB 30: Loss of master slave connection with RIOs
The connection is tested after each message sent by the master station to the slave station. If the test is negative, the master PCD calls XOB 30. This occurs, for example, when an online station is disconnected from the network or is powered off.
7.5 **Sequential Blocks** *(SB 0 to 96)*

Sequential blocks SB are a collection of Steps and Transitions. In each step you execute a part of your program and in each transition you wait for a condition to occur in order to continue with the following step. This is known as a Graftec program.

Graftec programs are created using the S-Graf editor, and the files have the extension "*.sfc". The Graftec editor is explained in the next chapter. It is an excellent tool if you have to solve programming tasks, where your installation must deal with a situation in a sequential manner.

SBs can be called from any other block.

7.6 **Summary**

<table>
<thead>
<tr>
<th>Service</th>
<th>Média</th>
<th>Opérand</th>
<th>Notices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyclic Organization Block</td>
<td>COB</td>
<td>0…31</td>
<td>Minimum 1 COB by program</td>
</tr>
<tr>
<td>Programme Block</td>
<td>PB</td>
<td>0…999</td>
<td>Under programs called by a COB, PB, FB, SB or XOB</td>
</tr>
<tr>
<td>Function Block</td>
<td>FB</td>
<td>0…1999</td>
<td>Function with parameters called by a COB, PB, FB, SB or XOB</td>
</tr>
<tr>
<td>Sequential Block</td>
<td>SB</td>
<td>0…95</td>
<td>Sequential under programs called by a COB, PB, FB (SB, XOB)</td>
</tr>
<tr>
<td>Step</td>
<td>ST</td>
<td>0…5999</td>
<td></td>
</tr>
<tr>
<td>Transition</td>
<td>TR</td>
<td>0…5999</td>
<td></td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>8.1.1</td>
<td>Creating a template</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>8.1.2</td>
<td>Importing templates</td>
<td>27</td>
<td></td>
</tr>
</tbody>
</table>
Saia PG5 Graftec is based on the French Grafcet standard NF C-03-190 and IEC 848, but it contains a few differences and improvements. It is also known as "Sequential Function Chart" (SFC).

Graftec is independent of the technology used for its implementation. It standardises the representation of sequential processes using a small number of graphical symbols governed by a few simple rules. A Graftec diagram is composed of steps which define actions, and transitions which check for events.

A sequence consists of a succession of alternating steps and transitions. A step is not executed until the preceding transition allows it.

The Saia PG5 Graftec editor creates all the instructions necessary to generate the diagram within a Sequential Block (SB).

A Graftec application is programmed in two stages:
- The creation of the step/transition diagram which describes the sequential process.
- The coding of the steps and transitions with the Fupla or IL editors, S-Fup or S-Edit.

After the Build and Download of the program into the PCD, the functioning of the program can be observed while it's running. This is a great help when testing and commissioning.

Graftec also allows you to subdivide a large structure into smaller pages. The pages are like a zoom which allows the representation of the process in the desired level of detail.

The execution of a Graftec program is purely sequential and follows the Graftec rules. The result is an optimal speed of execution with a very fast reaction time. Even if the program contains many steps and transitions, only the active transitions are executed. The execution times of other cyclic parts of the program are not influenced, even by a large Graftec program.
8.1 Sequential Blocks (SB 0 to 95)

Because the timing of events is indeterminate, we cannot estimate the cycle time of a sequential program. This makes it important to separate the cyclic programs from the sequential programs.

A wait on a sequential event will never block the execution of a cyclic program. To satisfy this condition, sequential programs are placed inside one of the SB structures which are called on every cycle of the program.

If the sequential program in an SB is waiting for an event, the PCD stops processing the SB and continues with the cyclic programs. The SB is called again on the next cycle of the program.

8.2 Cyclic structure with call to SB 0

Sequential Block
Structure of a Sequential Block (SB)

The Graftec editor (S-Graf) allows an SB to be created using steps and transitions which contain Instruction List or graphical Fupla code.

The Sequential Block starts with an Initial Step, represented by a square drawn with a double line. This is the start of the sequential process which is executed the first time the SB is called (cold start).

The structure must always be closed in a loop.

8.2.1 Rules for connecting Steps and Transitions

The structure of an SB has a simple but strict syntax. As you know, the block starts with the Initial Step, and then alternates between Transitions and Steps. Two Steps or two Transitions can never be directly connected.
8.2.2 Transitions (TR 0..5999)

The Graftec process is controlled by the code inside the transitions, which are active until an event is detected, such as the change of state of an input, output, indicator, or evaluation of a logical expression.

If the program is written in IL, the transition executes the next step only if the accumulator (ACCU) is high (1) at the end of the transition.

If the program is written in Fupla, the transition executes the next step only if the input to the FBox ETR is 1.

If these cases are not fulfilled at the end of the transition then it remains active and the same transition is evaluated repeatedly until the awaited condition occurs.

Example: Detecting the rising edge of an input signal

\[
\begin{align*}
\text{IL program} & & \text{Fupla program} \\
\text{STH} & I & 8 \\
\text{DYN} & F & 80
\end{align*}
\]

Note:
For transitions in instruction list, the ACCU is always high (1) at the start of a transition or a step. Therefore all ACCU dependent instructions are always executed, and empty transitions are always fulfilled.
8.2.3 Steps (ST 0..5999)

Steps contain the programs that are the actions of the process: turn on/off the outputs, flags, calculations, loading counter values, etc.

**Example:** Controlling the axis of a machine with a motor.

We define the destination position, the speed and the direction of the movement, then start the movement. These tasks are executed once, so they are placed in a step.

Next we monitor the movement, and wait for it to reach the destination position. The actual position is compared with the destination position. This task must be done repeatedly, so it is placed in a transition. When the destination is reached, the transition exits with the ACCU or the ETR FBox high, so the transition is fulfilled and the next step is executed.

The next step stops the motor at the destination position. This task is also executed once, so it's placed in a step.

**Note:**
A step without a program passes control directly to the next transition. A step is executed only once, it is not executed periodically like the transitions.

The entire Graftec structure starts with an Initial Step, represented by a square drawn with a double line. This is the start of the sequential process which is executed the first time the SB is called, after a cold start or power up.
8.2.4 Properties of steps and transitions

Selecting a step or transition in the Graftec structure with the mouse displays the following information in a window, if it has been opened with the Block, Properties command.

<table>
<thead>
<tr>
<th>General</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Number</td>
</tr>
<tr>
<td>Comment</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Scope</td>
</tr>
<tr>
<td>Editor</td>
</tr>
</tbody>
</table>

Name: Symbol name for the step or transition.
Number: Number of the step or transition. By default this number is empty, which means it is dynamic and is assigned by the Build. If necessary a number can be defined, either 2000 or 6000 steps and transitions are available according to the PCD type.
Comment: Free text comment which is shown on the right of the step or transition.
Type: Step or Transition.
Scope: Scope of the symbol (Local or Public). If Public then the symbol name can be accessed from other files, but that's not normally necessary for steps or transitions.
Editor: IL Instruction List or Function Block Diagram.
8.2.5 Typical Graftec sequences

Simple sequence
Alternate steps and transitions. Note that two steps or transitions cannot be connected together.

Alternate branching (OU)
A branch with a choice of sequences. The transitions are evaluated from left to right. The first transition to be active determines which sequence is executed. An alternate branch always starts from a step, which is connected to several transitions, and is terminated by transitions which converge into a single step. The Graftec editor supports up to 32 branches. If there are more than 32 branches, XOB 9 is called.

Simultaneous branching (ET)
A simultaneous branch contains several sequences which are executed in parallel at the same time. The simultaneous branch always starts with a transition which is connected to several steps, and terminated by a single synchronising transition. The Graftec editor supports up to 32 parallel branches. If there are more than 32 branches, XOB 9 is called.

Jump sequence
The jump sequence is like an alternate branch which allows a sequence to be processed in a conditional manner.

Repeat sequence
The repeat sequence is also like an alternate branch, but it is connected to a preceding step. In the example, a counter is initialised with the number of loops, then follows a simple sequence of any length, the last step decrements the counter, and if the counter is not zero then it repeats the loop.
8.3 Creating a Graftec project

For this example, we will create a new Project to contain the files for the Graftec program.
- For graphical programming, prepare a Graftec file and a Fupla file.
- For IL programming, prepare a Graftec file and an IL ".src" file.

8.3.1 Create a new project

From Project Manager, use the Project, New menu command to create the new project.

8.3.2 Add a Fupla or IL file

Choose the file type: Fupla File (*.fup) or Instruction List File (*.src).
8.3.3 Calling the SB from a COB

Depending on the choice of language (IL or Fupla), either call the SB with a CSB instruction or a Call SB FBox. Open the new file and write the program as shown below.

**IL Program:**
```
COB 1 ;start of COB
   0
CSB 0 ;call SB 0
ECOB ;end of COB
```

**Fupla Program:**
```
Call SB
# 0
FBox: Blocks Control, Call SB
```

The SB can be called as in the examples above. But there's an even easier way. The PG5 can automatically create a COB containing the CSB instruction to call the SB. This is enabled by an option on the Build Options dialog box, Advanced group, set Generate SB calls to Yes. (This option is set to Yes by default.)

You can create the SB first, as described below, and then add the call afterwards so you can use the symbol name of the SB instead of the number.

8.3.4 Add a Graftec file

Choose the file type:
Graftec File (*.sfc)
8.3.5 Page Navigator, adding an SB

When a new Graftec file is created, the editor automatically creates a sequential block containing the initial step.

Several SBs can be created inside a single Graftec file. The Page Navigator window shows the list of SBs in the file.

If necessary, another SB can be added to the file using the Block, New menu command, then modify the block’s details from the Properties window. This window is also used to modify the properties of the SB which is created when a new file is created.

<table>
<thead>
<tr>
<th>Name</th>
<th>SB_2 SB_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td></td>
</tr>
<tr>
<td>Comment</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>SB</td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
</tbody>
</table>

Name: The symbol name of the block. Assigning meaningful names to the blocks it makes the program easier to understand and maintain.

Comment: Free comment which can describe some details about the block.

Number: Block’s number. By default, the block’s number is blank, so it is assigned dynamically by the Build. If necessary, you can assign a number yourself.

Scope: Scope of the SB’s symbol name (Local or Public). Use Public if the symbol needs to be accessed from other files. For example, if it is called from a COB defined in another file.

To display the Graftec structure of an SB, right-click on the Page Navigator window and select the Open Block command from the context menu.

Next, we will structure the SB with the steps and transitions.
8.4 Editing the Graftec structure

A new Graftec file always contains the initial step, which is executed the first time the SB is run. Additional steps and transitions are added using keyboard or toolbar commands.

8.4.1 Editing a simple sequence

1. Press the Transition Mode toolbar button.
2. Place the mouse pointer on the initial step and click the left-hand mouse button.
3. Press the Step Mode toolbar button.
4. Move the mouse over the new transition and click again.
5. Continue in the same way.

8.4.2 Creating a loop

When the sequence ends, the SB also ends. To repeat the sequence, a loop must be added. Remember that it's illegal to directly connect two steps or two transitions, a loop always starts after a transition, and connects above a step.

1. Press the Select Mode toolbar button
2. Click on the transition before the jump
3. Press the Link Mode toolbar button
4. Click on the step to be jumped to

8.4.3 Smart cursor option

Sequences can also be edited using the button Smart Mode. This mode automatically changes the cursor mode according to the mouse pointer's location.

If the mouse pointer is over a step or transition, then Select Mode is activated and the mouse pointer becomes a hand. Double-clicking opens the program editor to edit the element's code.

Moving the mouse pointer to the bottom of a step or transition changes the mouse pointer to represent the transition or step which would be inserted below if the left-hand mouse button is pressed.

If the mouse pointer is moved over the right hand side of the step or transition, the mouse pointer changes to indicate that an alternate or simultaneous branch will be created if the left-hand mouse button is pressed.
8.4.4 Creating an alternate branch (OU)
1. Activate Transition Mode
2. Select a transition which is already followed by a state
3. Each time the left-hand mouse button is pressed, a new transition is added to the right

8.4.5 Connecting alternate branches
1. Activate Select Mode
2. Select a transition to connect
3. Press the Link mode button
4. Click on the step, the transition is connected after this step

8.4.6 Creating a simultaneous branch (ET)
1. Activate Step Mode
2. Click on the first step to connect a new step on the right
3. Each click of the mouse on a step connects a new step on the right

8.4.7 Connecting simultaneous branches
To synchronize simultaneous branches:
1. Activate Select Mode
2. Select the step to be connected
3. Activate Link Mode
4. Click on the destination transition

8.4.8 Adding a comment
1. Activate Select mode
2. Right-click with the mouse on the step or transition to display the Properties window
3. Enter the comment in the Comment field of the Properties window

Tip: To create a comment on two lines, enter ‘\n’, for example:
Line 1
Line 2

<table>
<thead>
<tr>
<th>General</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Name)</td>
</tr>
<tr>
<td>Number</td>
</tr>
<tr>
<td>Comment</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Scope</td>
</tr>
<tr>
<td>Editor</td>
</tr>
</tbody>
</table>
### 8.4.9 Inserting a sequence

1. Activate Transition Mode
2. Click on a step which already has a transition
3. A new step and transition are inserted

### 8.4.10 Deleting a sequence

1. Activate Select mode.
2. Click on the first step or transition of the sequence to be deleted
3. Hold down the Shift key and click on the last transition or step. The marked sequence in highlighted.
4. Press the Del key

The marked sequence cannot be deleted if it would result in an invalid structure.

### 8.4.11 Copy/pasting a sequence

**To copy a sequence to the clipboard:**
1. Activate Select Mode
2. Click on the first element of the sequence
3. Hold down the Shift key and click on the last element of the sequence
4. Copy the sequence to the clipboard with the menu command Edit, Copy or press Ctrl+C

**To paste the sequence from the clipboard:**
5. Activate Select Mode
6. Click on the element before the sequence to be inserted
7. Use the menu command Edit, Paste or press Ctrl+V
Note:
Depending on the destination element’s type (step or transition) and the elements to be inserted, the sequence will be added below (simple sequence), or to the left (alternate sequence) of the selected element.

Inserting on a step

- Set the counter to 3
- Wait for the start3_p signal
- Turn the output on
- Wait 1 second

Inserting on a transition

- Set the counter to 3
- Wait for the start3_p signal
- Turn the output on
- Wait 1 second
- Turn the output on
- Wait 1 second
8.5 Writing a first sequential block

Objective:
Write a program which blinks output \(O\) \(\text{Three}_p\) three times whenever input \(I\) \(\text{Start}_p\) (\(\text{Start}_3_p\)) is activated.

Timing diagram

Input \(\text{Start}_3_p\)

Output \(\text{Three}_p\)

8.5.1 Creating the Graftec structure

We always start with the initial step which is the start-up entry point. It will initialise the counter. (If it is not used, it can remain empty, without code or a comment.) Next, we wait for the signal \(\text{Start}_3_p\). Edit the transition as below and enter a comment in the Properties window.
At the start of the sequence, set the output *Three_pulses* to the high state for 1 second, then set it low for 1 second. Repeat this operation three times, then repeat the entire sequence including the initial step.

8.5.2 Choosing the editor: IL or Fupla (S-Edit or S-Fup)

Now that the Graftec structure has been created, it remains to write the code for each step and transition, using the *Instruction List* editor (S-Edit) or the *Function Block Diagram* editor (S-Fup). The default choice of editor is defined from the *Options* dialog box, *Code Editor, Default editor*. 
8.5.3  Editing the symbols

First declare all the data using the Symbol Editor, assigning symbol names, types, addresses and comments, as shown in the window below.

![Symbols window]

8.5.4  Programming an initial step, loading a counter

Open the initial step to add the code which loads counter *PulseCounter* with the constant *Number_of_pulses* (3). Double-click on the element to be edited and choose the editor *IL Instruction List* or *Function Block Diagram* (Fupla)

**Fupla Program:**
Use the FBox: *Counter, Counter (Graftec), Load Counter*

**Important:** Do not use timers or counters from other families which are reserved for cyclic programs.

![Fupla diagram]

**IL Program:**

```
LD  PulseCounter  ;initialise the counter
Number_of_pulses
```

8.5.5  Programming a transition, waiting for the start signal

A transition is reprocessed repeatedly until the ETR FBox input is high (Fupla), or the ACCU is high (IL), at the end of the transition. Transition 0 waits for the input *Start_3_pulses* to go high.

**Fupla program:**

![Fupla diagram]

**IL program:**

```
STH  Start_3_pulses  ;Start High – set the ACCU to the state
                     ;of input Start_3_pulses
```
8.5.6 Programming a step, turning on an output and starting a timer

This step turns on the output and loads the timer, then continues to the next transition to wait for the timer to reach 0.

**Fupla program:**
The timers and counters in the Fupla standard library are not designed for SBs, they are designed for COBs which are executed cyclically. Instead, you must use the functions in the Graftec family which are specially designed for SBs. These can be loaded in a step, and interrogated later in a transition.

**IL program:**

```
SET Three_pulses ; turn on the output
LD One_S_Timer   ; start the timer
Pulse_Time
```

8.5.7 Waiting for a timer

**Fupla program:**

```
STL One_S_Timer ; Set the ACCU high if the timer has reached 0
```

**IL program:**

```
STL One_S_Timer ; Set the ACCU high if the timer has reached 0
```

8.5.8 Turning off an output when a timer reaches 0

Step and transition 2 are similar to step and transition 1, except that output Three_pulses is set low, and a different timer is started.

**Fupla program:**

```
RES Three_pulses ; turn on the output
LD Pulse_Timer   ; load the timer
Pulse_Time
```

**IL program:**

```
RES Three_pulses ; turn on the output
LD Pulse_Timer   ; load the timer
Pulse_Time
```
Programme Fupla:

\[ \text{:2 } \quad \text{Wait on other second} \quad \text{Timer}=0 \quad \text{ETR} \]

Programme IL:

\[
\text{STL Pulse_Timer} \quad ;\text{Set the ACCU high if the timer has reached 0}
\]

Note: For steps and transitions 1 and 2 we have used two different timers (One_S_Timer and Pulse_Timer), but to economise on the number of timers used we could have used one, by using the same timer twice, because they are not used at the same time.

8.5.9 Decrementing a counter

Fupla program:
The counter is decremented each time the step is executed, because the FBox input is high (1).

\[
\text{:3 } \quad \text{Decrement Counter} \quad 1 \quad \text{Dec Counter} \quad \text{PulseCounter}
\]

IL program:

\[
\text{DEC PulseCounter } ;\text{if the ACCU is high, decrement the counter}
\]

Remember that the ACCU is always high (1) at the start of an ST or TR, so the ACCU-dependent instructions are always executed.

8.5.10 Alternate branching

The next two transitions make a choice.

Fupla program:

\[
\text{:3 } \quad \text{Counter} = 0 \quad \text{Counter=0} \quad \text{ETR} \quad \text{Transition 3: The input to FBox ETR is 1 if the counter is 0.}
\]

IL program:

\[
\text{STL PulseCounter} \quad \text{Transition 4: The input to FBox ETR is 1 if the counter is not zero.}
\]

Place an inverter at the input of the ETR FBox with the toolbar button Invert Binary Connector.
8.6 Building and debugging the program

Once the programming is complete, press the Build button on the toolbar to compile, assemble and link the entire program.

8.6.1 Messages window

Project Manager's Messages window indicates the results of the build. If the program was created without errors, the last line in the window will show:

Build successful. Total errors: 0
Total warnings: 0

Error messages are shown in red. Double-clicking on an error message will usually take you to the part of the program which contains the error.

Project Manager also has an Error List window which shows only the error and warning messages.

8.6.2 Online tools

It remains to download the program and go into online mode.

Using the Graftec editor, the execution of the sequential block can be visualised when online. A red dot indicates the active transition, so the sequential operation can be viewed.

We can stop the program at any time by pressing the Stop button, and then continue the program step-by-step.

Each time the Step By Step button is pressed, a step or transition executed.

The Step In button opens the step or transition with the Fupla or IL editor, so that the code inside the step or transition can be stepped through.

The Run to Element button processes all the Graftec steps and transitions and stops when the selected element is reached. If the program is already running, it will stop at the selected element.
8.7 Grouping a Graftec program into pages

Graftec allows a sequence of steps and transitions to be grouped into a new element called a Page. The element looks like a step, and can have its own comment, but is distinguished by an additional vertical line on the left hand side.

Pages allow the Graftec structure to represent the process from a higher level, and each page can be opened to display the next level of detail, which is composed of more pages, steps and transitions.

Pages can themselves contain pages, without any nesting depth limitations. The nesting of pages is bit like a zoom function which permits the representation of the program to several levels of detail.

8.7.1 Rules for editing pages

Graftec sequences to be made into pages must respect a few rules:

- A page must always start and end with a step.
- The sequence cannot be only one step.
- The entry and exit steps cannot be deleted.
8.7.2 Creating a new page

To create a new page, proceed as follows:

- Activate Select mode
- Select the first step for the page
- Hold down the Shift key and select the last page of the sequence
- Use the menu command Page, Create

8.7.3 Opening pages

To view the contents of a page, select the page and use the Page, Subpage menu command or toolbar button.

The command Page, Calling collapses the page to view the level above. The top level page is shown by the command Go To Main

8.7.4 Expanding a page

To replace a page with its original sequence, select the page and use the Page, Expand command.
8.7.5 Block Navigator

It is strongly recommended to reduce the size of a large Graftec structure by using pages. This makes it easier to read the program and to navigate between the high level functionality represented by the pages.

The Block Navigator provides a global view of all the SBs and pages in the file. The selection of a block or a page in this view displays the corresponding block or page without having to search for it in the Graftec structure.
8.8 Graftec templates
Sequences of steps and transitions can be grouped into templates which can be used like a library of sequences in other programs.

8.1.1 Creating a template

It is very easy to create a template. Select a sequence of steps and transitions, and use the Edit, Add to templates command. The command is also on the context menu. A dialog box is displayed which prompts for a group name, template name and comment.

Templates are organised into Groups which are comparable to FBox Families. The groups classify the templates according to the criteria defined by the author. Templates are listed below their group names in the Templates window.

The icons show how the template starts and ends. Any sequence can be made into a template, they can contain pages, branches, etc, and be coded in Fupla or IL.
8.1.2 Importing templates

The templates can be used in any projects. Open the Templates window using View, Templates, and drag-and-drop the template into the Graftec structure. All the steps, transitions, branches, symbols, comments and associated Fupla or IL code are inserted.

A dialog box is displayed which allows the modification of the names, addresses, comments and scopes of symbols imported by the template, and changes to various other data. This functionality can be compared to a Macro or Function Box with parameters.

![Dialog box for importing templates]

The Symbol List tab shows all the symbols contained in the template. The fastest way to rename the symbols and prevent duplication of symbol names is to put all the symbols into a symbol Group. The context menu command Insert Pre-group will place the symbols below into a group with the name of your choice.

![Symbol List tab]

To update the addresses of symbols, they can be sorted by type by pressing the column header button Type, edit the address of the first element, then drag the tiny square on the bottom right of the cell downwards to select all the addresses to be renumbered.

To import the same template several times, see the parameters on the General tab. It allows an index to be inserted into the symbol names or groups using the # character. This character is automatically replaced by the index number, incremented by one for each copy of the template. The context menu command Indexing can also be used.
9 PROGRAMMING IN IL (INSTRUCTION LIST) .......................................................... 2

9.1 Preparing an IL project .......................................................................................... 3
  9.1.1 Create new project ........................................................................................... 3
  9.1.2 Create new IL file ............................................................................................ 3

9.2 Layout of the IL Editor window ............................................................................ 4
  9.2.1 Editing a line of code ....................................................................................... 5
  9.2.2 Automatic formatting of instruction lines ....................................................... 6
  9.2.3 Creating an organization block ........................................................................ 6
  9.2.4 Sequence of processing for instructions and blocks ..................................... 6
  9.2.5 Rules to follow when editing blocks ............................................................... 7

9.3 Introduction to the PCD instruction set .................................................................. 8
  9.3.1 The accumulator (ACCU) ................................................................................ 8
  9.3.2 Binary instructions ......................................................................................... 9
  9.3.3 Edge detection ................................................................................................ 13
  9.3.4 Status flags .................................................................................................... 14
  9.3.5 Instructions for timers ................................................................................... 15
  9.3.6 Instructions for counters ............................................................................... 17
  9.3.7 Accumulator-dependent instructions ............................................................. 18
  9.3.8 Word instructions for integer arithmetic ....................................................... 19
  9.3.9 Word instructions for floating-point arithmetic ............................................ 20
  9.3.10 Conversion of integer and floating-point registers ..................................... 20
  9.3.11 Index register ............................................................................................... 20
  9.3.12 Program jumps ............................................................................................. 22

9.4 Editing a first application program ....................................................................... 24

9.5 Building the program ............................................................................................ 26

9.6 Load program into PCD ....................................................................................... 27

9.7 Debugging a program ........................................................................................... 27
  9.7.1 Go On/Offline, Run and Stop ......................................................................... 28
  9.7.2 Step-by-step mode ......................................................................................... 29
  9.7.3 Breakpoints .................................................................................................. 30
  9.7.4 Online modification of the program and data ............................................. 31
  9.7.5 Viewing and modifying symbol states with the Watch Window ................ 32

9.8 Commissioning an analogue module ................................................................... 33
  9.8.1 Example for PCD2.W340 analogue input modules ..................................... 33
  9.8.2 Example for PCD2.W610 analogue output modules ..................................... 34
Programming in IL (Instruction List)

The Saia PG5 IL editor (S-Edit) can be used to create the fastest and most efficient programs for Saia PCD controllers. IL stands for Instruction List, which is a low level non-graphical language using the PCD's native instruction set. All PCD models share the same instruction set, thereby guaranteeing portability of programs from one PCD to another.

The IL editor is more than just a valuable aid to program editing, it is also a diagnostic and on-line testing tool.
9.1 Preparing an IL project
First, prepare a new project and file in which to edit the IL program.

9.1.1 Create new project
In the Saia PG5 Project Manager window, select menu File, Project, New… and create the new project.

9.1.2 Create new IL file
To add a new program file to the project, select the folder Program Files, right-click with the mouse, and select menu New… (or press the New File button on the toolbar):
9.2 Layout of the IL Editor window

The IL editor is similar to any other commercial text editor. The same editing features are present, such as Copy/Paste or Find/Replace. However, the IL editor offers more than just conventional text editing:

- Automatic formatting of each line as it's entered
- Syntax colouring, enabling each type of information to be identified
- Integrated Symbol Editor window
- Online step-by-step and debugging features
### 9.2.1 Editing a line of code

<table>
<thead>
<tr>
<th>Label</th>
<th>Mnemo.</th>
<th>Operand</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>;Increment a register</td>
<td>STH</td>
<td>Flag</td>
<td>;Copy the flag state into the accu</td>
</tr>
<tr>
<td></td>
<td>DYN</td>
<td>Dflag</td>
<td>;On a positive flank of the flag, set the accu eight</td>
</tr>
<tr>
<td></td>
<td>JR</td>
<td>L Next</td>
<td>;If the accu is low, jump to label Next</td>
</tr>
<tr>
<td>INC</td>
<td>Register</td>
<td>;Increment the register</td>
<td></td>
</tr>
</tbody>
</table>

Next: NOP ;No instruction

IL program lines are formatted into 4 columns:

**Label**
Represented by the default colour red, the label is a symbol name for a program line, which is used as a destination for jump instructions, e.g. JR L Next.

**Mnemonic**
Represented by the default colour blue, the mnemonic is the instruction's name.

**Operand**
Represented by the default colour black, the operand defines the data type: input, output, flag, register, etc. which the instruction will operate on.

The **View Symbols or Values** button allows either the operand address or its symbol to be displayed.

**Comment**
Program comments are shown in green and begin with a semi-colon. They are aligned on the right of the mnemonic and operand, but may also occupy a whole line.

If the comment will take several lines, it can be edited between two directives: $SKIP and $ENDSKIP. These tell the assembler to disregard all text which appears between them.

```
$SKIP
Author:  DuPont Fred
Date: 28.10.2003
File: Logic.src
$ENDSKIP
```

The **View User or Auto Comment** button can be used to view either the user comments attached to each line of the program, or the automatic comments attached to each symbol as defined in the Symbol Editor window.
9.2.2 Automatic formatting of instruction lines

If the Auto Format while typing option is enabled, whenever Enter is pressed at the end of a line, the line is auto-formatted using the user-defined column widths. Options are defined from the menu command Tools, Options.

If auto-formatting is off, lines can be marked and formatted using the Tools, Auto Format menu command.

9.2.3 Creating an organization block

The Saia PCD programming language is structured using organization blocks, in which the user writes application programs.

Each block provides a particular service: cyclical organization blocks (COB) for cyclical programs; sequential blocks (SB) for sequential programs, program blocks (PB) for subroutines; function blocks (FB) for subroutines with parameters; exception organization blocks (XOB) for exception routines.

Blocks are delimited by a start instruction and an end instruction. For example, the instruction COB marks the start of a cyclic organization block, which ends with the same instruction preceded by the letter E for "end" (ECOB). All program code belonging to this block must be placed between the instructions COB and ECOB, never outside the block.

Even the smallest PCD program will always have a COB. Other blocks may then be added as required.

9.2.4 Sequence of processing for instructions and blocks

Within each block, the PCD processes program instructions line by line, from the start instruction to the end-of-block instruction.

The order in which instruction lines are written within an organization block is important. However, the order in which the organization blocks themselves are written is not important. Different rules define the sequence of block processing:

In a PCD coldstart, the programmable controller first looks for XOB 16, the coldstart block. If it is present, it will always be processed first, regardless of whether it is at the beginning or end of the file.

Then, the machine looks for COBs in the program and processes them in numerical order: COB 0, COB 1, ... COB 15, regardless of the order in which they appear in the file. After the last COB, the program will start again from COB 0.
All the blocks for sequential programs (SB), subroutines (PB) and functions (FB) are called by the user program with the instructions CSB (Call SB), CPB (Call PB) and CFB (Call FB). The user program therefore determines when and in what order SBs, PBs and FBs are processed.

All exception blocks are automatically called as soon as the particular event concerned occurs. These events are unpredictable and may happen at any time. The order in which they are processed cannot be defined. Each hardware or software event is linked to a distinct XOB. These events cannot be modified by the user. However, the user is free to program which action to take within each of the XOBs.

9.2.5 Rules to follow when editing blocks

Even though blocks can be written in any order, the following rules must be followed:

Blocks cannot be written inside other blocks. They must always follow each other.
No program instructions may be defined outside a block, with the exception of symbol definitions, texts and data blocks.
9.3 Introduction to the PCD instruction set

This section provides an overview of the PCD instruction set. For more detailed information, consult the full description of each instruction given in the Instruction Guide 26/733 or on the PG5 help screens. To obtain specific help about an instruction from the IL editor: write the instruction, put the cursor on it and press key F1. General help is also available with the menu Help, Instruction List Help.

9.3.1 The accumulator (ACCU)

The accumulator is a binary value whose state is set by binary instructions and a some integer instructions. The PCD has just one accumulator, which may be considered as a special kind of flag. The state of the accumulator can be forced with the ACC instruction, which also allows the accumulator to be set to the value of a status flag (see description of status flags).

**Examples:**

ACC H
Forces accumulator state high

ACC L
Forces accumulator state low

ACC C
Inverts (complements) accumulator state
9.3.2 Binary instructions

Binary instructions use operands that may have just one of two distinct states: 0 or 1 (low or high). These instructions are used to perform binary equations with the states of PCD inputs, outputs, flags, counters and timers.

Binary instructions always involve the accumulator. Some binary instructions affect the state of the accumulator:

**Examples:**

ACC H
Forces accumulator state high

ACC L
Forces accumulator state low

STH I 4
Copies state at input 4 to accumulator. The accumulator state will be high if 24 volts are applied to input 4. The accumulator state will be low if zero volts are applied to input 4.

Other instructions read the state of the accumulator to execute a binary function and put the result back into the accumulator:

**Examples:**

ANH I 5
Reads accumulator state and executes logical AND function with state of input 5. The accumulator is set to the result.

ORH F 100
Reads accumulator state and executes logical OR function with the state of flag 100. The accumulator is set to the result.

XOR T 3
Reads accumulator state and executes logical XOR function with the state of timer 3. The accumulator is set to the result.
The result of any binary equation is always saved in the accumulator. The OUT instruction allows the content of the accumulator to be copied to an output or flag:

**Example:**

```
OUT O 32
```

Copies accumulator state to output 32.
If accumulator state is high, 24 volts will be applied to output 32.
If accumulator state is low, zero volts will be applied to output 32.

**Example:** programming a simple binary equation

This example of a program performs the binary equation: \( O_{32} = I_0 \cdot I_1 + I_2 + I_3 \cdot I_4 \cdot I_5 \)
It may also be represented by the following diagram:

A binary equation always starts with a STH or STL instruction, which will then be followed by the necessary ANH (*), ORH (+), XOR functions.
Note that the ORH instruction has priority over ANH. Each ORH instruction marks the start of a new line of contacts in the above diagram. The partial or final result of a binary equation is always put in the accumulator. The OUT instruction enables the accumulator result to be used to modify the state of an output or flag.

```
COB 0 ;Start of cyclic program
0
STH I 0 ;Copies state of input I 0 to accumulator: Accu = I0
ANH I 1 ;AND function between state of accumulator and input 1: Accu = I0*I1
ORH I 2 ;OR function between state of accumulator and input 2: Accu = I0*I1+I2
ORH I 3 ; Accu = I0*I1+I2+I3
ANH I 4 ; Accu = I0*I1+I2+I3*I4
ANH I 5 ; Accu = I0*I1+I2+I3*I4*I5
OUT O 32 ;Copies result of equation present in accumulator to output
ECOB ;End of cyclic program
```
**Example:** Programming a binary equation with a specific order of evaluation.

This example of a program performs the binary equation: \( O33 = (I1 \times I2 + I4) \times I3 \)

It may also be represented by the following diagram:

```
  11  12  14  O33
    |    |    |
    13
```

It is sometimes necessary to change the order of priority of binary functions. Generally, we do this by putting brackets into the equations. However, the PCD instruction set does not support brackets. The equation must therefore be divided into two smaller equations. The first equation works out the result of the bracketed part and saves it temporarily to a flag, while the second equation takes the interim result saved on the flag and calculates the final result.

```
COB 0
  0
  STH I 1 ;First equation
  ANH I 2
  ORH I 4
  OUT F 0 ;Result of bracketed function: F0 = (I1 \times I2 + I4)

STH F 0 ;Second equation
  ANH I 3
  OUT O 33 ;Final result: O 33 = F0 \times I3
ECOB
```

Other binary instructions also allow the accumulator to be used to modify the state of an output or flag. Each instruction supports a different function.

**Example:**

- **SET O 32**
  If accumulator state is high, output 32 will be forced high. Otherwise the output will remain in its current state.

- **RES O 32**
  If accumulator state is high, output 32 will be forced low. Otherwise the output will remain in its current state.

- **COM O 33**
  If accumulator state is high, output 33 will be inverted high. Otherwise the output will remain in its current state.
**Example:**
This example shows differences between the instructions OUT, SET, RES, and COM

```plaintext
COB 0
STH I 0
OUT O 32 ; copy I 0 to O 32
STH I 0
SET O 33 ; set output 33 to 1
STH I 1
RES O 33 ; set output 33 to 0
STH I 0 ; on rising edge of I 0
DYN F 1
COM O 34 ; invert output 34
ECOB
```

Some binary instructions end with the letter H or L. Instructions that end with L will invert the state of any information before performing their function.

**Examples:**

**STH I 4**
Copies state of input 4 to accumulator. Accumulator state is high if 24 volts are applied to input 4.

**STL I 4**
Copies inverse state of input 4 to accumulator. Accumulator state is low if 24 volts are applied to input 4.

**ANH I 5**
Performs a logical AND function between the accumulator state and the state of input 5.

**ANL I 5**
Performs a logical AND function between the accumulator state and the inverse state of input 5.
9.3.3 Edge detection

Binary instructions generally use the low or high binary state to perform a binary function or modify the state of an output or flag.

Sometimes it is not the low or high binary state that interests us, but the change of state.

To detect a rising edge, proceed as follows: place the result of a binary equation in the accumulator and use the DYN instruction to find the positive change. After the DYN instruction, the accumulator state will be high if a positive change has been detected, otherwise it will be low. The flag used by the DYN instruction may only be used for a single 'dynamisation' instruction. This is because it is used to conserve the state for the next program cycle.

**Example:** detection of a rising edge

```
STH I 0
DYN F 3
COM O 34
```

**Example:** detection of a falling edge

```
STL I 0
DYN F 3
COM O 34
```

To help you see the influence of the DYN instruction on the program shown above, try removing the DYN instruction and see how the program behaves.


### 9.3.4 Status flags

Unlike binary instructions, integer 'word' instructions rarely use the accumulator. However, they almost always modify status flags.

The PCD's 4 status flags are modified by word instructions and inform us of the result.

| Flag positive | P | Set if the result is positive. |
| Flag negative | N | Set if the result is negative. |
| Flag zero     | Z | Set if the result is zero.    |
| Flag error    | E | Set in case of error.        |

The error flag may be set for a number of reasons, causing the exception block XOB 13 to be called:
- Overflow caused by an instruction which multiplies two large numbers
- Division by zero
- Square root of a negative number
- Error assigning the communications interface (SASI instruction)
- etc

**Example:** Status flags after a subtraction

Status flags are set depending on the result of a subtraction \((R_3 = R_1 - R_2)\). Register values are shown in square brackets \([\,]\). The result of the subtraction is negative: flag N alone is set.

If necessary, status flags can be copied to the accumulator for use with binary instructions, program jump instructions, or when calling PBs, FBs or SBs:

| ACC P | Copy status flag P to accumulator |
| ACC N | Copy status flag N to accumulator |
| ACC Z | Copy status flag Z to accumulator |
| ACC E | Copy status flag E to accumulator |
9.3.5 Instructions for timers

Timers contain two values: the integer delay time value and the timer's binary state. To implement a delay time, load the time value as a positive integer that will determine the length of the delay time in tenths of a second. The controller will automatically decrement the time value until it reaches zero. The timer's binary state is high while the time value is decrementing, and goes low when the time value reaches zero.

**Loading a delay time**

<table>
<thead>
<tr>
<th>IL program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD T 4</td>
<td>If the accumulator state is high, timer T 4 will be loaded with a constant of 10. Otherwise the timer will keep its current value.</td>
</tr>
</tbody>
</table>

**Reading the state of the timer**

<table>
<thead>
<tr>
<th>IL program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STH T 4, ANH T 4, ORH T 4, ...</td>
<td>Use a binary instruction, such as:</td>
</tr>
</tbody>
</table>

**Example:**
Send a one second pulse to output 36 for each rising edge at input 2

State diagram:

```
I 2
Integer value T 4
Binary state T 4
O 36 1s 1s
Time
```

IL program:

```
COB 0
  0
STH I 2 ;Detection of rising edge at input 2
DYN F 2 ;sets accu state high
LD T 4 ;If accu is high, load time delay for 10 units of time
  10
STH T 4 ;Copy logical state of time delay to output 36
OUT O 36
ECOB
```

¹ A time base other than 1/10th of a second (default value) can also be set. This can be done from the Build Options.
**Example:**
Send a one-second pulse to output 37 with a 5 second delay for each rising edge at input 3

State diagram:

```
  I 3

  Integer value T 2
  Binary state T 2

  Integer value T 3
  Binary state T 3

  O 37
```

**IL program:**

```
COB  0
  0
STH I 3
DYN F 3
LD T 2
  50
LD T 3
  60
STH T 2
XOR T 3
OUT O 37
ECOB
```
9.3.6 Instructions for counters

Like timers, counters also have two values: the integer count value and the binary state of the counter.

To implement counting, load the counter with a positive integer value.

Unlike timers, counters are only incremented or decremented by instructions in the user program. The counter's binary state is high when the count value is greater than zero and goes low when the count value reaches zero.

**Status flags**

Instructions INC and DEC counter modify the status flags depending on the result of the operation (Positive, Negative, Zero, Error).

**Example:** Counting pulses from a binary input with a counter.

```
COB 0

STH I 2 ; Copy input state to accumulator
DYN F 3 ; Force accu state high at positive edge of I 2
INC C 35 ; If accu state is high, increment counter
ECOB
```

Instructions STH and DYN read information from input 2 and set the accu state high for a rising edge or low in the absence of an edge. Depending on the accu state, the INC instruction will increment counter 35.
9.3.7 **Accumulator-dependent instructions**

We have seen that binary instructions make much use of the accumulator, and that some word instructions also use it.

But not all instructions use the accumulator in the same way. There are 7 instructions which use it in a special way. These are the accumulator-dependent instructions. They are only processed if the accumulator has previously been set high. The accumulator state is therefore a determining condition.

The 7 accumulator-dependent instructions are listed below:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET</td>
<td></td>
</tr>
<tr>
<td>RES</td>
<td></td>
</tr>
<tr>
<td>COM</td>
<td></td>
</tr>
<tr>
<td>LD</td>
<td>Only for timers and counters</td>
</tr>
<tr>
<td>LDL</td>
<td>Only for timers and counters</td>
</tr>
<tr>
<td>INC</td>
<td>Only for timers and counters</td>
</tr>
<tr>
<td>DEC</td>
<td>Only for timers and counters</td>
</tr>
</tbody>
</table>

**Example:**

Create a time base that inverts an output once every second.

This example uses three instructions. The first (STL) uses the accumulator to put in it the timer's inverse state. The following two (LD and COM) depend on the accumulator. They will only load the time base and invert the output if the accumulator has previously been set high by the instruction *STL*.

```
COB 0
0
STL T 1 ;If the timer state is low, the accu state will be high
LD T 1 ;load time delay with 10 units of time
  10
COM O 38 ;invert output state
ECOB
```
9.3.8 **Word instructions for integer arithmetic**

These instructions are used for calculating arithmetical equations using integer format registers and constants. Each arithmetical instruction has several lines and applies operands to registers or constants, but the result will always be placed in a register.

<table>
<thead>
<tr>
<th>Addition</th>
<th>Subtraction</th>
<th>Square root</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD R 0</td>
<td>SUB R 0</td>
<td>SQR R 100</td>
</tr>
<tr>
<td>R 1</td>
<td>K 18</td>
<td>R 101</td>
</tr>
<tr>
<td>R 3 ; R 3 = R 0 + R 1</td>
<td>R 3 ; R 3 = R 0 - 18</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Multiplication</th>
<th>Division</th>
<th>Comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUL K 5</td>
<td>DIV R 0</td>
<td>CMP R 0</td>
</tr>
<tr>
<td>R 1</td>
<td>R 1</td>
<td>R 1</td>
</tr>
<tr>
<td>R 3 ; R 3 = 5 * R 1</td>
<td>R 3 ; R 3 = R 0 / R 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R 4 ; Reste</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Increment</th>
<th>Decrement</th>
<th>Initialize register</th>
</tr>
</thead>
<tbody>
<tr>
<td>INC R 0 ; R 0 = R 0 + 1</td>
<td>INC R 0 ; R 0 = R 0 + 1</td>
<td>LD R 0 K 19 ; R 0 = 19</td>
</tr>
</tbody>
</table>

**Status flags**

All the above arithmetical instructions modify status flags according to the result of the operation (Positive, Negative, Zero, Error), with the exception of the instruction for loading a register with a constant (LD).

**Differences between registers and timers/counters**

Unlike counters, the instructions for loading a constant into a register, incrementing a register or decrementing a register are not dependent on accumulator state.

The register value to be incremented or decremented may be either a positive or negative integer.

**Example:**

Compare the contents of two registers and switch on three outputs, according to the following conditions:

<table>
<thead>
<tr>
<th>Registers</th>
<th>O 32</th>
<th>O 33</th>
<th>O 34</th>
</tr>
</thead>
<tbody>
<tr>
<td>R 0 &gt; R 1</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>R 0 = R 1</td>
<td>Low</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>R 0 &lt; R 1</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
</tr>
</tbody>
</table>

The compare instruction does a subtraction R 0 – R 1 and sets status flags according to the result:

<table>
<thead>
<tr>
<th>Registers</th>
<th>P</th>
<th>N</th>
<th>Z</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>R 0 &gt; R 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R 0 = R 1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>R 0 &lt; R 1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

CMP R 0 ; Perform subtraction R 0 – R 1, status flags will be modified according to result of subtraction
ACC P
OUT O 32 ; R 0 > R 1
ACC Z
OUT O 33 ; R 0 = R 1
ACC N
OUT O 34 ; R 0 < R 1
9.3.9 Word instructions for floating-point arithmetic

These instructions are used for calculating arithmetical equations using floating-point format registers and constants. Each arithmetical instruction starts with the letter F to indicate that it's a floating-point instruction. The operands of these instructions are always registers, never constants. If a constant is needed, it must be loaded into a register and then the register can be used in the floating-point instruction.

<table>
<thead>
<tr>
<th>Addition</th>
<th>Subtraction</th>
<th>Square root</th>
</tr>
</thead>
<tbody>
<tr>
<td>FADD R 0</td>
<td>FSUB R 0</td>
<td>FSQR R 100</td>
</tr>
<tr>
<td>R 1</td>
<td>R 1</td>
<td>R 101 ;result</td>
</tr>
<tr>
<td>R 3 ;R3=R0+R1</td>
<td>R 3 ;R3=R0-R1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Multiplication</th>
<th>Division</th>
<th>Comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMUL R 0</td>
<td>FDIV R 0</td>
<td>FCMP R 0</td>
</tr>
<tr>
<td>R 1</td>
<td>R 1</td>
<td>R 1</td>
</tr>
<tr>
<td>R 3 ;R3=R0*R1</td>
<td>R 3 ;R3=R0/R1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sine</th>
<th>Cosine</th>
<th>Arc tangent</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSIN R 10</td>
<td>FCOS R 10</td>
<td>FATAN R 10</td>
</tr>
<tr>
<td>R 11 ;result</td>
<td>R 11 ;result</td>
<td>R 11 ;result</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Exponent</th>
<th>Natural logarithm</th>
<th>Absolute value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEXP R 20</td>
<td>FLN R 20</td>
<td>FABS R 30</td>
</tr>
<tr>
<td>R 21 ;result</td>
<td>R 21 ;result</td>
<td>R 31 ;result</td>
</tr>
</tbody>
</table>

Status flags
All the above instructions modify the status flags, with the exception of the LD instruction for loading a floating-point format constant.

<table>
<thead>
<tr>
<th>Initialize a register</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD R 0</td>
</tr>
<tr>
<td>3.1415E0 ; R 0 = PI</td>
</tr>
</tbody>
</table>

9.3.10 Conversion of integer and floating-point registers

The PCD has separate instructions for arithmetic operations on integers and floating-point numbers. If an application program has to add or multiply two registers, one containing an integer and the other a floating-point number, one of the registers must be converted either to integer or floating-point before performing the arithmetical operation – both registers must contain data in the same format.

<table>
<thead>
<tr>
<th>Convert integer-fltg point</th>
<th>Convert fltg point-integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFP R 0 ; integer -&gt; float</td>
<td>FPI R 0 ; float -&gt; integer</td>
</tr>
<tr>
<td>0 ; exponent</td>
<td>0 ; exponent</td>
</tr>
</tbody>
</table>

9.3.11 Index register

Each COB has a special register: the index register. The content of the index register can be checked with the following instructions:

<table>
<thead>
<tr>
<th>SEI K 10</th>
<th>SEI Index register</th>
<th>Loads the index register with a constant of 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>INI K 99</td>
<td>INcrement Index register</td>
<td>Increments the index register and sets accu state high as long as: Index register &lt;= K 99</td>
</tr>
<tr>
<td>DEI K 5</td>
<td>DEcrement Index register</td>
<td>Decrements the index register and sets</td>
</tr>
</tbody>
</table>
Many PCD instructions support the use of the index register. This register allows indirect addressing of registers, flags, inputs, outputs, timers etc, used by instructions in the program. These instructions are the same as those normally used, but the mnemonic has an additional letter X.

**Example:**

Registers are non-volatile. This means they keep their information when the power supply is cut or if there is a cold-start. If we want to make a range of 100 registers volatile, we would have to initialise these 100 registers with the value zero during the cold-start. To initialise a register with zero, we can use the following instruction:

```plaintext
LD R 10
K 0
```

If we have 100 registers (R 10 to 109) to initialise, we would have to write this instruction 100 times, changing the register address each time. That would be rather tedious.

A better solution is to initialise the index register with an index of zero and implement a program loop to load the first register with zero, incrementing the index. Therefore, for each loop, we load zero into a different register (R 10, R 11,..., R 109). At the 100th loop, the index counter reaches the maximum index value (K 99) and forces the accumulator state low. This causes the loop to be exited so that the rest of the program can be processed.

```plaintext
XOB 16 ;Cold-start block
SEI K 0 ;Index = 0
Loop: LDX R 10 ;Load register address 10 + index with zero
INI K 99 ;Increment index and modify accu state
JR H Loop ;If accu is high, program jump to label Loop
EXOB

COB 0 ;Cyclic organization block

0
...
ECOB
```
9.3.12 Program jumps

The IL instruction set has three program jump instructions. They allow a sequence of instructions to be processed according to a binary condition binary, or program loops to be implemented for repetitive tasks (indexing).

<table>
<thead>
<tr>
<th>Jump instructions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JR</td>
<td>Jump relative Jumps a few lines forward or back from the line containing</td>
</tr>
<tr>
<td>JPD</td>
<td>the JR instruction</td>
</tr>
<tr>
<td>JPI</td>
<td>Jump indirect As JPD, but the line number is contained in a register</td>
</tr>
</tbody>
</table>

The jump destination is generally indicated by a label that defines a line of the program. However, it is also possible to define a relative jump with the number of lines to jump forward or back.

Jump using a line label:

```
JR   L  Next
INC  R 10
Next: NOP
```

Jump using the number of lines:

```
JR   L  +1
INC  R 10
```

The jump must always occur within a current block (COB, PB,...) never outside it.

If necessary, the jump may be implemented always, or only under a predetermined binary condition, such as the accumulator state or that of a status flag.

### Syntax for an unconditional jump instruction

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JR</td>
<td></td>
<td>Jump always implemented on line</td>
</tr>
<tr>
<td></td>
<td></td>
<td>corresponding to label</td>
</tr>
</tbody>
</table>

### Syntax for a conditional jump instruction

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Condition</th>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JR</td>
<td>H</td>
<td></td>
<td>If accu is high</td>
</tr>
<tr>
<td>JPD</td>
<td>L</td>
<td></td>
<td>If accu is low</td>
</tr>
<tr>
<td>JPI</td>
<td>Z</td>
<td></td>
<td>If status flag Z is high</td>
</tr>
<tr>
<td></td>
<td>P</td>
<td></td>
<td>If status flag P is high</td>
</tr>
<tr>
<td></td>
<td>N</td>
<td></td>
<td>If status flag N is high</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td></td>
<td>If status flag E is high</td>
</tr>
</tbody>
</table>

### Example: Count pulses from a binary input binary with a register (relative jump)

Unlike counters, the instruction to increment a register does not depend on accumulator state. It is therefor practical to use a jump instruction to increment a register when only that is necessary.

```
COB   0
  0
STH   F 1 ;Copy flag state to accumulator
DYN   F 2 ;Force accu high on rising edge of flag F1
JR    L Next ;If accu state is low, jump to label Next
```
The instructions \textit{STH} and \textit{DYN} read information from flag F 1 and set the accu state high for a positive flank or low in the absence of a flank. Depending on accu state, the instruction \textit{JR} either jumps to the line corresponding to the label \textit{Next}; or increments the register with the instruction \textit{INC}. The letter \textit{L} indicates the condition for implementing a jump (in this example, the jump will only be implemented if the accumulator state is low).

\textbf{Example:} Solution with an indirect jump

\begin{verbatim}
COB 0
LD R 2 ;Load line number into register
STH F 1 ;Copy flag state to accumulator
DYN F 2 ;Force accu state high on rising edge of flag F1
JPI L 2 ;If accu is low, jump to line number defined in register 2
INC R 10
Next: NOP
ECOB
\end{verbatim}

The indirect jump offers great flexibility. The program can itself modify the line number to which it will jump.
9.4 Editing a first application program

Count the number of spaces left in an 8-space parking lot and illuminate a red lamp when it is full.

When the PCD powers up, we assume that all parking spaces are available. We must therefore start by initialising the free space counter with the value 8. This initialisation takes place once only, when the PCD starts up. We will therefore program it in the cold-start block: XOB 16. The remaining program functions will be carried out by a cyclical organisation block (COB).

At the entrance, the sensor Car_incoming delivers a pulse each time a new vehicle enters. The rising edge of this signal must be detected to decrement the free space counter.

At the exit, a second sensor Car_outgoing delivers a pulse each time a vehicle exits. The rising edge of this signal must be detected to increment the free space counter.

If the parking lot is full, the counter's integer value will indicate zero available spaces. The counter's logic state informs us of this situation when it is low. The red lamp at the entrance to the parking lot must therefore be illuminated.
; Cold start organisation block
;----------------------------------------
XOB 16                    ; Program executed at start up
ACC H
LD Number_of_free_slots  ; Initialize the free slots counter
  8                     ; with the value 8 (unconditionally)
EXOB                        ; End of start-up program

; Cyclical Organisation Block
;----------------------------------------
COB 0                     ; Cyclical program
  0                     ; No supervision time
STH Car_incoming          ; A car comes into the parking:
DYN Dynamise_incoming_car_signal ; On the rising edge,
DEC Number_of_free_slots  ; Decrement number of free parking slots

STH Car_outgoing          ; A car leaves into the parking:
DYN Dynamise_leaving_car_signal ; On rising edge,
INC Number_of_free_slots  ; Increment number of free parking slots

STL Number_of_free_slots  ; If no more free parking slots
  (counter state= Low)
OUT Red_light             ; Set the red light
ECOB                        ; End of Cyclical program

<table>
<thead>
<tr>
<th>Symbol/Name</th>
<th>Type</th>
<th>Address</th>
<th>Comment</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Car_incoming</td>
<td>I</td>
<td>0</td>
<td>Gets high when a car comes into the parking</td>
<td>Local</td>
</tr>
<tr>
<td>Car_outgoing</td>
<td>I</td>
<td>1</td>
<td>Gets high when a car leaves the parking</td>
<td>Local</td>
</tr>
<tr>
<td>Red_light</td>
<td>D</td>
<td>0</td>
<td>Stops new cars at the entry</td>
<td>Local</td>
</tr>
<tr>
<td>Number_of_free_slots</td>
<td>C</td>
<td>32</td>
<td>Counts the number of free parking slots</td>
<td>Local</td>
</tr>
<tr>
<td>Dynamise_incoming_car_signal</td>
<td>F</td>
<td>11</td>
<td>Flag detects the rising edge of the car in...</td>
<td>Local</td>
</tr>
<tr>
<td>Dynamise_leaving_car_signal</td>
<td>F</td>
<td>12</td>
<td>Flag detects the rising edge of the car le...</td>
<td>Local</td>
</tr>
</tbody>
</table>
9.5 Building the program

The user program is fully edited, but not yet usable by the PCD. It must be translated into a
binary file. This is what the programming tool does when the user activates the button or
menu Rebuild All Files in the Project Manager or IL editor.

The Messages window tells us how the build is proceeding. It will be noted that the build
has assembly and linkage stages. If the program has been edited correctly, the build will
end with the message: Build successful. Total errors 0, Total warnings: 0

Any errors will be indicated by a message in red. A double mouse-click on an error
message will, if possible, open the relevant editor at the correct location.
9.6 Load program into PCD

Now that the application program is ready, it needs to be transferred from the PC into the PCD. In Project Manager, use either with the Online, Download Program menu command, or press the Download Program toolbar button.

If any communications problems arise, check the Online Settings, the cable between the PC and the PCD (PCD8.K111 or USB), and make sure the PCD is switched on. A USB connection may take a few seconds before the PC is able to recognize the attached device.

9.7 Debugging a program

Programs are not always perfect in their first version. It is helpful to test them carefully. Testing a program is supported by the same editor used for editing it.

The white lines represent the original source code, with symbols and comments.

The grey lines represent the code produced by the build, with the addresses of operands and program line numbers.

```plaintext
;-----------------------------
; Cyclical Organisation Block
;-----------------------------

CDB 0 ; Cyclical program
      0 ; No supervision time

000007  CDB 0
000008  0
000010  NOP

STH Car_incoming ; A car comes into the park

000011  STH 10 0 [8]
000012  DYN Dynamise_incoming_car_signal ; On the positive flank of

DEC Number_of_free_slots ; Decrement the number of

000013  DEC C 1404 [8]
```
9.7.1 Go On/Offline, Run and Stop

Online mode allows communication with the PCD to monitor the mode of operation (Run, Stop, Step-by-step). Any information needed to test the program can also be displayed.

Press Go On/Offline button
Put controller into Run mode with Run button

At the same time, note the RUN lamp, located on the front of the PCD. When the Run button is pressed, the RUN lamp comes on. The PCD is executing the user program.

When the Stop button is pressed, the RUN lamp goes off. The PCD stops executing the user program.

After Stop, note the line shown in red. It indicates the instruction at which the program stopped. The number in square brackets shows the contents of counter 1400. To the right is shown the accumulator and status flags states, and the index register value.

<table>
<thead>
<tr>
<th>STH</th>
<th>Car_incoming</th>
<th>; A car comes into the park</th>
</tr>
</thead>
<tbody>
<tr>
<td>000011</td>
<td>STH</td>
<td>I</td>
</tr>
<tr>
<td>DYN</td>
<td>Dynamise_incoming_car_signal</td>
<td>; On the positive flank of</td>
</tr>
<tr>
<td>000012</td>
<td>DYN</td>
<td>F</td>
</tr>
<tr>
<td>DEC</td>
<td>Number_of_Free_slots</td>
<td>; Decrement the number of</td>
</tr>
<tr>
<td>000013</td>
<td>DEC</td>
<td>C</td>
</tr>
</tbody>
</table>
9.7.2 Step-by-step mode

If the PCD is in Run mode, mark the first line to observe in step-by-step mode and press the Run to Cursor button. The PCD stops when it reaches the line with the cursor. Now begin step-by-step program execution by pressing the F11 key, or one of the buttons below.

If the program calls any PBs, FBs or SBs, it is not always necessary to step through these with step-by-step mode. The following three options are available:

Step In: Enter the block and step through it.

Step Over: Process the called block in Run, then continue in step-by-step after returning to the block that made the call.

Step Out: If the program has entered a block whose content is of no interest, it is possible to exit it immediately in Run mode and then continue in step-by-step mode after returning to the block that made the call.

For each program step, note the line shown in red. It moves to the following instruction line. The figure in square brackets represents the logical state of input \( \text{I1} \). To the right are shown the accumulator and status flags states, and the index register value.
9.7.3 Breakpoints

Breakpoints let you stop the program at an event linked to a program line or a symbol:
- State of an input, output, flag, status flag
- Value present in a register or counter

**Breakpoint on a symbol**

The breakpoint condition can be defined with the help of the Online Breakpoints menu, or of the Set/Clear Breakpoint button.

Using the above window, define the symbol type and address, or just drag a symbol from the Symbol Editor into the Symbol Name field, then set the breakpoint condition and state/value.

Selecting the Set & Run button forces the PCD into conditional run mode. The PCD’s Run LED flashes and the PCD’s Run button alternates between green and red.

The PCD automatically goes into stop mode when the breakpoint condition is reached. For example, when an instruction modifies the value of counter 1400 with a value greater than 4. The line following the last instruction processed by the PCD will be marked in red. It is then possible to continue processing the program in step-by-step mode, or with another breakpoint condition.

If necessary, conditional run mode can be interrupted in the following ways:
- The Clear - Run button forces the PCD into RUN mode. The PCD’s Run LED comes on and the PCD’s Run button turns green.
- The Clear - Stop button forces the PCD into stop mode. The PCD’s Run LED goes off and the PCD’s Run button turns red.

Breakpoint states which have been used before are stored in the breakpoint History list. They can be selected and activated with the Set & Run button.

**Breakpoint on a program line**

By selecting a program line, followed by the menu or button Online, Run To Cursor, the program can be made to stop at the line.
9.7.4 **Online modification of the program and data**

When testing a program step-by-step, it is helpful to modify the states/values of certain operands/symbols and check program behaviour under certain conditions.

Select one of the code view lines (grey) using the mouse and right-click to display the context menu.

The *Edit Data* context menu allows you to modify the operand state/value in the instruction selected.

The *Edit Instruction* command allows you to modify the mnemonic and address of the operand of the selected instruction line.

Status flags can also be modified with the help of the *Edit Status* context menu.
9.7.5 Viewing and modifying symbol states with the Watch Window

Another useful way of testing and viewing the state of symbols in our example is provided by the Watch Window. Use the Symbol Editor's Add to Watch Window command from the context menu.

Or press the Watch Window button, then drag symbols from the Symbol Editor into the Watch Window.

To modify the state/value of one of the symbols in the window, proceed as follows:

1. Start/Stop Monitoring
2. Position mouse pointer on value to be edited. Double left-click with mouse and edit new value.
3. Download Values
9.8 Commissioning an analogue module

All program instructions presented up until now used digital inputs or outputs, which are accessed directly for a single IL instruction, e.g. ANH I 45.

Analogue I/Os need a small program to read the values from each type of analogue module, which manages the multiplexing and A/D and D/A conversion. These can be programmed in IL, or using the new Device Configurator's Media mapping features which are described in the Device Configurator documentation.

9.8.1 Example for PCD2.W340 analogue input modules

If the PCD is equipped with a PCD2.W340 module, which has 8 universal input channels, the following routine may be used:

```
BA EQU O 96 ; Module base address in PCD
ACC H ; ACCU must be high
LD R 100 ; Defines the measuring channel (0..7)
2
MUL R 100
K 32 ; Calculates
R 100 ; control byte
ADD R 100 ; including
K 264 ; release bit
R 100
SET BA+15 ; Triggers A/D conversion
BITO 9 ; Sends control byte
R 100 ; including release bit
BA+0 ; to W3xx
BITIR 12 ; Reads 12 bits (0..4095) into R 77
BA+0
R 77
RES BA+15 ; Stop A/D conversion
```

The PCD2.W340 is a general-purpose analogue module which supports ranges 0..10V, 0..2.5V, 0..20mA and Pt/Ni 1000 temperature sensors. A bridge must be selected on the module to define the measurement range. Resolution is 12 bits, with a value range of 0..4095.

The routine shown above enters the channel defined in register 100 and supplies a raw measurement to register 77. For this module with a resolution of 12 bits, that corresponds to a measured value between 0 and 4095.

The user program must then convert the measurement into a standard physical unit.
9.8.2 Example for PCD2.W610 analogue output modules

Outputs work in a similar way to inputs.

If the PCD is equipped with a PCD2.W610 module, which has 4 universal analogue output channels, the following routine may be used:

```
BA EQU 0 96 ; Module base address in PCD
ACC H ; ACU must be high
LD R 100 ; Defines output channel (0..6)
2
BITOR 2 ; Transfers channel to W6x0
R 100
BA+0
BITOR 2 ; Writes 2 filler bits
R 100
BA+0
LD R 277 ; Defines digital value of output (0..4095)
3879
BITOR R 12 ; Transfer 12 bits of output value to W6x0
277
BA+0
SET BA+12 ; Triggers D/A conversion
```

A bridge must be selected on the module to define the output range: 0…20 mA or 0…10 V. Resolution is 12 bits, equating to 4095 distinct setpoint states.

The integer value at register 12 determines the output voltage or current at the channel defined in register 100:

<table>
<thead>
<tr>
<th>Input value at register 12</th>
<th>Output voltage [V]</th>
<th>Output current [mA]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2047</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>4095</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

For more detailed information and sample IL programs for analogue modules, please refer to your hardware manual or the support website:

http://www.sbc-support.com
Chapter 10

ADDITIONAL TOOLS

10.1 Data transfer utility

10.1.1 Using data transfer

10.1.2 Start up Data Transfer

10.1.3 Save data with Quick Data Upload

10.1.4 Restore data

10.1.5 Save data with help of script file

10.1.6 Restore data with help of script file

10.1.7 Upload options

10.1.8 Save data with command line mode

10.2 Watch window

10.2.1 Open the Watch Window

10.2.2 Add data to a Watch Window

10.2.3 Online display of data

10.2.4 Online modification of data

10.2.5 Display format

10.2.6 Watch Window and applications with several devices

10.2.7 Trend Function

10.2.8 Log Function

10.2.9 Symbols with a small and big magnitude on the same trend

10.2.10 Trend with several binary symbols

10.3 Online Configurator

10.3.1 Adjust the PCD's clock

10.3.2 PCD History

10.4 Updating firmware. (Firmware Downloader)

10.5 User menus
10. Additional tools

The Saia PG5® provides you with several additional utilities for a variety of services.
10.1 Data transfer utility

10.1.1 Using data transfer

This tool is used to save PCD data states/values in an ASCII file (*.dt5) or to restore them from the file into PCD memory.

The following data is transferred with this tool: inputs, outputs, flags, timers, counters, registers, data and text blocks.

Caution! The PCD program and hardware configurations are not saved by the Data Transfer utility. To save the program, hardware configurations and data, it is advisable to back up the program. See description of Project Manager.

10.1.2 Start up Data Transfer

Start up the program with menu:

Start → Programs → PG5 Suite V2.3.160 → Data Transfert

10.1.3 Save data with Quick Data Upload
Select menu *Online, Quick Data Upload* … or press the *Quick Data Upload* button to display the above window.

Select the types of data to save, address ranges, possibly also the display format for registers.

Select the OK button to upload data.

If a message like the one shown here is displayed, check the communications parameters using the menu *Online, Online Settings* and ensure that the PCD8.K111 cable correctly links the PC to the PCD.

Data upload takes a few moments to be displayed as follows:

![Data file display]

The data file can be edited with new values, then saved with the *File, Save* menu, or with the *Save* toolbar button.

### 10.1.4 Restore data

Previously saved files can be displayed again with the *File, Open* menu, or the *Open* toolbar button.

If necessary, the user can edit file values.

Data is restored to PCD memory with the *Online, Download Data to the PCD* menu, or with the *Download* button.

### 10.1.5 Save data with help of script file

If necessary, the list of data to save can be edited in a script file. Example:

```
;Exemple de script
R 0-99, ;Sauvegarde les registre 0 à 99 avec un format décimal
R 12, %h ;Sauvegarde le registre 12 avec un format hexadécimal
R 55, %f ;Sauvegarde le registre 55 avec un format flottant
F 0-999, ;Sauvegarde les indicateurs 0 à 999
F 1000, ;Sauvegarde l'indicateur 1000
```
Select the *Online, Upload Data from PCD …* menu, or the *Upload* button, to upload PCD data into a second window, distinct from the control window.

For more information about script commands available, please refer to program help. See menu *Help, Help Topics F1, General.*
10.1.6 Restore data with help of script file

A script file also allows you to edit data to be restored. Example:

```
:Exemple de script
R 0-99, 0 ;Charge les registre 0 à 99 avec zéro
R 12, 32h ;Charge le registre 12 avec 32 hexadécimal
R 55, 64.3 ;Charge le registre 55 avec 64.3 flottant
F 0-999, 0 ;Charge les indicateurs 0 à l'état bas
F 1000, 1 ;Charge l'indicateur 1000 à l'état haut
```

Select the Online, Download Data to PCD … menu or the Download button to download script data to the PCD.

10.1.7 Upload options

The window displayed with the Edit, Options menu allows you to adjust the format of data to be saved in file "*.dt5". With the following options, a data file can easily be imported to a Microsoft Excel editor.
10.1.8 Save data with command line mode

The Data Transfer tool can also be controlled with the help of DOS command lines. This allows batch files to be created for the regular, automatic saving of PCD data. The data can then be used by a Microsoft Excel program or database, ...

Command line syntax:

```
SDAT [Name_of_file[.dt5][data...]][/R=nnn][/I0nnn][/A=nnn]/D=nnn]
```

- **Name_of_file**: Name of file to save/restore
- **Data**: Definition of data to save. If no data is defined, the file is restored to the PCD

**Format**: `<type><start>[<end>][units]`

- **type**: R,C,O,F, DB
  - (C = counters/timers, O = inputs/outputs) First address
- **start**: First address
- **end**: Last address
- **units**: D,H,F (Decimal, hexadecimal, floating point) for R,C, DB

- **/R=nnn**: nnn = value per line for R,T,C, DB (1..256, default = 5)
- **/I=nnn**: nnn = value per line for I,O,F (1..256, default = 10)
- **/A=nnn**: nnn = address separator (TAB, SPACE, COMMA, COLON, default=TAB)
- **/D=nnn**: nnn = data separator (TAB, SPACE, COMMA, COLON, default=TAB)

Example:

```
sdat5 MyDatas.dt5 R0-99 R12H R55F F0-999 F1000 /R005 /I010
```
10.2 **Watch window**

The *Watch Window* is an excellent tool for checking programs and installations. It allows all the data of an application to be viewed and modified online.

10.2.1 **Open the Watch Window**

The *Watch Window* is displayed by selecting the *View, Watch Window* menu, or with the *Watch Window* button.

It is also possible to prepare several different *Watch Windows* in the *Program File* directory of the project manager. Add a new *Watch Window File* (*.5ww) with the *File New* menu, or with the *New File* button.

N.B.: Files of the type *.5ww* are never linked to a project (no arrow inside the file icon). The information in them has no bearing on any program build.

To open a *.5ww* file, select it with a double mouse-click, or mark the file and select the *File Open* menu.
10.2.2 Add data to a Watch Window

Drag symbols from the program or from the symbol editor into the Watch Window.

Select this button and press the left mouse button

Move the selection to the Watch Window

Symbols with their comments and states/values

It is also possible to edit symbols directly in the window:

Edit new address
10.2.3 Online display of data

The Start/Stop Monitoring button lets you display values present in the PCD for each of the symbols in the Watch Window. Check that the Watch Window’s status bar indicates RUN mode. If necessary, force the PCD into RUN or STOP with the Online menu.

10.2.4 Online modification of data

The Modify Value column lets you define new values for a number of symbols and download them into the PCD by selecting the Download Values button.

10.2.5 Display format

The display format of values can be adjusted as required.

Example: Display register R 2004 in hexadecimal
10.2.6 Watch Window and applications with several devices

The Watch Window lets you open several documents at one time. The menu, toolbar and status bar always relate to the active window, i.e. the window identified by a blue header bar.

By default, each open Watch Window document uses the Online settings of the device to which it belongs. Data from different PCDs available in the project can therefore be displayed on the communications network.
10.2.7 Trend Function

Watch window can trace a chart with maximum 8 values: registers, flags etc... If more data needs to be traced, it is possible to open a new watch window file.

Select the button Show /Hide Trend to display the watch window chart, then set some symbols present in the grid with a Trend colour and start the trend with the corresponding button in the tool bar.

If we select the view with the symbols or the trend, the properties windows shows some adjustable parameters like the trend sample time and scales units.
10.2.8 Log Function

Open Watch Window and open its Properties Window from menu: “View->Properties Window”. Select one or several Symbols in the Watch Window grid and set the option “Logging Enable” to yes then set to no if the measurement is finished.

The menu “Online -> Export Data” allows selecting the data to be logged and the time period with date and time.

The button Preview displays the measurements.
The button *Export on Export Data* dialog saves the trend data into a file. User provides the name and location to save the file.

### 10.2.9 Symbols with a small and big magnitude on the same trend

If symbols values present on a trend have a different magnitude, the symbols with a wide variation use all the vertical scale and the symbols with a small vertical magnitude use only a small fraction of this vertical scale.

There is now two possibilities to improve the visibility of the values:

1. A ‘Trending scale factor’ can be defined in the properties of the symbol. It allows to amplify or reduce the magnitude on the symbol values in the trend. The user must then take into account this factor to read the vertical scale in order to get the right value.

2. A second scale (second Y axis) can be add at the right of the trend. Select the symbol in the watch window grid to display the properties window and assign the symbol to the left or right scale with the properties option “Trending, Axis”.

Note:
The trend is always better if the vertical scale is automatically adapted to the symbol magnitude. Select the trend properties and set the option “Left/Right Axis, Auto adapt Y Scale” with Yes.

### 10.2.10 Trend with several binary symbols

If the trend of several symbols has the same magnitude on the vertical scale, it is possible to define an offset that will be add to the symbol value. This offset is usually automatically defined for binary symbols and can be displayed or modified using the symbol properties “Trending, Offset”.

10.3 **Online Configurator**

![Online Configurator](image)

**Information**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCD Type</td>
<td>PCD type reference number</td>
</tr>
<tr>
<td>Version</td>
<td>Version of PCD firmware</td>
</tr>
<tr>
<td>Program Name</td>
<td>User program name</td>
</tr>
<tr>
<td>Date</td>
<td>PCD clock date (if no clock: 1/1/92)</td>
</tr>
<tr>
<td>Time</td>
<td>PCD clock time</td>
</tr>
<tr>
<td>Day</td>
<td>Day of week: 1 = Monday, ... 7 = Sunday</td>
</tr>
<tr>
<td>Week</td>
<td>Week number</td>
</tr>
<tr>
<td>Status</td>
<td>Mode of operation: Run, Stop, Halt, Conditional</td>
</tr>
<tr>
<td>Online settings</td>
<td>Connection direct PGU or S-BUS</td>
</tr>
</tbody>
</table>

If the information in red is not displayed, or if a *No response* message box is displayed, it is not possible to establish communications between the PCD and the *Saia PG5 Online Configurator*.

If so, please check:
- Is the computer correctly connected to the PCD with PCD8.K111/USB cable?
- Have communications parameters been selected correctly with the *Settings* button?

### 10.3.1 Adjust the PCD's clock

1. Select the *Online configurator* button in the *Saia PG5 Project Manager* window. Then select *Clock* button.

2. Copy time from the PC to the controller with the *Copy to PCD*** button, or adjust the clock in the *Saia PCD Clock* fields.
### 10.3.2 PCD History

The History logs all hardware or software errors that occur during PCD operation. This table is permanently updated, even if the XOBs have not been programmed. Consult the history when the CPU's Error lamp comes on.

#### Notes:
- If an error can be traced to a line of the program, it will be specified. Otherwise it is displayed in hexadecimal.
- XOB 0 only appears if it has been programmed.

<table>
<thead>
<tr>
<th>Reason</th>
<th>Address</th>
<th>Time</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL LEVELS</td>
<td>30</td>
<td>14:09:43</td>
<td>06/01/2003</td>
</tr>
<tr>
<td>CALL LEVELS</td>
<td>30</td>
<td>14:09:43</td>
<td>06/01/2003</td>
</tr>
<tr>
<td>CALL LEVELS</td>
<td>30</td>
<td>14:09:43</td>
<td>06/01/2003</td>
</tr>
<tr>
<td>CALL LEVELS</td>
<td>30</td>
<td>14:09:44</td>
<td>06/01/2003</td>
</tr>
<tr>
<td>CALL LEVELS</td>
<td>30</td>
<td>14:09:44</td>
<td>06/01/2003</td>
</tr>
<tr>
<td>CALL LEVELS</td>
<td>30</td>
<td>14:09:44</td>
<td>06/01/2003</td>
</tr>
<tr>
<td>CALL LEVELS</td>
<td>30</td>
<td>14:09:44</td>
<td>06/01/2003</td>
</tr>
<tr>
<td>CALL LEVELS</td>
<td>30</td>
<td>14:09:44</td>
<td>06/01/2003</td>
</tr>
<tr>
<td>CALL LEVELS</td>
<td>30</td>
<td>14:09:44</td>
<td>06/01/2003</td>
</tr>
<tr>
<td>CALL LEVELS</td>
<td>30</td>
<td>14:09:44</td>
<td>06/01/2003</td>
</tr>
<tr>
<td>CALL LEVELS</td>
<td>30</td>
<td>14:09:44</td>
<td>06/01/2003</td>
</tr>
<tr>
<td>CALL LEVELS</td>
<td>30</td>
<td>14:09:44</td>
<td>06/01/2003</td>
</tr>
<tr>
<td>IR OVERFLOW</td>
<td>0</td>
<td>12:00:00</td>
<td>06/01/2003</td>
</tr>
<tr>
<td>ERROR FLAG</td>
<td>772</td>
<td>14:09:44</td>
<td>06/01/2003</td>
</tr>
</tbody>
</table>

- Date and time
- Line of program
- Error count
- Description of error
- Most recent error
10.4 Updating firmware. *(Firmware Downloader)*

Sometimes the program firmware has to be updated to benefit from the latest PCD product innovations.

Saia PCD firmware can be reloaded in flash memory using a little utility accessed with the *Tool, Firmware Downloader* menu of the Project Manager.

![Firmware Downloader](image)

**Download instructions:**

The *ADD* button adds a new firmware file (*.blk) to the list: *Files*.

The most recent firmware files are available in directory *FW* on the PG5 distribution CD.

Use menu *File, Settings* to adjust communications parameters to PGU mode (only mode currently supported).

Select firmware to download into PCD.

Download the firmware with the *Start* button. A dialogue box indicates the progress of data transfer.

When data transfer is complete, the PCD's *Run, Halt* and *Error* LEDs will start to flash. The PCD is reorganizing information in its memory. Please wait a further minute before powering off the controller, or continuing your work.
10.5 User menus

The Tools menu of the Saia PG5 Project Manager window can be extended with shortcuts to your favourite programs.

To add a shortcut, use the Tools, Customize Menu command. Press Help for more details.
11 SAIA PCD NETWORKS (S-NET) ................................................................. 2

11.1 Summary .......................................................................................................................... 2

11.2 Choice of network ............................................................................................................. 2
   11.2.1 Supported services ........................................................................................................ 2
   11.2.2 Design features ............................................................................................................ 3
11 Saia PCD Networks (S-Net)

11.1 Summary

Automation solutions often consist of several decentralized PCD controllers, terminals and supervision computers, connected by a communications network. Each station controls part of the process, and exchanges data with the other stations on the network.

To guarantee the flexibility of such a concept, the PCD system supports several types of communications network. Each network has its own capabilities, so the user should choose the network which is most appropriate for the application.

The PG5 is an effective tool for implementing these solutions:

- Saia PG5 Project Manager provides an overview of the stations (PCDs) and their configuration parameters including the network's communications parameters.
- The Fupla or IL editor allows the programming of the data exchange between PCD stations on the network.

The programming examples described in the following chapters are all installed with the PG5, and serve as basis to test and understand the functionality of the data exchange across different PCD networks. You will notice that some examples are very close to full practical implementations.

11.2 Choice of network

The choice of network depends on the application's requirements. These are the available S-Net network types:

- Profi-S-Bus: fieldbus network based at the Profibus FDL standard
- Ether-S-Bus: information network based on the standard Ethernet
- Serial S-Bus: network based on serial interface RS 485/232
- S-Bus Modem: network based on analogue or digital telephone line
- Profi-S-IO: fieldbus network based on the standard Profibus DP
- Profibus DP: fieldbus network based on the standard Profibus DP

The different networks are distinguished by their services, technical characteristics and their application domains.

11.2.1 Supported services

Although all the communication networks support the transport of PCD data as inputs, outputs, flags, registers etc., some also support the programming, control and commissioning of the PCD systems through the network using the PG5 tools.
11.2.2 Design features

11.2.2.1 Communications speed
The communications speed defines the reaction time for the transfer of data between the stations. If the amount of data to be transferred is large, or if the reaction time must be short, then the communication speed must be high. Note that if the communication speed of the network is adjustable, the same speed must be used by all stations on the network.

11.2.2.2 Maximum distance
The distance between stations can be a limitation for stations which are a long way apart. The maximum distance cannot be exceeded without amplification of the electrical signals, using a repeater or switch / Hub. Generally the maximum distance also depends on the communications speed. The higher the speed, the shorter the distance. Reducing the communications speed can often be a solution for crossing greater distances.

11.2.2.3 Communications protocol
The "protocol" is the message format used for data exchange between stations on the network. We can compare the protocol to the language used when two people speak to each other - they will only understand each other if they speak the same language. Likewise, two stations can only exchange data if they use the same protocol.

The protocols of some communications networks are official standards. This is a great advantage when equipment from different manufacturers must communicate. Field busses and sensors often use the standard Profibus DP protocol.

On certain communication networks like Ethernet or Profibus FDL it is possible to support data exchange using different protocols on the same physical network. But in all cases, the two communicating stations must use the same protocol.

11.2.2.4 Data exchange master-slave or multi-master mode
A "master-slave" network is composed of one master station and several slave stations. The master station controls the exchange of data between the slave stations.

A "multi-master" network is composed of several master stations, and several slave stations. Each master station can exchange data with other master or slave stations.

In both cases, direct data exchange between slave stations is not allowed.

11.2.2.5 Application domains
Some networks are designed for specific uses. For example, Profibus DP is a protocol oriented towards the machinery domain. The protocol of this network is well standardized, and a lot of compatible equipment from many suppliers allows data transfer on the same bus as used for the motor commands etc.

The Ether-S-Bus network is more oriented towards supervision systems, OPC servers, or can simply be used by the PG5 programming and commissioning tools.

Serial S-Bus provides an easy way to connect PCD systems. It is a very economical network, supporting the same services as Ether-S-Bus via RS-485, but also through analogue and ISDN telephone lines (S-Bus Modem).
The new Profi-S-Bus network merges all the advantages of a multi-master network and a high communications speed into a fieldbus network intended for industrial automation applications.

**Communication network S-Net**

<table>
<thead>
<tr>
<th>Services</th>
<th>Ether-S-Bus</th>
<th>Profi-S-Bus</th>
<th>Serial S-Bus</th>
<th>S-Bus Modem</th>
<th>Profi-S-IO</th>
<th>Profinet DP</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCD Programming</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Data Exchange</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Characteristics:**

| Max. transmission speed   | 10 and 100 Mbd | 12 Mbd | 38.4/115.2 Kbd | 38.4/115.2 Kbd | 12 Mbd |
| Max. distance without repeater or switch/Hub | 100 m | 100 m | 1200 m | - | 100 m |
| Cable type                | 4 twisted pairs | 1 twisted pair | 1 twisted pair | - | 1 twisted pair |
| Protocol                  | Saia PCD     | Saia PCD    | Saia PCD     | Saia PCD     | Normalized ISO |
| Exchange mode             | Multi-Master | Multi-Master | Master-Slave | Multi-Master | Master-Slave |
| Max. number of stations   | Unlimited   | 126        | 254          | Unlimited   | 126       |
| Application Domain        | Industry, building | Industry, building | Industry, building | Industry, building | Industry, building |
12 Profi-S-Bus

This example shows how to exchange data, such as Registers and Flags, between the PCDs connected to a Profi-S-Bus network

12.1 Profi-S-Bus network Example

12.2 Examples of the Data Exchange in Profi-S-Bus

<table>
<thead>
<tr>
<th>Master with data exchanges</th>
<th>Data on the network</th>
<th>Passive master or slave</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master station A</td>
<td>Master station B</td>
<td></td>
</tr>
<tr>
<td>1 Blinker0 .. 7 F 0 .. 7</td>
<td>Write 8 flags in the Master station B</td>
<td>Station_A.Blinker0 .. 7 F 100 .. 107</td>
</tr>
<tr>
<td>2 Master_B.Value100 R 125</td>
<td>Read 1 register in the Master station B</td>
<td>Value100 R 25</td>
</tr>
<tr>
<td>Slave station C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 Slave_C.Binary0 .. 7 F 100 .. 107</td>
<td>Read 8 flags in the slave station C</td>
<td>Binary0 .. 7 F 20 .. 27</td>
</tr>
<tr>
<td>4 Value0 .. 5 R 0 .. 5</td>
<td>Write 6 registers in the slave station C</td>
<td>Master_A. Value0 .. 5 R 20 .. 25</td>
</tr>
<tr>
<td>Master station B</td>
<td>Master station A</td>
<td></td>
</tr>
<tr>
<td>5 Temperature1 .. 4 Dynamic registers</td>
<td>Write the temperature measures to the slave C</td>
<td>Master_B.Temperature1 .. 4 R 100 .. 104</td>
</tr>
<tr>
<td>Slave station C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 Temperature1 .. 4 Dynamic registers</td>
<td>Write the temperature measures to the master A</td>
<td>Master_B.Temperature1 .. 4 R 100 .. 104</td>
</tr>
</tbody>
</table>
12.3 The PG5 Project

`Saia PG5 Project Manager` shows all the PCD stations in an application's Project, and also the network communication parameters. We will begin with adding a device to the Project for each of the Network Stations.

12.4 Device Configurator parameters

The `Device Configurator` parameters are the similar for a master and Slave station.

12.4.1 Define the device type

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCD2 M5540</td>
<td>CPU with 1M Bytes RAM, 3 I/O slots, 2 communication slot.</td>
</tr>
</tbody>
</table>

**Device Type**

Define the PCD type

12.4.2 Define S-Bus station number in the Network

<table>
<thead>
<tr>
<th>S-Bus</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S-Bus Support</td>
<td>Yes</td>
</tr>
<tr>
<td>Station Number</td>
<td>10</td>
</tr>
</tbody>
</table>

**Device properties:**

**Station Number**

S-Bus station number is common to all communication channels of the PCD.
Define communication channel of the Profi-S-Bus

<table>
<thead>
<tr>
<th>Location</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Onboard</td>
<td>RS-232/RS-485</td>
</tr>
<tr>
<td>Onboard</td>
<td>RS-485/S-Net</td>
</tr>
<tr>
<td>Onboard</td>
<td>USB</td>
</tr>
<tr>
<td>Onboard</td>
<td>Ethernet</td>
</tr>
</tbody>
</table>

### Full Protocol (PGU) Profi-S-Bus

Define the channel as slave or PGU. This definition can be accumulated with master function, adding a SASI Fbox in Fupla program.

**Slave + PGU**

Slave PGU

Supports data exchange with master stations, supervision systems and terminals. It also supports the PG5 programming tools.

**Slave**

Supports only data exchange with other master stations, supervision systems and terminals.

**Address**

Profi-S-Bus station number connected to channel.

**Baud Rate Profi-S-Bus**

Communication speed must be the same for all stations on the network.

**Bus Profile**

Transmission timings are grouped in three profiles: S-Net, DP or user-defined. With the user-defined profile, you can define your own timings using the Bus Parameter button. The profile must be identical for all network stations. The S-Net Profile is necessary when using RIO PCD3.T76x in the network.
12.4.4 Download the Device Configurator parameters in the device

With the new PCD systems, the Device Configurator parameters can be downloaded via a USB connection. It is necessary just to define Online Settings with the channel Profi-S-Bus + PGU.

Download the parameters to the PCD using Download Configuration on the Device Configurator window.

12.5 Fupla Program

12.5.1 Assign the channel using SASI Fbox

Assignment is done using a SASI Fbox, placed at the beginning of the Fupla File. Each communication network needs its own SASI Fbox, because the parameters are different depending on the network, the same for Master or Slave stations.

If the PCD uses more communication channels, define each channel using corresponding SASI Fbox. Then place the mouse over the SASI Fbox and using the context menu select Fbox properties, define a different Name for the Fbox of each channel. This name allows binding the exchange Fboxes SEND and RCV with SASI Fbox corresponding to the channel.
According to the network, the communication channel parameters can be partially defined from the adjust parameters of the SASI Fbox, and to be completed in the Device Configurator.

The channel number is always defined in the adjust parameters of the SASI FBox. The channel number depends on PCD Hardware and on the communication hardware used: slot B1, B2, serial interface PCD7.F, …

### 12.5.2 Assign Master channel

The assignment of the Master channel is done by combining the Device Configurator parameters with one of the Fboxes above. Only the communication channel and the timings of the Master Channel can be adjusted from the Fbox. Other parameters are all defined in the Device Configurator.

**Adjust parameters:**

**Channel**
 Defines the corresponding channel of the serial interface connected in the network. Depends from the PCD and his hardware.

**Timing**
 The Timeout is general defined with the value by default (0) and will be adjusted only for the particular applications (Gateway).

### 12.5.3 Assign slave channel

No SASI FBox is necessary for the slave station in the Profi-S-Bus network. All definitions necessary are already present in the Device Configurator.

### 12.5.4 Principles of data exchange in a multi-master network

A multi-master communication network has more than one master station. Master Stations are the only stations authorized to read or write the data of the other master and slave stations. Data exchange between slaves is not allowed.

With a Multi-master communication mode, data exchange is carried out between the masters in the network. Only one master at a time holds a token which authorizes it to exchange data with other master or slave stations on the network. When the master has finished transferring the data, the token is passed to the next master, which is then free to exchange data with the other masters or slaves. The token circulates automatically between the master stations, the slaves never have the token and so cannot read or write the data of other stations in the network.
12.5.5 Data Exchange between master and slave stations

User-controlled data exchange between stations is done using Fupla Fboxes placed on the Fupla pages, chosen the Fbox Selector. You will find the Fboxes to write (SEND) or to read (RCV) data packets, and also support different data formats: binary, integer, floating point, Data Block, etc.

The SEND or RCV Fbox can be resized to increase or decrease the number of inputs and outputs, defining the data packet to be exchanged with another station.

The address of the Communication Channel, used by data transmission Fbox is defined by the symbol shown at the top left of the Fbox, which binds it to the SASI Fbox of the same name in which the channel address is defined. This symbol can be edited by putting the mouse on the Fbox and selecting the context menu's Fbox Properties.

Each SEND and RCV Fbox has a binary input for activation of the data exchange. If this input is permanently high, data exchange will repeated as fast as possible. If a short pulse is applied to the input, data exchange will be executed at least once, but it is always possible to force it using the Execute button, or by a Restart Cold the PCD with Initialization option of the adjust parameters.

Master station data present at the inputs of the SEND Fbox, are sent to the Slave station defined in adjust window. Whereas the data present at the output of the RCV Fbox comes from the slave station defined by the parameters of the adjust window: address of the slave station, source element and base address.

Only the master stations are programmed with the SEND and RCV Fboxes! The slave stations can only be assigned with the communication channel.

According to the Fboxes used, the adjust parameters allows the definition of the slave stations to which data can be sent from the master station (SEND), or from which slave stations the Master can read data (RCV).

*Adjust parameters.*

*Profi-S-Bus Address*
Defines the number of the Profi-S-Bus slave station.

*Source, destination station*
Defines the number of the S-Bus slave station.
**Source, destination element**
Defines the type of the data to write or read from the slave.

**Source, destination address**
Defines the start address of the data to write or read in the slave. The number of the exchanged data values depends on the number of the inputs or outputs of the SEND or RCV Fbox.

### 12.5.6 Diagnostics

If the program is Online, a green or red LED is displayed at the top right of the SASI, SEND or RCV Fbox. Green indicates that the data transmission is OK, red indicates an error.

**Correct functionality**

All the Fbox are green, data exchange are done correctly.

**No data can be exchanged in the network**

SASI Fbox, SEND and RCV are red; no data can be exchanged in the network.
Possible corrective actions in master or slave station:

- Verify the *Device Configurator*
- Verify that the *Device Configurator* parameters have been downloaded into the PCD
- Verify that all stations use the same profile: S-Net, DP
- Verify that all stations communicate at the same speed
- Verify that the defined communication channel with the *Device Configurator* and SASI function are identical (same channel number)
- Verify that the PCD is equipped with the necessary communication hardware
- Verify that the stations are connected to the network and are powered on
- Verify the network wiring
- Verify that the firmware version supports Profi-S-Bus

Only some Fboxes do not exchange data

*SASI* Fbox and some *SEND* and *RCV* Fboxes are red. The Fbox in green exchanges the data correctly

Possible corrective actions in the master station

Verify the parameters of the *adjust window* of the red *SEND* and *RCV* Fbox.
Verify that the slave address is present in the network.

Possible corrective actions in the slave station

For every red *SEND* and *RCV* Fbox, view the slave station number and verify the concerned stations.

- Verify if the *Device Configurator* parameters are defined correctly
- Verify if the PCD is equipped with necessary communication hardware
- Verify if the stations are connected to the network and are powered on
- Verify the network wiring
- Verify if the firmware version supports Profi-S-Bus
**Only SASI Fbox is red**

Open adjust window of the SASI Fbox, and clear the last error using *Clear* button.

**Diagnostic Fbox**

If SASI lamp is red, it is always possible to obtain a diagnostic while consulting the adjust window of the *SASI Diagnostic* function. This Fbox should be placed just below SASI Fbox.
12.6 IL Programm

12.6.1 Assign master Channel using SASI instruction

Mode Profi-S-bus Master
Register with slave station address
First address of the range of 8 flags and one diagnostic register

$SASI
TEXT 10 "MODE:PSM,R0500;DIAG:F1000,R0501;"
$ENDSASI

SASI 10 ; Address of communication channel
10 ; Address of text with assignment parameters

The assignment of the channel is done using SASI instruction, which is placed at the beginning of the program: Graftec initialization sequence or initialization bloc XOB 16.

SASI instruction contains two parameters: communication channel address and address of the text with all the necessary channel parameters.

Text assignment parameters are different from one communication network to other, same as for slave or master station.

If the PCD exploit more communication channels de, each channel must be defined using SASI instruction and assignment text.

Depending on the network, channel parameters can be completed with Device Configurator parameters.

12.6.2 Assign slave channel

No SASI FBox is necessary for the slave station in the Profi-S-Bus network. All definitions necessary are already present in the Device Configurator.

12.6.3 Principles of data exchange in a multi-master network

A multi-master communication network has more than one master station. Master Stations are the only stations authorized to read or write the data of the other master and slave stations. Data exchange between slaves is not allowed.

With a Multi-master communication mode, data exchange is carried out between the masters in the network. Only one master at a time holds a token which authorizes it to exchange data with other master or slave stations on the network. When the master has finished transferring the data, the token is passed to the next master, which is then free to exchange data with the other masters or slaves. The token circulates automatically between the master stations, the slaves never have the token and so cannot read or write the data of other stations in the network.
12.6.4 Data Exchange between master and slave stations

Data exchange between the stations is the sequential program: The assignment of the communication channel is treated only once, data exchange in the network will be executed only if the previous exchange of the data’s is finished. That’s why we propose to treat IL data exchange with Graffec Editor. Initial Step allows assigning the communication channel at the Restart Cold of the PCD.

Other Steps are executed in loop, and step one supports one data package.

Every Step is separated by one Transition which tests diagnostic flag TBSY, and defines if data Exchange is finished. We are authorized to exchange data’s defined by step which follows, only if TBSY is Low.

Data Exchange using a Step

Before to exchange data, we must define address of the slave station in the register, which is declared for this by text assignment:

**Define the address of the slave station**

LDL R 500 ; Register address with the slave station address
11 ; S-Bus address

LDH R 500 ; Register address with slave station address
21 ; Profi-S-Bus Address

Data exchange between the stations is supported using two instructions:
STXM for writing data in the slave station (*SEND*)
SRXM for reading data in the slave station (*RCV*)

Each instruction contains four parameters: Channel address, number of data’s to exchange, address of the first data source, and the destination.
Write 8 Flags (F 0... F 7) in the slave station (F 200... F 207)

STXM 10 ; Channel address
8 ; Number of the data’s to exchange
F 0 ; address of the first source data (local Station)
F 200 ; address of the first destination data (slave Station)

Read a register (R 25) of the slave station (R 125)

SRXM 10 ; Channel address
1 ; Number of the data’s to exchange
R 25 ; address of the first source data (local Station)
R 125 ; address of the first destination data (slave Station)

Note:
Only the master stations are programmed with STXM and SRXM ! The slave stations must only be assigned with the communication channel.

Waiting the transmission end de using the transition

STL F 1003 ; Verify that TBSY is in Low state

Le Assignment text defines a range of 8 diagnostic flags for communication. Third flag will go in the high state during the data transmit, and in low state when exchange is finished.

12.6.5 Diagnostics

Channel assinations
In the case of the communication problem, verify if the channel assignation is donne correctly. Analyse the program step by step, and verify that the SASI instruction doesn’t display a flag error. If the channel assignation isn’t donne correctly, then the communication will not work.

Possible corrective actions in master or slave station:

- Verify the Device Configurator
- Verify that the Device Configurator parameters have been downloaded into the PCD
- Verify that all stations use the same profile: S-Net, DP
- Verify that all stations communicate at the same speed
- Verify that the defined communication channel with the Device Configurator and SASI instruction are identical (same channel number)
- Verify that the PCD is equipped with the necessary communication hardware
- Verify that the stations are connected to the network and are powered on
- Verify the network wiring
- Verify that the firmware version supports Profi-S-Bus
**Data's are not exchanged in the network**

Assignation Text defines a range with 8 diagnostic flags for the communication, Fifth Flag (*TDIA: Transmitter diagnostic*) will go in the high state during the data transmit error. Step by step test of the communication program, allows determining the instructions STXM and SRXM in error.

Attention: if the communication error occurs, then the diagnostic flag TDIA stays in high state, until the diagnostic register will not be reset to zero.

**Possible corrective actions in the master station**
Verify the parameters of the instructions STXM and SRXM in error. Verify that the slave address is present in the network.

**Possible corrective actions in the slave station**
For every instruction STXM and SRXM in error, read the slave station number and verify concerned stations.

- Verify if the *Device Configurator* parameters are defined correctly
- Verify if the PCD is equipped with necessary communication hardware
- Verify if the stations are connected to the network and are powered on
- Verify the network wiring
- Verify if the firmware version supports Profi-S-Bus

**Diagnostic register**

Diagnostic register can give us more information's about the nature the communication error. Display the binary content of the register and compare it with the descriptions of the PCD manual or the communication network manual.
12.7 Gateway Function

The Gateway feature is commonly used to allow two different communication networks to communicate together, or adapt a programming tool (PG5) or a supervision system (Visi+) to use a different network that the one usually supported.

12.7.1 Application

The Gateway function creates a bridge between two networks, for example to link an Ethernet network with a Profi-S-Bus network. In this way the PCD systems exchange data on a common bus, specific to the automation field and separated from information network of the company. But the PCs running the PG5 software or the supervision system Visi+ can exchange still data with the PCDs.

The Gateway function can be used as an interface between a communications network and the external world. For example, to make modem or USB communication interfaces.
To respect the communication timings, we cannot define two cascaded Gateways functions. But it is possible to define two parallel Gateways on the same network.

If necessary, a Gateway can make a bridge between several communication sub networks.

### 12.7.2 Configuration of the Gateway PGU function

It is easy to configure the Gateway function; it doesn’t need any program, only some parameters in the Device Configurator.

Generally, only a Gateway Slave Port and a Gateway Master Port should be defined, then all is automatically supported by Gateway function.

If the message received by the Gateway Slave Port is not for the local station (the Gateway), then data is re-transmitted via one of the sub-networks connected to the Gateway Master Port, according to the address ranges defined for the sub-network.
Example: Gateway USB, Profi-S-Bus

![Diagram of Gateway USB, Profi-S-Bus]

**Onboard Communication, properties of the Master A station**

<table>
<thead>
<tr>
<th>Profi-S-Bus Master Gateway</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Profi-S-Bus For Gateway</td>
</tr>
<tr>
<td>First S-Bus Station Profi-S-Bus</td>
</tr>
<tr>
<td>Last S-Bus Station Profi-S-Bus</td>
</tr>
<tr>
<td>Response Timeout</td>
</tr>
</tbody>
</table>

The USB Gateway is an exception; it doesn’t need any parameters for the Gateway Slave port, only the Gateway Master port must be defined.

(Don’t forget to download the new configuration into Master A!)

**Online Settings of the project CPU**

To make a USB communication with each PCD, the Online Settings should be configured with USB channel and S-Bus station number.
Testing the functionality of the Gateway Function

Activate one of the device, Master B or Slave C, of the project and Go Online for testing the communication with the station.

If necessary, the Online Configurator allows you to verify the station number online. It is also possible to download the program in the active device and to test it, staying always connected via USB cable to station Master A.

To communicate with another network station, activate the device and Go Online.

Remark:
With the Gateway feature, only the slave S-Bus station number is defined, the Profi-S-Bus station number is not taken into account because the telegrams are addressed to all Profi-S-Bus stations (Broadcast).

12.7.3 Configuration of the Gateway Slave port supplementary slave

The Gateway Slave port is a way to access the network from outside. If necessary, a second or the third Gateway Slave port can be defined.

Device Configurator parameters
In general, the PCD supports only one slave PGU channel. But the new controllers may support more PGU port on the same PCD. The configuration of the second Gateway Slave PGU is supported by the Device Configurator.

Example: add a second Gateway Ether-S-Bus, Profi-S-Bus
The second Gateway Slave port PGU is added, configuring the Device Configurator with the node and TCP/IP address.

**Fupla or IL Program**

It is possible to use a supplementary SASI Fbox/instruction and add a second Gateway Slave port.

This Gateway slave port, without PGU functionality, will not support the PG5 programming tools, but only a supervision system terminal. Only reading and writing PCD data are supported: registers, flags, etc.

**Example Fupla: add a third Serial-S-Bus, Profi-S-Bus**

The adjust Gateway parameter then must be defined with option Yes. According to channel type, the parameters of the adjust window should also correctly defined.

**Example IL : add a third Serial-S-Bus, Profi-S-Bus**

Use the following text to assign the channel:

```
$SASI
TEXT 11 "UART:9600; MODE:GS2; DIAG:F1110, R0501;"
$ENDSASI
```

Flag and diagnostic register
Mode S-Bus Gateway Slave Data mode
Transmission speed

### 12.7.4 Communication Timing

Generally the communication *timing* is defined with default values and this works correctly. But the use of the **Gateway** feature increases the times of the reactions necessary for the data exchange. It is then sometimes necessary to adjust the
timeout of the master stations which use the Gateway. The above picture shows which are the master channels whose timeouts must be adjusted.

To adjust the */Timeout* of the PG5, use */Online Settings* of the *Master Station A*:

To adjust the */Timeout* of the data exchange program to the PCD, use Fbox: SASI Profi-S-Bus Extended

### 12.8 Other References

For more information’s, you can also refer to the following manuals:

- Instruction Guide 26/133
- Profi-S-Bus (in preparation)
- Example of the project Profi-S-Bus installed with your PG5
13 ETHER-S-BUS .......................................................................................................................... 2

13.1 Ether-S-Bus network Example ........................................................................................................ 2

13.2 Examples of the Data Exchange in Ether-S-Bus ........................................................................................................ 2

13.3 The PG5 Project ........................................................................................................................................ 3

13.4 Device Configurator parameters ............................................................................................................ 3
  13.4.1 Define the device type .................................................................................................................... 3
  13.4.2 Define S-Bus station number in the Network ................................................................................ 3
  13.4.3 Define communication channel of the Ether-S-Bus ..................................................................... 4
  13.4.4 Download the Device Configurator parameters in the device ..................................................... 5

13.5 Fupla Program ........................................................................................................................................ 5
  13.5.1 Assign the channel using SASI Fbox ............................................................................................. 5
  13.5.2 Assign Master channel .................................................................................................................. 6
  13.5.3 Assign slave channel ....................................................................................................................... 6
  13.5.4 Principles of data exchange in a multi-master network .................................................................. 6
  13.5.5 Data Exchange between master and slave stations ...................................................................... 7
  13.5.6 Diagnostics ...................................................................................................................................... 8

13.6 IL program ............................................................................................................................................ 11
  13.6.1 Assign the master channel using SASI instruction ....................................................................... 11
  13.6.2 Assign slave channel ....................................................................................................................... 11
  13.6.3 Principles of data exchange in a multi-master network .................................................................. 11
  13.6.4 Data Exchange between master and slave stations ...................................................................... 12
  13.6.5 Diagnostics ...................................................................................................................................... 13

13.7 Gateway Function .................................................................................................................................. 15
  13.7.1 Application ....................................................................................................................................... 15
  13.7.2 Configuration of the Gateway PGU function ............................................................................... 16
  13.7.3 Configuration of the Gateway Slave port supplementary slave .................................................. 18
  13.7.4 Communication Timing ................................................................................................................. 20

13.8 Other References .................................................................................................................................. 21
13 Ether-S-Bus

This example shows how to exchange data, such as Registers and Flags, between the PCDs connected to an Ether-S-Bus network

13.1 Ether-S-Bus network Example

13.2 Examples of the Data Exchange in Ether-S-Bus

<table>
<thead>
<tr>
<th>Master with data exchanges</th>
<th>Data on the network</th>
<th>Passive master or slave</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Master station A</strong></td>
<td></td>
<td><strong>Master station B</strong></td>
</tr>
<tr>
<td>1 Blinker0 .. 7 F 0 .. 7</td>
<td>Write 8 flags in the Master station B</td>
<td>Station_A.Blinker0 .. 7 F 100 .. 107</td>
</tr>
<tr>
<td>2 Master_B.Value100 R 125</td>
<td>Read 1 register in the Master station B</td>
<td>Value100 R 25</td>
</tr>
<tr>
<td><strong>Slave station C</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 Slave_C.Binary0 .. 7 F 100 .. 107</td>
<td>Read 8 flags in the slave station C</td>
<td>Binary0 .. 7 F 20 .. 27</td>
</tr>
<tr>
<td>4 Value0 .. 5 R 0 .. 5</td>
<td>Write 6 registers in the slave station C</td>
<td>Master_A. Value0 .. 5 R 20 .. 25</td>
</tr>
<tr>
<td><strong>Master station B</strong></td>
<td></td>
<td><strong>Master station A</strong></td>
</tr>
<tr>
<td>5 Temperature1 .. 4 Dynamic registers</td>
<td>Write the temperature measures to the slave C</td>
<td>Master_B.Temperature1 .. 4 R 100 .. 104</td>
</tr>
<tr>
<td>6 Temperature1 .. 4 Dynamic registers</td>
<td>Write the temperature measures to the master A</td>
<td>Master_B.Temperature1 .. 4 R 100 .. 104</td>
</tr>
</tbody>
</table>
13.3 The PG5 Project

The Saia PG5 Project Manager shows all the PCD stations in an application's Project, and also the network communication parameters. We will begin with adding a device to the Project for each of the Network Stations.

13.4 Device Configurator parameters

The Device Configurator parameters are the similar for a master and Slave station.

13.4.1 Define the device type

**Device Type**

Define the PCD type

<table>
<thead>
<tr>
<th>Device</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCD2 M5540</td>
<td>CPU with 1M Bytes RAM, 8 I/O slots, 2 communication slot.</td>
</tr>
</tbody>
</table>

13.4.2 Define S-Bus station number in the Network

**Station Number**

S-Bus station number is common to all communication channels of the PCD.
13.4.3 Define communication channel of the Ether-S-Bus

<table>
<thead>
<tr>
<th>Location</th>
<th>Type</th>
<th>TCP/IP Enabled</th>
<th>IP Node</th>
<th>IP Address</th>
<th>Subnet Mask</th>
<th>Default Router</th>
<th>PGU port</th>
<th>Slave</th>
<th>Network groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>Onboard</td>
<td>RS-232/RS-485</td>
<td>Yes</td>
<td>1</td>
<td>192.168.12.129</td>
<td>255.255.255.0</td>
<td>0.0.0.0</td>
<td>Yes</td>
<td>Yes</td>
<td>(Default)</td>
</tr>
<tr>
<td>Onboard</td>
<td>RS-485/S-Net</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Onboard</td>
<td>USB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Onboard</td>
<td>Ethernet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**IP Node**
TCP/IP node number. The Node is used in the SEND and RCV Fbox-es to define a Slave station with which the data’s has to be exchanged.

**IP Address**
Ether-S-Bus station number connected to channel.

**PGU Port**
Define the channel as slave or PGU. This definition can be accumulated with master function, adding a SASI Fbox in Fupla program.

**Slave + PGU**
Supports data exchange with master stations, supervision systems and terminals. It also supports the PG5 programming tools.

**Slave**
Supports only data exchange with other master stations, supervision systems and terminals.
13.4.4 Download the Device Configurator parameters in the device

With the new PCD systems, the Device Configurator parameters can be downloaded via a USB connection. It is necessary just to define Online Settings with the channel Ether-S-Bus + PGU.

Download the parameters to the PCD using Download Configuration on the Device Configurator window.

13.5 Fupla Program

13.5.1 Assign the channel using SASI Fbox

Assignment is done using a SASI Fbox, placed at the beginning of the Fupla File. Each communication network needs its own SASI Fbox, because the parameters are different depending on the network, the same for Master or Slave stations.

If the PCD uses more communication channels, define each channel using corresponding SASI Fbox. Then place the mouse over the SASI Fbox and using the context menu select Fbox properties, define a different Name for the Fbox of each channel. This name allows binding the exchange Fboxes SEND and RCV with SASI Fbox corresponding to the channel.
According to the network, the communication channel parameters can be partially defined from the adjust parameters of the SASI FBox, and to be completed in the Device Configurator.

The Channel number is always defined in the adjust parameters of the SASI FBox. The channel number depends from PCD Hardware and on the communication hardware used: slot B1, B2, serial interface PCD7.F, …

13.5.2 Assign Master channel

The assignment of the Master channel is done by combining the Device Configurator parameters with one of the Fboxes above.

Adjust parameters:

Channel
Defines the channel number connected in the network. Depends from the PCD and his hardware.

Timing
The Timeout is general defined with the value by default (0) and will be adjusted only for the particular applications (Gateway).

13.5.3 Assign slave channel

No SASI FBox is necessary for the slave station in the Ether-S-Bus network. All definitions necessary are already present in the Device Configurator.

13.5.4 Principles of data exchange in a multi-master network

A multi-master communication network has more than one master station. Master Stations are the only stations authorized to read or write the data of the other master and slave stations. Data exchange between slaves is not allowed.

With a Multi-master communication mode, data exchange is carried out between the masters in the network. Only one master at a time holds a token which authorizes it to exchange data with other master or slave stations on the network. When the master has finished transferring the data, the token is passed to the next master, which is then free to exchange data with the other masters or slaves. The token circulates automatically between the master stations, the slaves never have the token and so cannot read or write the data of other stations in the network.
13.5.5 Data Exchange between master and slave stations

User-controlled data exchange between stations is done using Fupla Fboxes placed on the Fupla pages, chosen the Fbox Selector. You will find the Fboxes to write (SEND) or to read (RCV) data packets, and also support different data formats: binary, integer, floating point, Data Block, etc.

The SEND or RCV Fbox can be resized to increase or decrease the number of inputs and outputs, defining the data packet to be exchanged with another station.

The address of the Communication Channel, used by data transmission Fbox is defined by the symbol shown at the top left of the Fbox, which binds it to the SASI Fbox of the same name in which the channel address is defined. This symbol can be edited by putting the mouse on the Fbox and selecting the context menu's Fbox Properties.

Each SEND and RCV Fbox has a binary input for activation of the data exchange. If this input is permanently high, data exchange will repeated as fast as possible. If a short pulse is applied to the input, data exchange will be executed at least once, but it is always possible to force it using the Execute button, or by a Restart Cold the PCD with Initialization option of the adjust parameters.

Master station data present at the inputs of the SEND Fbox, are sent to the Slave station defined in adjust window. Whereas the data present at the output of the RCV Fbox comes from the slave station defined by the parameters of the adjust window: address of the slave station, source element and base address.

Only the master stations are programmed with the SEND and RCV Fboxes! The slave stations can only be assigned with the communication channel.

According to the Fboxes used, the adjust parameters allows the definition of the slave stations to which data can be sent from the master station (SEND), or from which slave stations the Master can read data (RCV).

Adjust parameters.

IP Node
Defines the node number of the Ether-S-Bus slave station.

Source, destination station
Defines the number of the S-Bus slave station

Source, destination element
Defines the type of the data to write or read from the slave.
Source, destination address
Defines the start address of the data to write or read in the slave. The number of the exchanged data values depends on the number of the inputs or outputs of the SEND or RCV Fbox.

13.5.6 Diagnostics
If the program is Online, a green or red LED is displayed at the top right of the SASI, SEND or RCV Fbox. Green indicates that the data transmission is OK, red indicates an error.

Correct functionality
All the Fbox are green, data exchange are done correctly.

No data can be exchanged in the network
SASI Fbox, SEND and RCV are red; no data can be exchanged in the network.
**Possible corrective actions in master or slave station:**

- Verify the *Device Configurator*
- Verify that the *Device Configurator* parameters have been downloaded into the PCD
- Verify that the communication Channel defined with the *Device Configurator*, and SASI function are identical (same channel number)
- Verify that the PCD is equipped with the necessary communication hardware
- Verify that the stations are connected to the network and are powered on
- Verify the network wiring
- Verify that the firmware version supports Ether-S-Bus

**Only some Fboxes do not exchange data**

SASI Fbox and some *SEND* and *RCV* Fboxes are red. The Fbox in green exchanges the data correctly

**Possible corrective actions in the master station**

Verify the parameters of the *adjust window* of the red *SEND* and *RCV* Fbox.
Verify that the slave address is present in the network.

**Possible corrective actions in the slave station**

For every red *SEND* and *RCV* Fbox, view the slave station number and verify the concerned stations.

- Verify if the *Device Configurator* parameters are defined correctly
- Verify if the PCD is equipped with necessary communication hardware
- Verify if the stations are connected to the network and are powered on
- Verify the network wiring
- Verify if the firmware version supports Ether-S-Bus
Only SASI Fbox is red
Open adjust window of the SASI Fbox, and clear the last error using Clear button.

Diagnostic Fbox
If SASI lamp is red, it is always possible to obtain a diagnostic while consulting the adjust window of the SASI Diagnostic function. This Fbox should be placed just below SASI Fbox.
13.6  IL program

13.6.1 Assign the master channel using SASI instruction

Mode Ether-S-bus Master
Register with slave station address
First address of the 8 flag range of the diagnostic register

$SASI
TEXT 10  "MODE: EM, R0500; DIAG: F1000, R0501;"
$ENDSASI

SASI  8  ; Address of the communication channel
      10  ; Address of the text with the assign parameters

Channel assignation is done using SASI instruction, which is placed in the beginning
of the program: initialization of the Graftec sequence, or initialization block XOB 16.

SASI instruction contains two parameters: The address of the communication channel
and the text address, with all necessary channel parameters.

The parameters of the assignation text are different from one network to other, also
for master or slave station.

If the PCD exploits more communication channels, define each channel using SASI
instruction and assignation text.

According to network, channel parameters can be completed with Device
Configurator parameters.

13.6.2 Assign slave channel

No SASI instruction is necessary for the slave station in the Ether-S-Bus network. All
definitions necessary are already present in the Device Configurator.

13.6.3 Principles of data exchange in a multi-master network

A multi-master communication network has more than one master station. Master
Stations are the only stations authorized to read or write the data of the other master
and slave stations. Data exchange between slaves is not allowed.

With a Multi-master communication mode, data exchange is carried out between the
masters in the network. Only one master at a time holds a token which authorizes it to
exchange data with other master or slave stations on the network. When the master
has finished transferring the data, the token is passed to the next master, which is
then free to exchange data with the other masters or slaves. The token circulates
automatically between the master stations, the slaves never have the token and so
cannot read or write the data of other stations in the network.
13.6.4 Data Exchange between master and slave stations

Data exchange between the stations is the sequential program: The assignation of the communication channel is treated only once, data exchange in the network will be executed only if the previous exchange of the data’s is finished. That’s why we propose to treat IL data exchange with Graftec Editor.

Initial Step allows assigning the communication channel at the Restart Cold of the PCD.

Other Steps are executed in loop, and step one supports one data package.

Every Step is separated by one Transition which tests diagnostic flag TBSY, and defines if data Exchange is finished. We are authorized to exchange data’s defined by step which follows, only if TBSY is Low.

**Data Exchange using a Step**

Before to exchange data, we must define address of the slave station in the register, which is declared for this by text assignation:

**Define the address of the slave station**

```
LDL     R 500   ; Register address with the slave station address
11      ; S-Bus address

LDH     R 500   ; Register address with slave station address
2       ; IP Node
```

Data exchange between the stations is supported using two instructions:

STXM for writing data in the slave station (*SEND*)

SRXM for reading data in the slave station (*RCV*)

Each instruction contains four parameters: Channel address, number of data’s to exchange, address of the first data source, and the destination.
Write 8 Flags (F 0... F 7) in the slave station (F 200... F 207)

STXM 8 ; Channel address
8 ; Number of the data’s to exchange
F 0 ; address of the first source data (local Station)
F 200 ; address of the first destination data (slave Station)

Read a register (R 25) of the slave station (R 125)

SRXM 8 ; Channel address
1 ; Number of the data’s to exchange
R 25 ; address of the first source data (local Station)
R 125 ; address of the first destination data (slave Station)

Note:
Only the master stations are programmed with STXM and SRXM! The slave stations must only be assigned with the communication channel.

Waiting the transmission end de using the transition

STL F 1003 ; Verify that TBSY is in Low state

Le Assignment text defines a range of 8 diagnostic flags for communication. Third flag will go in the high state during the data transmit, and in low state when exchange is finished.

13.6.5 Diagnostics

Channel assignations
In the case of the communication problem, verify if the channel assignation is done correctly. Analyse the program step by step, and verify that the SASI instruction doesn’t display a flag error. If the channel assignation isn’t done correctly, then the communication will not work.

Possible corrective actions in master or slave station:

- Verify the Device Configurator
- Verify that the Device Configurator parameters have been downloaded into the PCD
- Verify that all stations use the same profile: S-Net, DP
- Verify that all stations communicate at the same speed
- Verify that the defined communication channel with the Device Configurator and SASI instruction are identical (same channel number)
- Verify that the PCD is equipped with the necessary communication hardware
- Verify that the stations are connected to the network and are powered on
- Verify the network wiring
- Verify that the firmware version supports Ether-S-Bus
Data’s are not exchanged in the network

Assignation Text defines a range with 8 diagnostic flags for the communication, Fifth Flag (TDIA: Transmitter diagnostic) will go in the high state during the data transmit error. Step by step test of the communication program, allows determining the instructions STXM and SRXM in error.

Attention: if the communication error occurs, then the diagnostic flag TDIA stays in high state, until the diagnostic register will not be reset to zero.

Possible corrective actions in the master station
Verify the parameters of the instructions STXM and SRXM in error. Verify that the slave address is present in the network.

Possible corrective actions in the slave station
For every instruction STXM and SRXM in error, read the slave station number and verify concerned stations.

- Verify if the Device Configurator are defined correctly
- Verify if the PCD is equipped with necessary communication hardware
- Verify if the stations are connected to the network and are powered on
- Verify the network wiring
- Verify if the firmware version supports Ether-S-Bus

Diagnostic register

Diagnostic register can give us more information’s about the nature the communication error. Display the binary content of the register and compare it with the descriptions of the PCD manual or the communication network manual.
13.7 Gateway Function

The Gateway feature is commonly used to allow two different communication networks to communicate together, or adapt a programming tool (PG5) or a supervision system (Visi+) to use a different network that the one usually supported.

13.7.1 Application

The Gateway function creates a bridge between two networks, for example to link an Ethernet network with a Profi-S-Bus network. In this way the PCD systems exchange data on a common bus, specific to the automation field and separated from information network of the company. But the PCs running the PG5 software or the supervision system Visi+ can exchange still data with the PCDs.

The Gateway function can be used as an interface between a communications network and the external world. For example, to make modem or USB communication interfaces.
To respect the communication timings, we cannot define two cascaded Gateways functions. But it is possible to define two parallel Gateways on the same network.

13.7.2 Configuration of the Gateway PGU function

It is easy to configure the Gateway function; it doesn’t need any program, only some parameters in the PCD Device Configurator.

Generally, only a Gateway Slave Port and a Gateway Master Port should be defined, then all is automatically supported by Gateway function.

If the message received by the Gateway Slave Port is not for the local station (the Gateway), then data is re-transmitted via one of the sub-networks connected to the Gateway Master Port, according to the address ranges defined for the sub-network.
Example: Gateway USB, Ether-S-Bus

Onboard Communication, properties of the Master A station

- USB Gateway
  - Station Master A
    - IP Node: 1
    - IP address: 192.168.12.128
    - S-Bus Station Number: 10

- Gateway Master port
  - Gateway

- Gateway slave port

- Ether-S-Bus
  - Station Slave C
    - IP Node: 2
    - IP address: 192.168.12.130
    - S-Bus Station Number: 12

The USB Gateway is an exception; it doesn’t need any parameters for the Gateway Slave port, only the Gateway Master port must be defined. (Don’t forget to download the new configuration into Master A!)

Online Settings of the project CPU

- Online Settings [Slave station C]
  - S-Bus USB
    - No
    - No
    - No
    - No
To make a USB communication with each PCD, the Online Settings should be configured with USB channel and S-Bus station number.

**Testing the functionality of the Gateway Function**

Activate one of the device, *Master B or Slave C*, of the project and Go Online for testing the communication with the station.

If necessary, the Online Configurator allows you to verify the station number online. It is also possible to download the program in the active device and to test it, staying always connected via USB cable to station *Master A*

To communicate with another network station, activate the device and Go Online.

**Remark:**
With the Gateway feature, only the slave S-Bus station number is defined, the Ether-S-Bus station number is not taken into account because the telegrams are addressed to all Ether-S-Bus stations (Broadcast).

### 13.7.3 Configuration of the Gateway Slave port supplementary slave

The Gateway Slave port is a way to access the network from outside. If necessary, a second or the third Gateway Slave port can be defined.

**Device Configurator parameters**
In general, the PCD supports only one slave PGU channel. But the new controllers may support more PGU port on the same device. The configuration of the second Gateway Slave PGU is supported by the *Device Configurator.*
Example: add a second Gateway S-Bus modem, Ether-S-Bus

The second Gateway Slave port PGU is added, configuring the Device Configurator with the parameters for the modem.

Fupla or IL Program
It is possible to use a supplementary SASI Fbox/instruction and add a second Gateway Slave port.

this Gateway slave port, without PGU functionality, will not support the PG5 programming tools, but only a supervision system terminal. Only reading and writing PCD data are supported: registers, flags, etc.

Example Fupla: add a third Serial-S-Bus, Ether-S-Bus

Use the following text to assign the channel:

$SASI
TEXT 11 "UART:9600; MODE:GS2; DIAG:F1110, R0501;"
$ENDSASI

Flag and diagnostic register
Mode S-Bus Gateway Slave Data mode
Transmission speed

Example IL : add a third Serial-S-Bus, Ether-S-Bus

Use the following text to assign the channel:

$SASI
TEXT 11 "UART:9600; MODE:GS2; DIAG:F1110, R0501;"
$ENDSASI

Flag and diagnostic register
Mode S-Bus Gateway Slave Data mode
Transmission speed
13.7.4 Communication Timing

Generally the communication timing is defined with default values and this works correctly. But the use of the Gateway feature increases the times of the reactions necessary for the data exchange. It is then sometimes necessary to adjust the timeout of the master stations which use the Gateway. The above picture shows which are the master channels whose timeouts must be adjusted.

To adjust the Timeout of the PG5, use Online Settings of the Master Station A:

To adjust the Timeout of the data exchange program to the PCD, use Fbox: SASI S-Bus IP Extended
13.8 Other References

For more information's, you can also refer to the following manuals:

- Instruction Guide 26/133
- Ethernet TCP/IP 27/776
- Example of the Ether-S-Bus project installed with your PG5
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>14 PROFI-S-IO</td>
<td>2</td>
</tr>
<tr>
<td>14.1 Profi-S-IO network example</td>
<td>2</td>
</tr>
<tr>
<td>14.2 General functionality</td>
<td>2</td>
</tr>
<tr>
<td>14.3 PG5 project</td>
<td>3</td>
</tr>
<tr>
<td>14.4 Defining stations on the network</td>
<td>3</td>
</tr>
<tr>
<td>14.5 Configuring the master station</td>
<td>4</td>
</tr>
<tr>
<td>14.6 Configuring slave stations</td>
<td>4</td>
</tr>
<tr>
<td>14.6.1 Configuring Input /Output modules</td>
<td>4</td>
</tr>
<tr>
<td>14.6.2 Configuring symbol names for remote data</td>
<td>5</td>
</tr>
<tr>
<td>14.6.3 Configuring I/O parameters</td>
<td>5</td>
</tr>
<tr>
<td>14.7 Using network symbols in Fupla or IL programs</td>
<td>6</td>
</tr>
<tr>
<td>14.8 Further information</td>
<td>7</td>
</tr>
</tbody>
</table>
14 Profi-S-IO

This example shows how remote binary and analog inputs and outputs from the PCD3.T7xx RIO are used.

14.1 Profi-S-IO network example

![Diagram of Profi-S-IO network example]

14.2 General functionality

With both Profibus DP and Profibus-S-IO, network data exchange is configured using the S-Net Configurator. No Fupla or IL code needs to be written, and no Device Configurator need to be configured (apart from the communications module types and bus parameters if using the PCD2.M480 or PCD3).

The configurator defines each slave station on the network, and which I/O modules are fitted. I/O data from these remote I/Os is mapped to symbols or absolute addresses in the master station. Code generated by the S-Net configurator continually transfers I/O data from the slaves to and from the memory image in the master.

When the program is compiled, S-Net generates all the code needed to continually transfer the data between the remote slave stations and the master station's memory image at the start/end of every cycle. The I/O image data can be accessed directly by the master station's Fupla or IL programs.

In this way, network data exchange is clearly separated from the process control.
14.3 PG5 project

The S-Net configurator file is added to the master station in the same way as Fupla or IL files, using File New, selecting the "Profi-S-IO Network File (.sio)" file type.

S-Net configurator usage is similar for both Profi-S-IO and Profibus DP data exchange. The only differences are:

- File extension of the configuration: .SIO, .DP
- The supported devices in the network: SIO = Saia PCD devices, DP = devices for Saia PCD+ other suppliers.
- Bus timing profiles: S-Net or DP.

14.4 Defining stations on the network

For each station, select the station type in the device list, and add it in the network with the >> button.
14.5 Configuring the master station

The only information which needs to be defined for the master station is the Associated device File, which is the access path of the master device. This where S-Net will create the master station’s network control file. This dialog box also allows the station name and address to be defined.
For each input/output module fitted in the slave station, select the module type in the **Supported Modules** list and add it to the **Installed Modules** list using the >> button. Ensure that the *Slot* number corresponds to the slot where the module is actually installed, use the up/down Move arrows to change the slot.

### 14.6.2 Configuring symbol names for remote data

![Module Media Map](image)

For each module in the **Installed Modules** list, select the module and press the **Media Map** button to define symbol names and media types for the module's data. If necessary, a base address for the first flag or register in the master station can be defined. But the easiest way is to leave the "Base Address" field empty, so that dynamic addresses are used.

### 14.6.3 Configuring I/O parameters

![Edit Parameters](image)
With some modules, such as analogue measurement modules, additional parameters should be defined for selecting units, sensor types etc. These are configured by selecting the module and pressing the *Parameters* button.

### Configuring the network

The communications speed and bus profiles are defined using the *Edit - Bus Parameters* menu command.

Note:
If a PCD7.T7xx station is connected to the network, always choose the "S-Net" bus profile.

### 14.7 Using network symbols in Fupla or IL programs
After compilation of the S-Net file (Project / Compile menu command), the Symbol Editor displays a new page containing the accessible network symbols. These symbols can be used directly in Fupla and IL programs.

14.8 Further information

For more information please refer to these manuals:

- Profibus DP 26-765
- Profi-S-IO (in preparation)
- Example Profi-S-IO project installed with your PG5
Saia-Burgess Controls AG
Bahnhofstrasse 18
3280 Murten / Switzerland

Phone. : +41 26 580 30 00
Fax : +41 26 580 34 99

E-mail support : support@saia-pcd.com
Support site: www.saia-pcd.com
SBC site: www.sbc-support.com
International Representatives & SBC Sales Companies: www.saia-pcd.com/contact

Postal address for returns from customers of the Swiss Sales office

Saia-Burgess Controls AG
Service Après-Vente
Rue de la Gare 18
3280 Morat / Switzerland