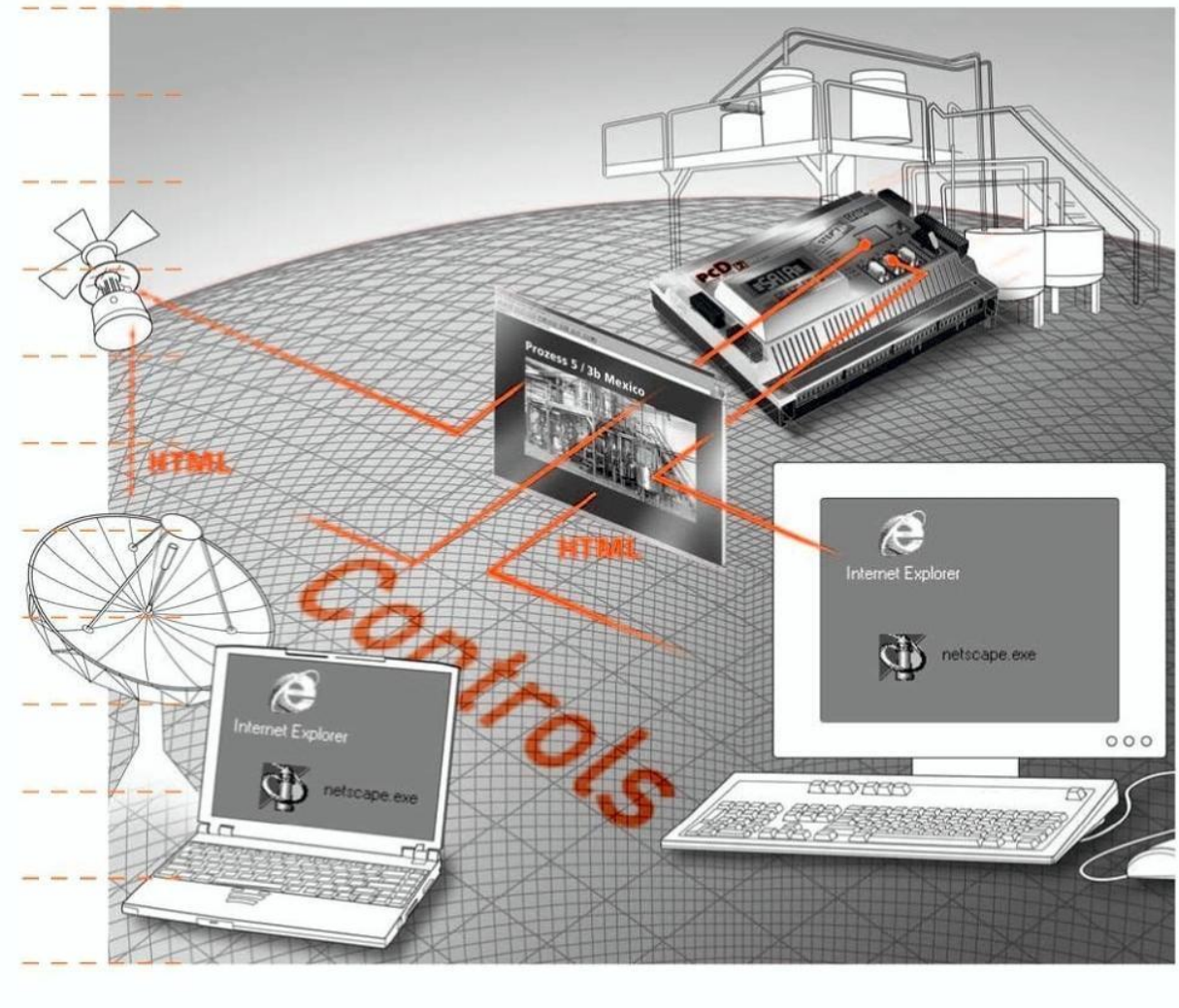
Manual FBox Builder (Full Version)



saia-burgess
Smart solutions for comfort and safety

Document-No. 26/837; Edition E5; 17.01.2006

Document-History

Document-No.	Edition	Modification	Publication	Remarks
	PE2	08.09.2004		Preliminary Edition
	E2		09.09.2004	Published Edition
	E3	09.06.2005		Updated Version
	E4	11.11.2005		Updated for PG5 B1.4.030
	E5	17.01.2006		Updated for PG5 B1.4.043

0.2 Trademarks

- Saia® and Saia® PCD are registered trademarks of Saia-Burgess Electronics AG
- Windows 95/98, Windows NT, Word, Excel, PowerPoint, FrontPage and Microsoft Internet Explorer are registered trademarks of The Microsoft Corporation.
- Netscape Navigator is a registered trademark of The Netscape Communications

Technical modifications and changes depending on state of the art.

Published in Switzerland

Preface

This document is intended to give you a good understanding of the FBOX Builder and therefore also a good understanding of the FBOX development in general. The document brings you quickly with its “Quick Start” chapter into the main features of the tool. The chapter “FBOX Basics” gives you smoothly the knowledge to build simple FBOXES. When you have a good understanding of the basics, the chapter “FBOX Advanced” will open you the way to use the full power of the FBOX functionalities.

To use properly the FBOX Builder you must have a good knowledge of PG5 and the Instruction List language.

The SAIA FBOX Builder allows you to create your own FBOXES for FUPLA programming. The FBOX Builder greatly improves the FBOX development process. You can build your FBOXES based on existing code like FB/PB/FBOX or you can begin a new FBOX from scratch.

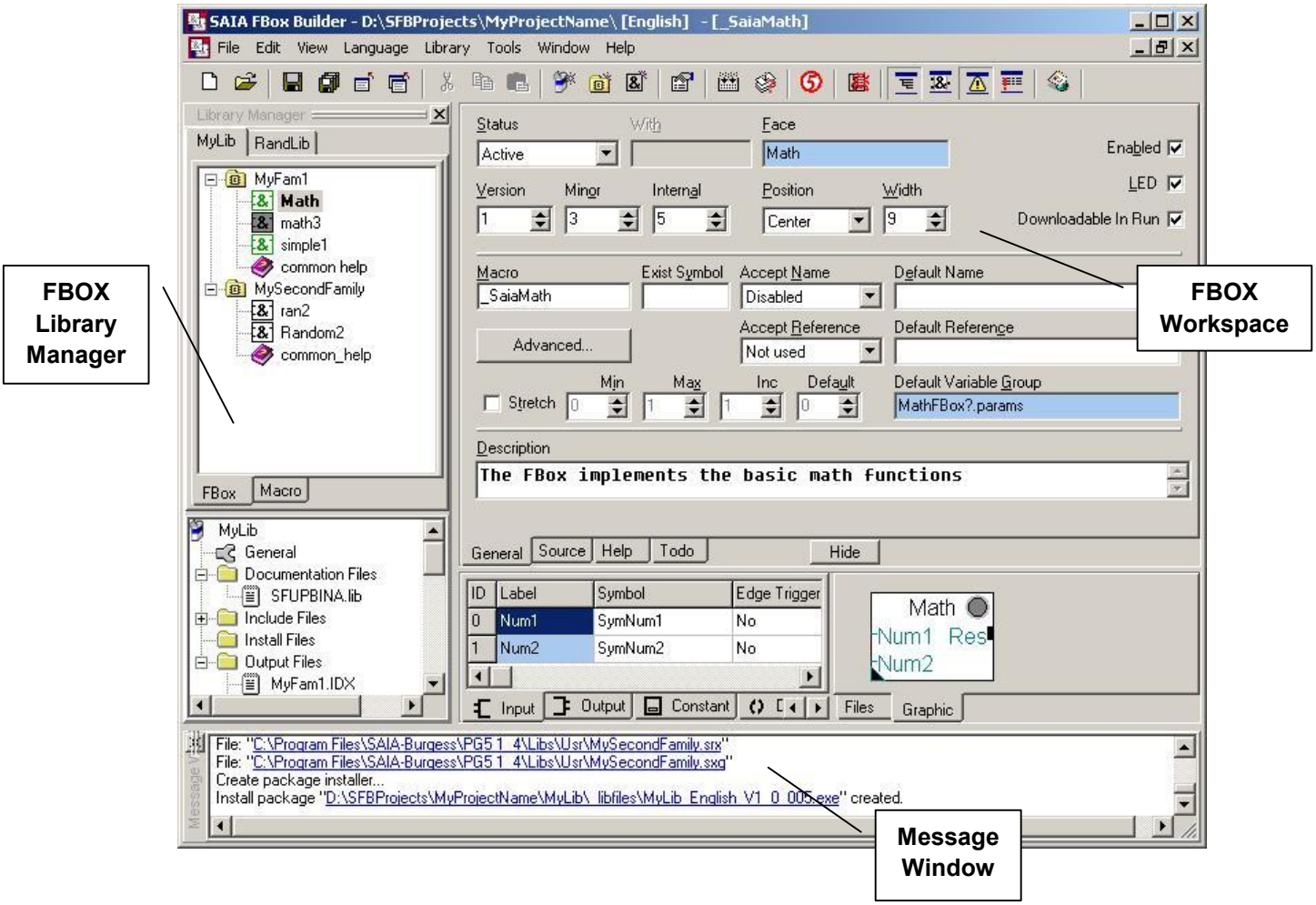
The FBOX Builder does support you also in the deployment process of your FBOX Library, functionalities like **Language Management**, **Help File Generator**, **Install Library Package** or **Version Management** at each level: library, family and FBOX are the main strong points of this software tool.

Contents

Contents

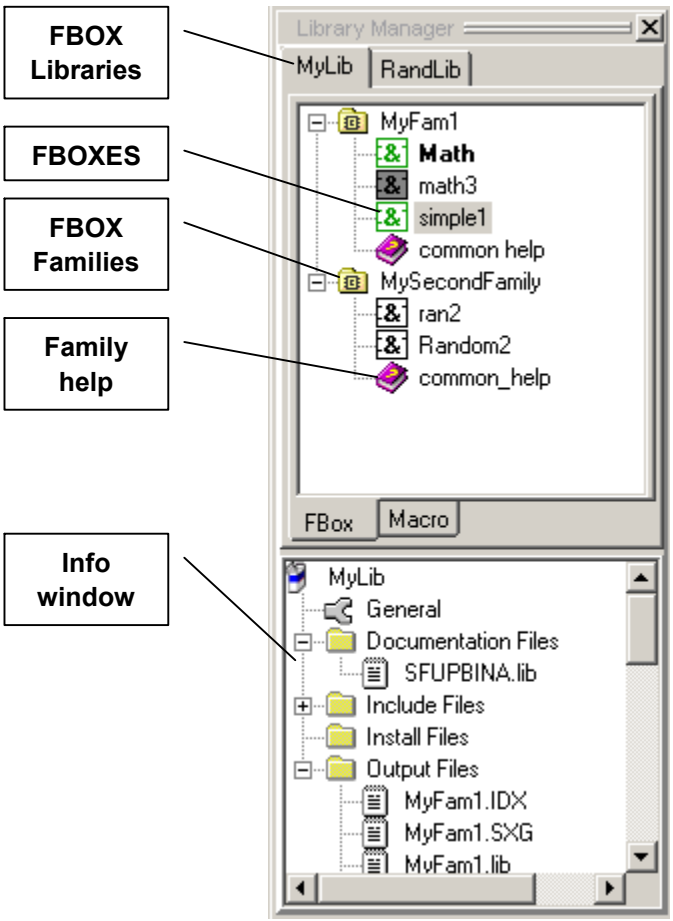
1 Overview

1.1 Main Software Components



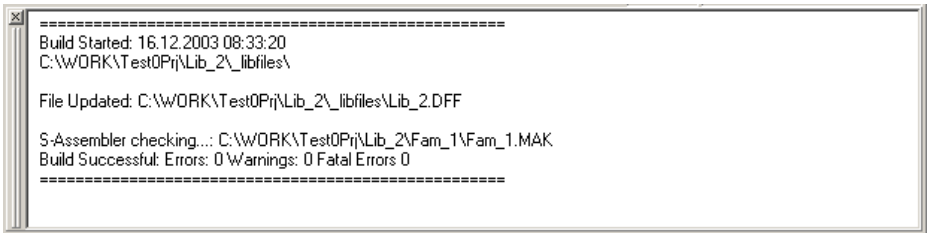
1.1.1 FBOX Library Manager

The begining of every operations starts here: create a library, create a family, create/import/modify an FBOX.



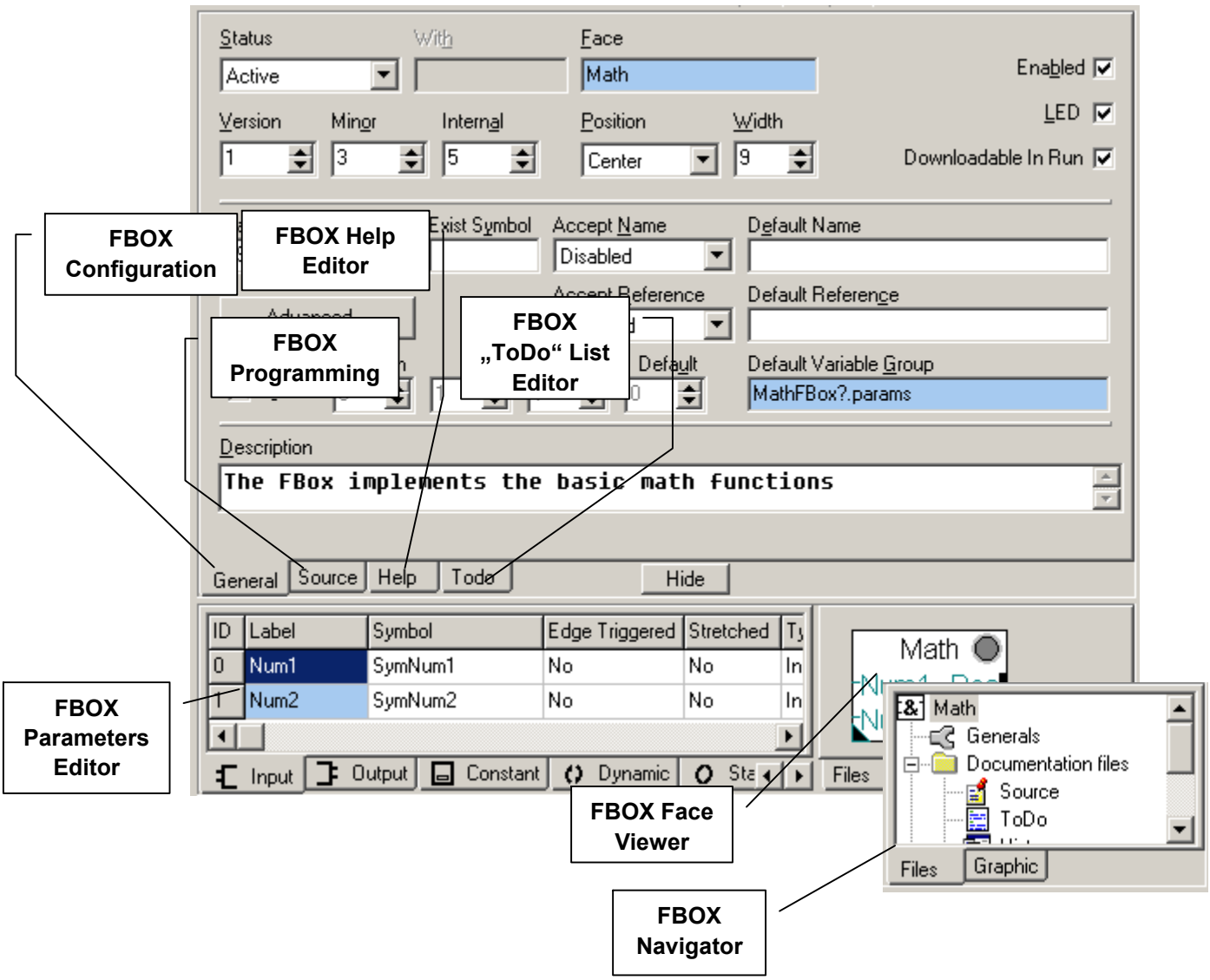
1.1.2 Message Window

Compilation Errors and Warnings will be displayed in the Message Window. In case of error, you can find the mistake in double clicking on the message.



1.1.3 FBOX Workspace

The Workspace belongs to one FBOX and allows the full configuration and programming of it.



Contents

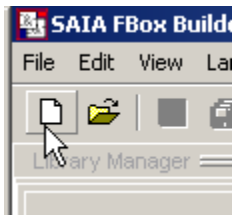
2 FBOX Builder Quick Start

2.1 Import an Existing FB

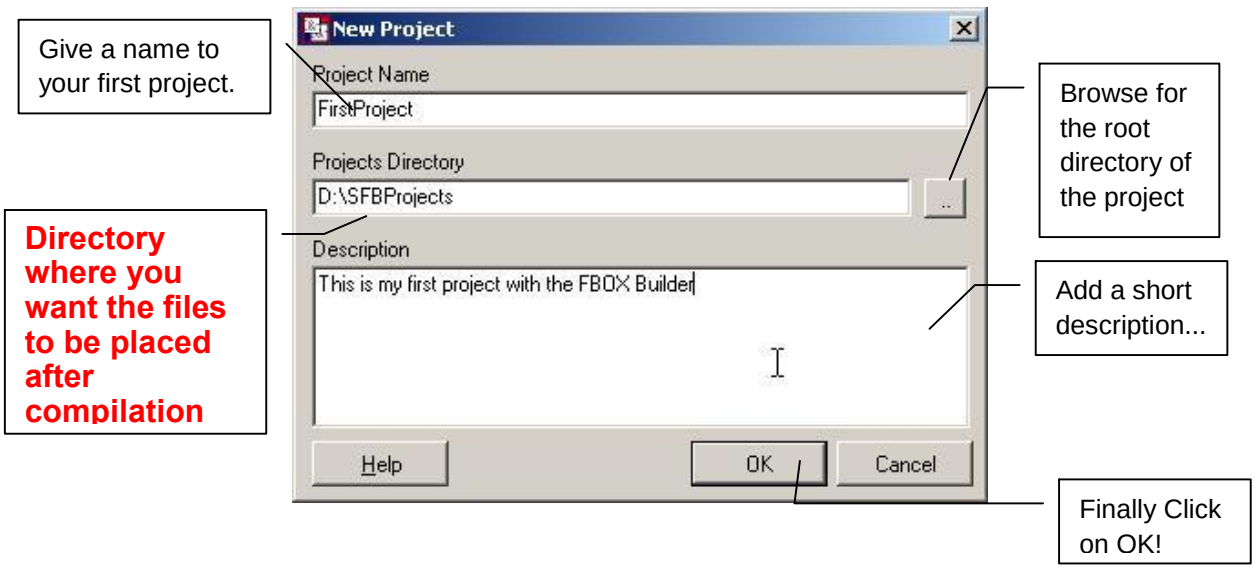
The aim of this chapter is to build an FBOX based on the functionality of an existing FB.

2.1.1 Create a new Project

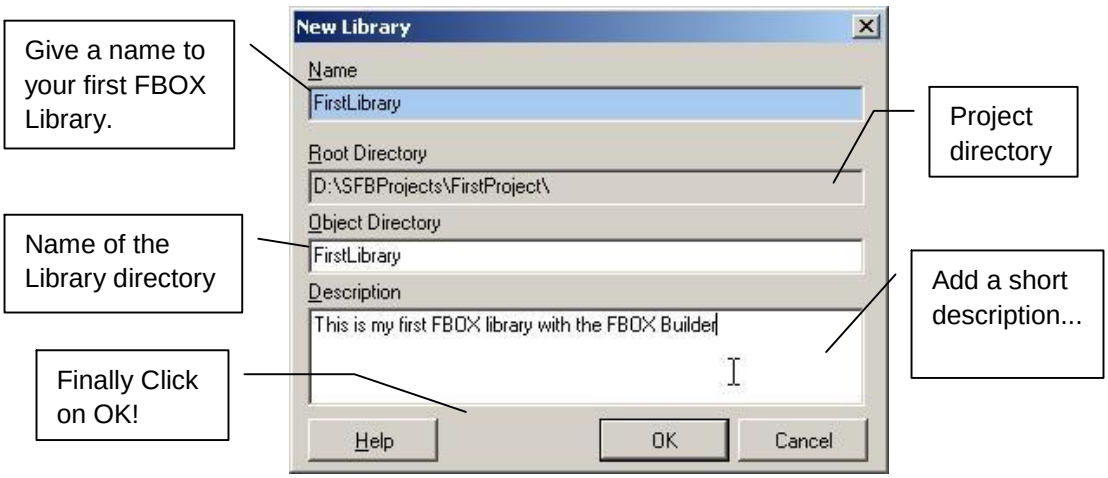
1. Click On New Project:



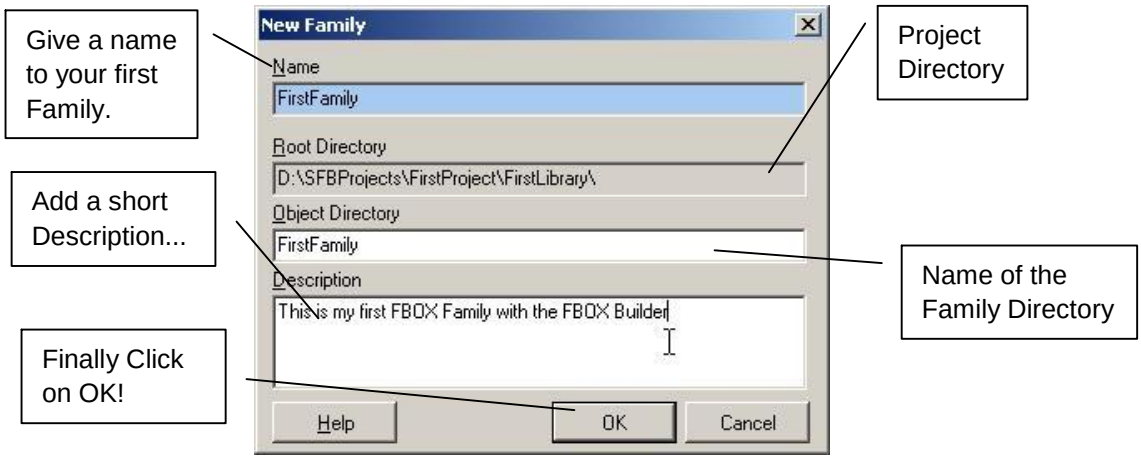
2. You get the New Project Dialog:



3. You get now the New Library Dialog:



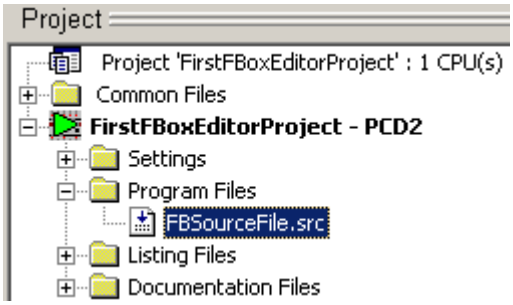
4. You get the Family Dialog:



Important: All the languages must be added when creating a project and should never be deleted. The Restore mechanism is not able to restore more languages than the “Actual” project languages number.

2.1.2 Write a simple Blinker FB

1. Create a new PG5 project called “FirstFBoxEditorProject” and add an IL source file called ***“FBSourceFile.src”*** to it, you should get something as follow:

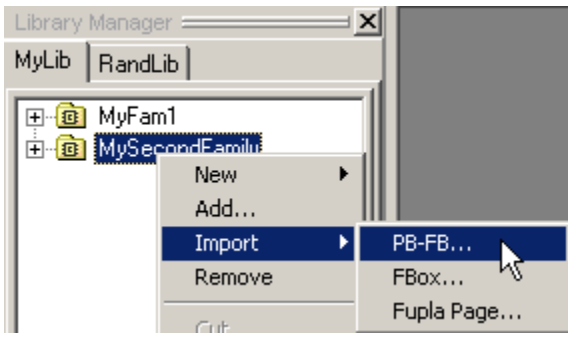


2. Copy the following code to the source file and save it.

```
;;Blinker FB
BlinkInit  EQU   FB
Blinker    EQU   FB
Timer      EQU   T

;;         Init FB
           FB          BlinkInit
IV         DEF         =1          ;; {In=IV_I} Timer Init Value
           GET         =1
                       Timer
           EFB

;;         Run FB
           FB          Blinker
Ou         DEF         =1          ;; {Out=Ou_B} Output
TV         DEF         =2          ;; {In=TV_I} Time Value
           STH         Timer
           JR          H EndFb
           ACC         H
           GET         =2
                        Timer
           COM          =1
EndFb:
          EFB
```

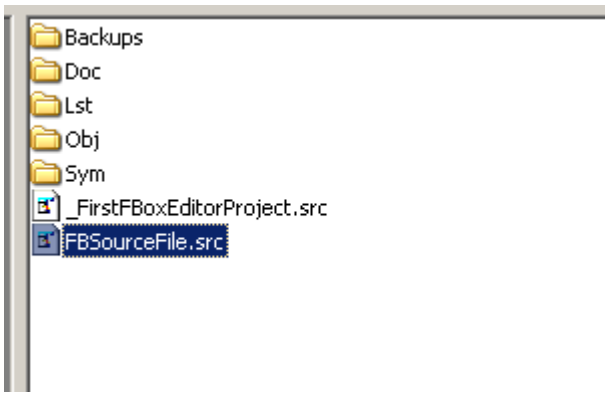
2.1.3 Create an FBOX based on the “Blinker” FB

1. Right click on the Family and select the menu “Import -> PB-FB…”:

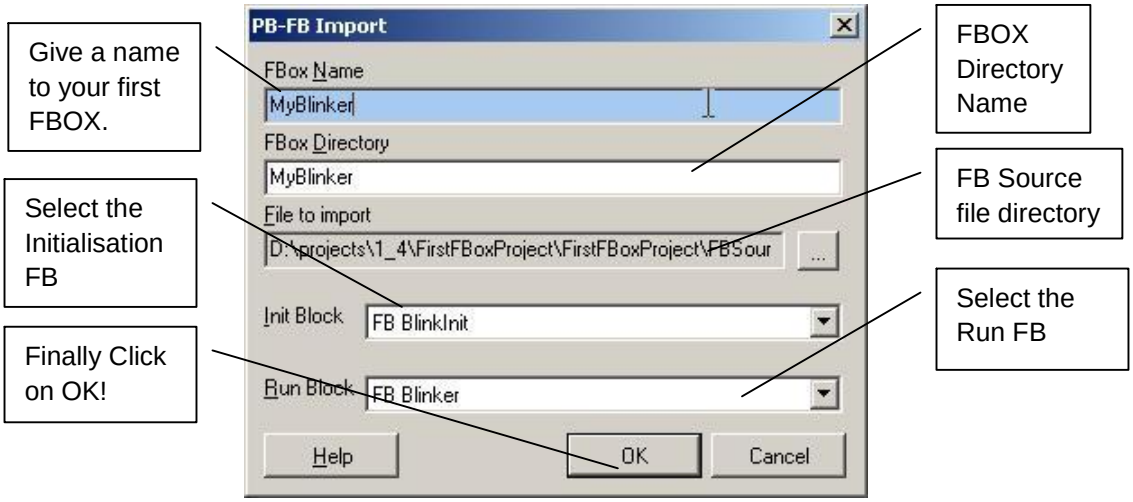


2. Browse to the PG5 project you just create:

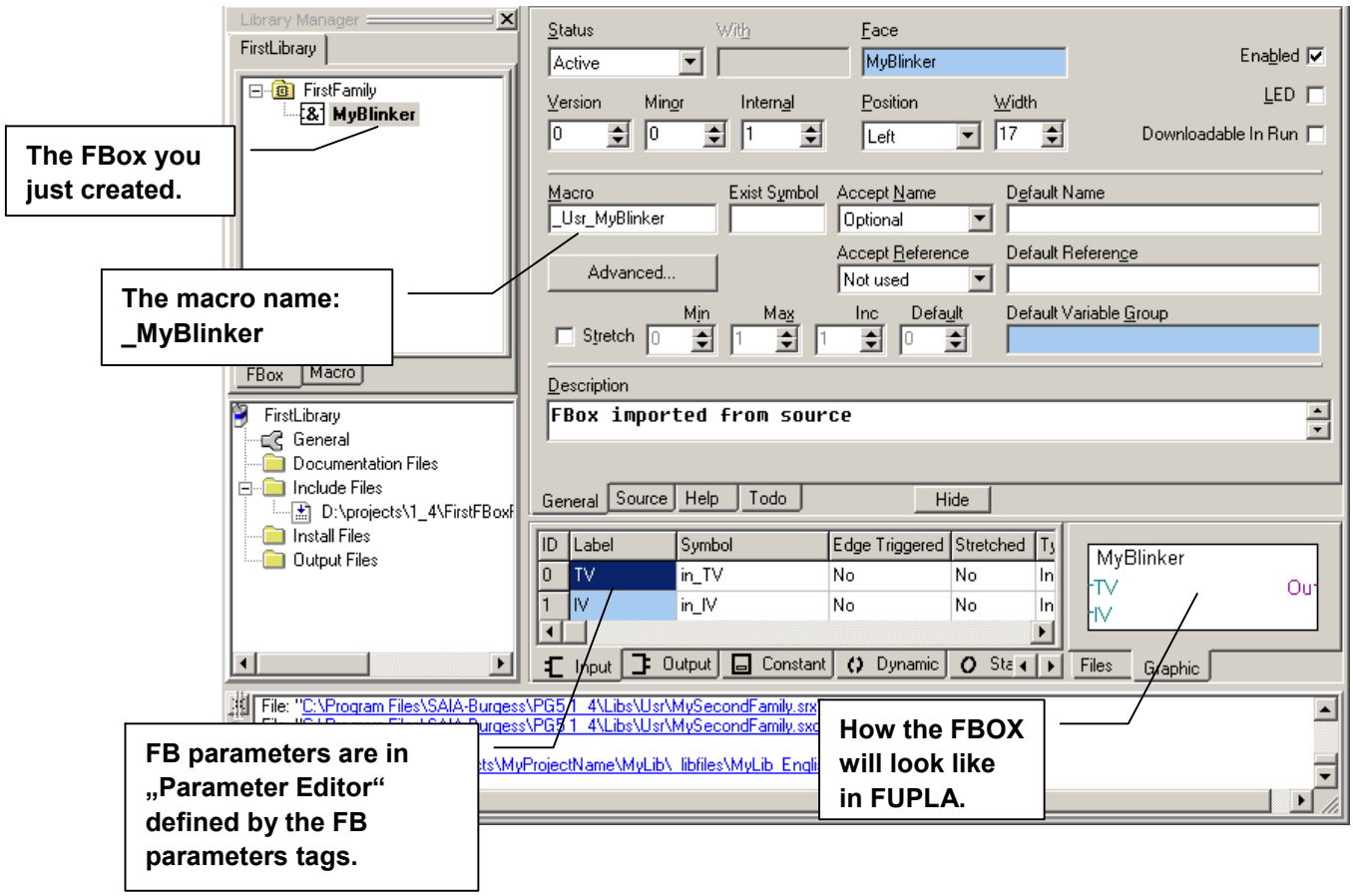
3. Select the FB source file:



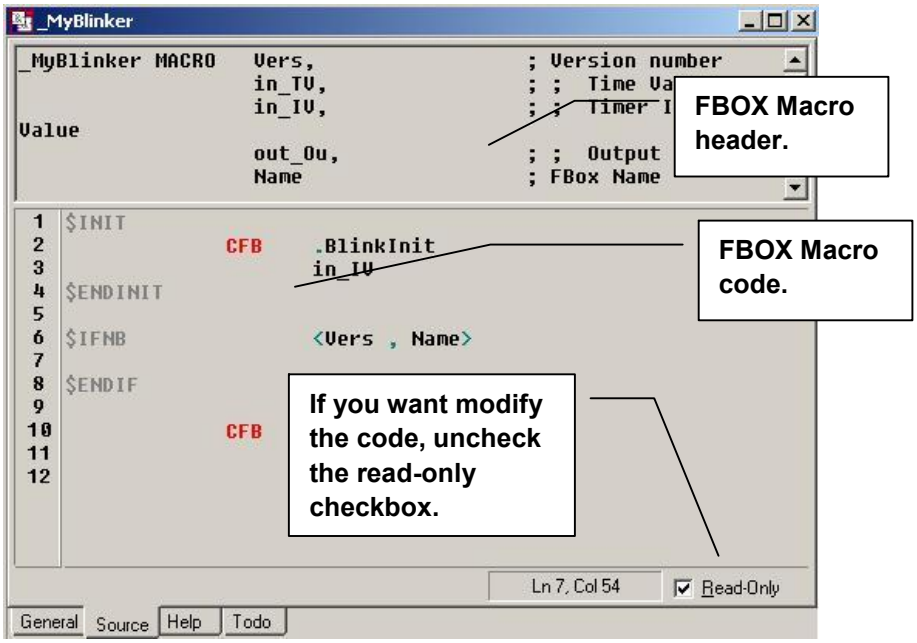
4. You get the PB-FB Import Dialog



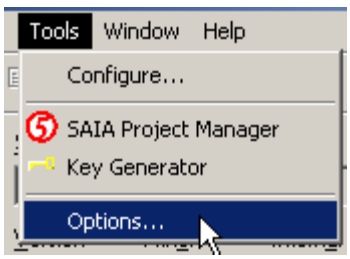
5. You get the following workspace:

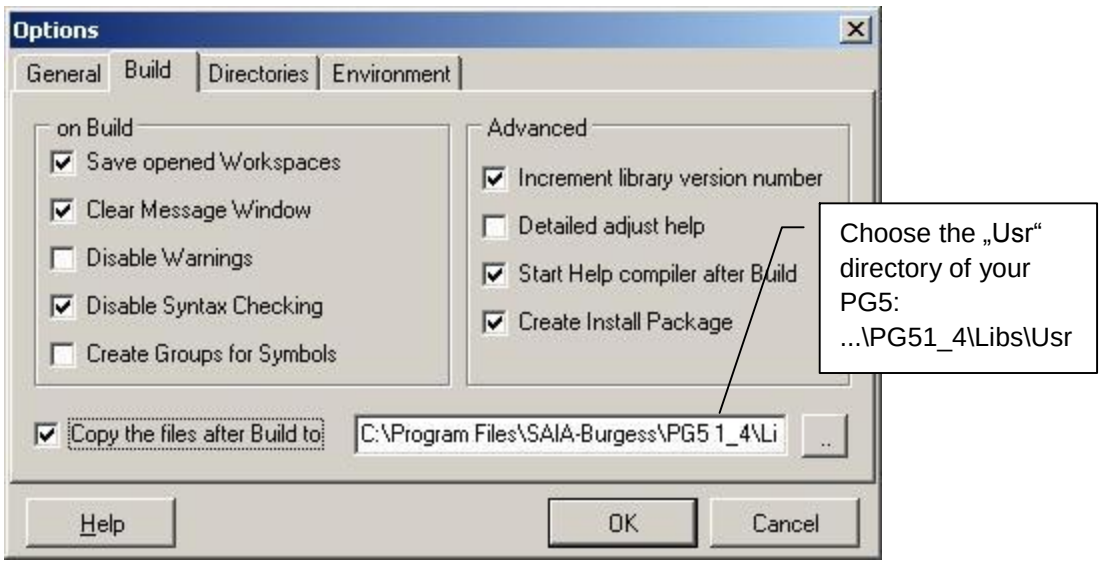


6. You can see the code of your FBOX:



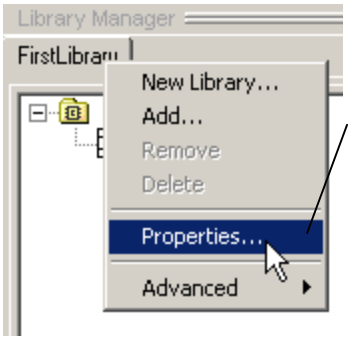
7. Choose where to copy the library files after build:





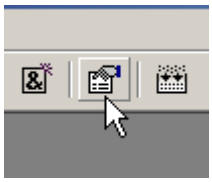
And click the “OK” button to confirm your selection.

8. Add the Library Information

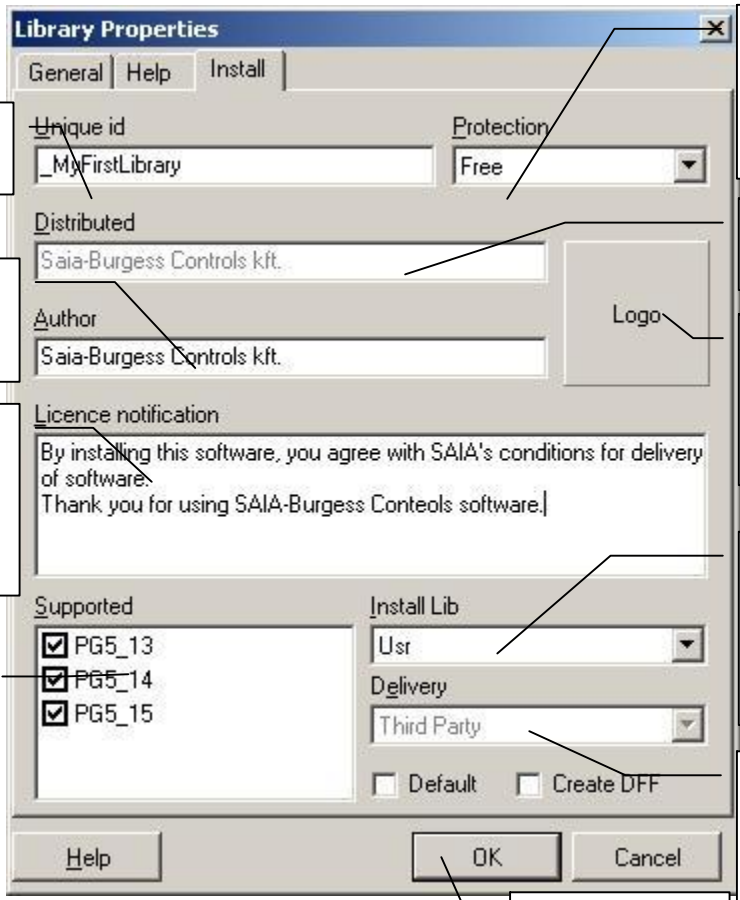


Select Properites in Library Tab's context menu.

or



And fill up the information fields:



Enter a unique library name

Enter the name of the developer

Enter the text that will be displayed during the installation

Check the PG5 versions that support your library

Choose the kind of protection you want when distributing your library

Your company name (read-only)

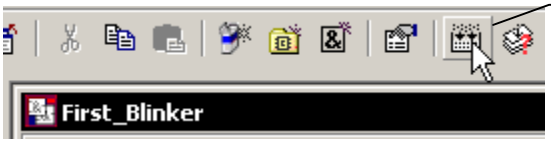
Browse for a Logo that will be displayed on the setup package

Choose the directory in which you want to install your library

Choose the type of delivery that correspond to your needs

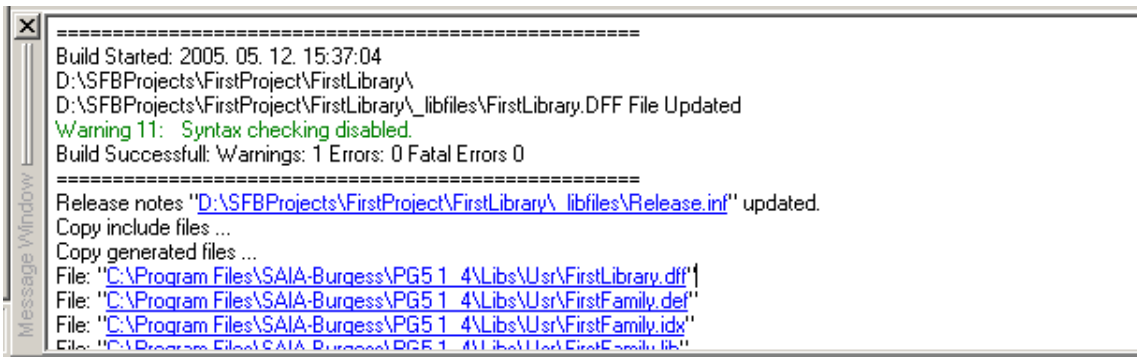
Finally click OK to save your work

9. You are ready now to generate the FBOX Library Files:



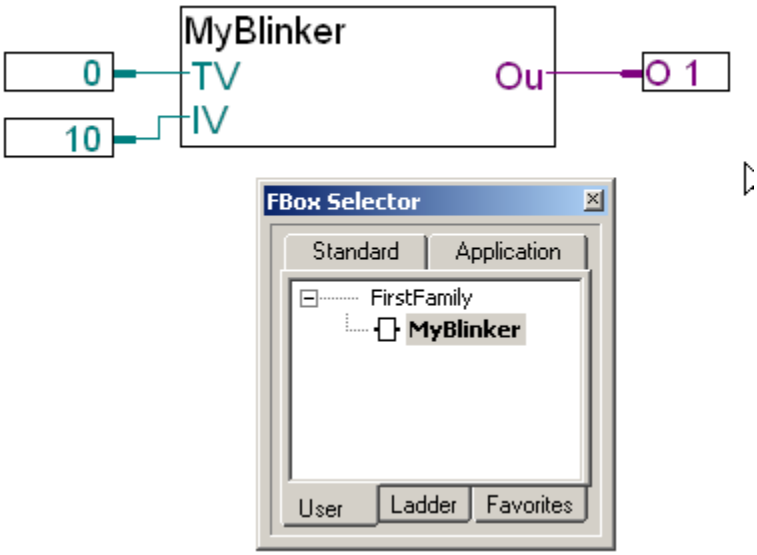
Click on the Build Library button of the ToolBar or press „F2“

10. The message window looks like this:



11. Try your First FBOX:

    Create a new FUPLA file in PG5, select the User Tab of the FUPLA “FBOX Selector”, select the “MyBlinker” FBOX and drop it on the FUPLA page. This is what you get:

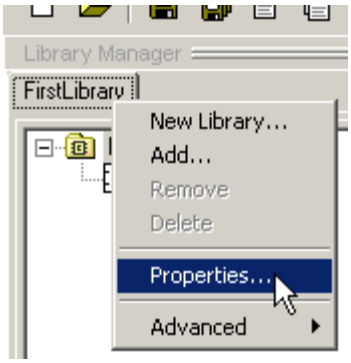


12. Compile and download the program in your PCD.

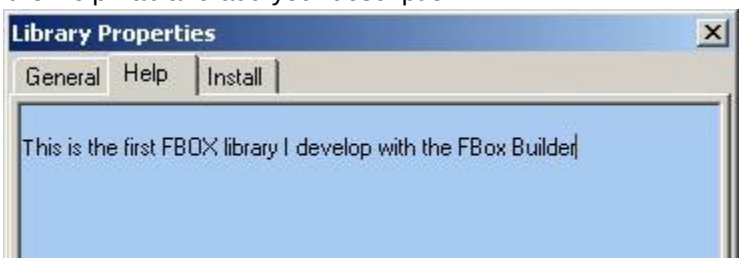
2.1.4 Create the Help File for your new FBOX Library

1. Add a description for the Library:

   Right Click on the Library Tab and Select the “Property” menu:



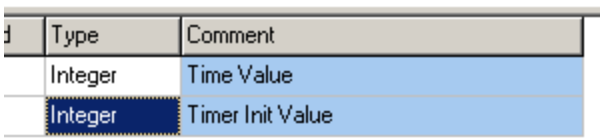
   Click on the Help Tab and add your description:



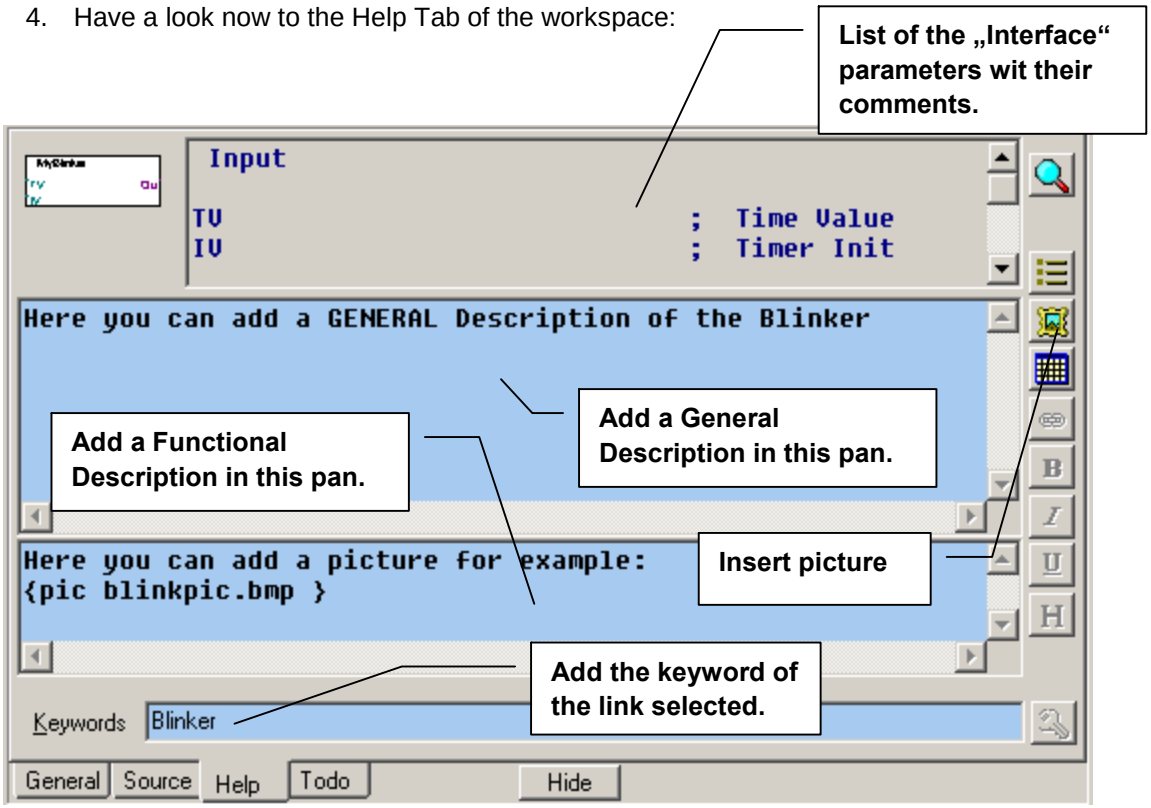
2. Add a description for the Family:

   Right Click on the Family and Select the “Property” menu. (Idem than the Library)

3. Add Comments for each parameter of the “Parameter Editor”:

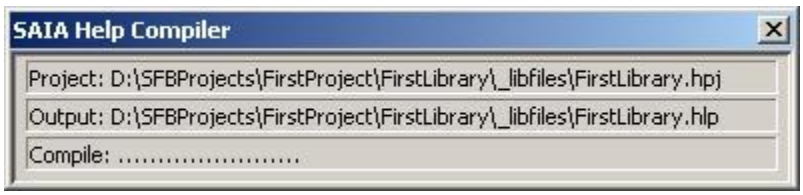


4. Have a look now to the Help Tab of the workspace:



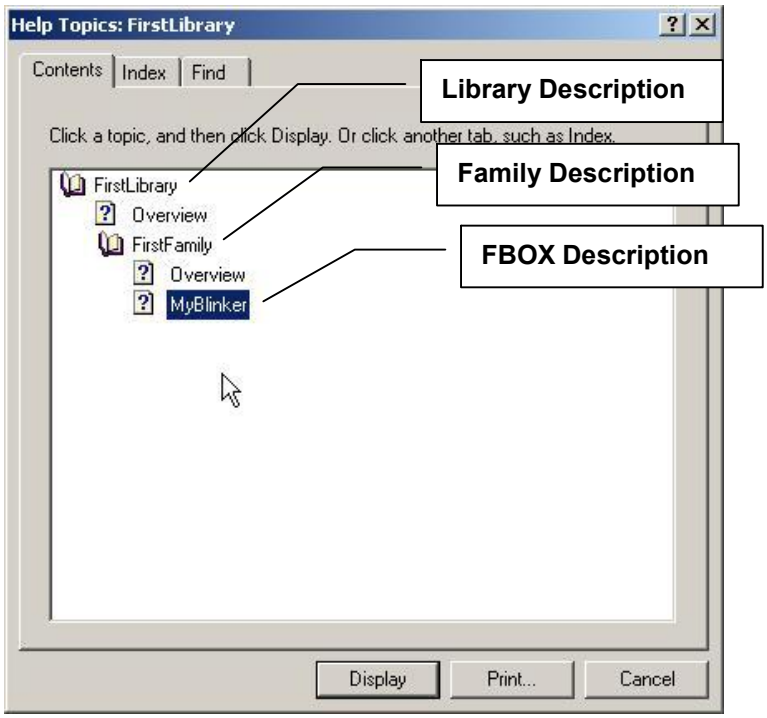
5. Compile the Help File:

You will see the following window during the help build:

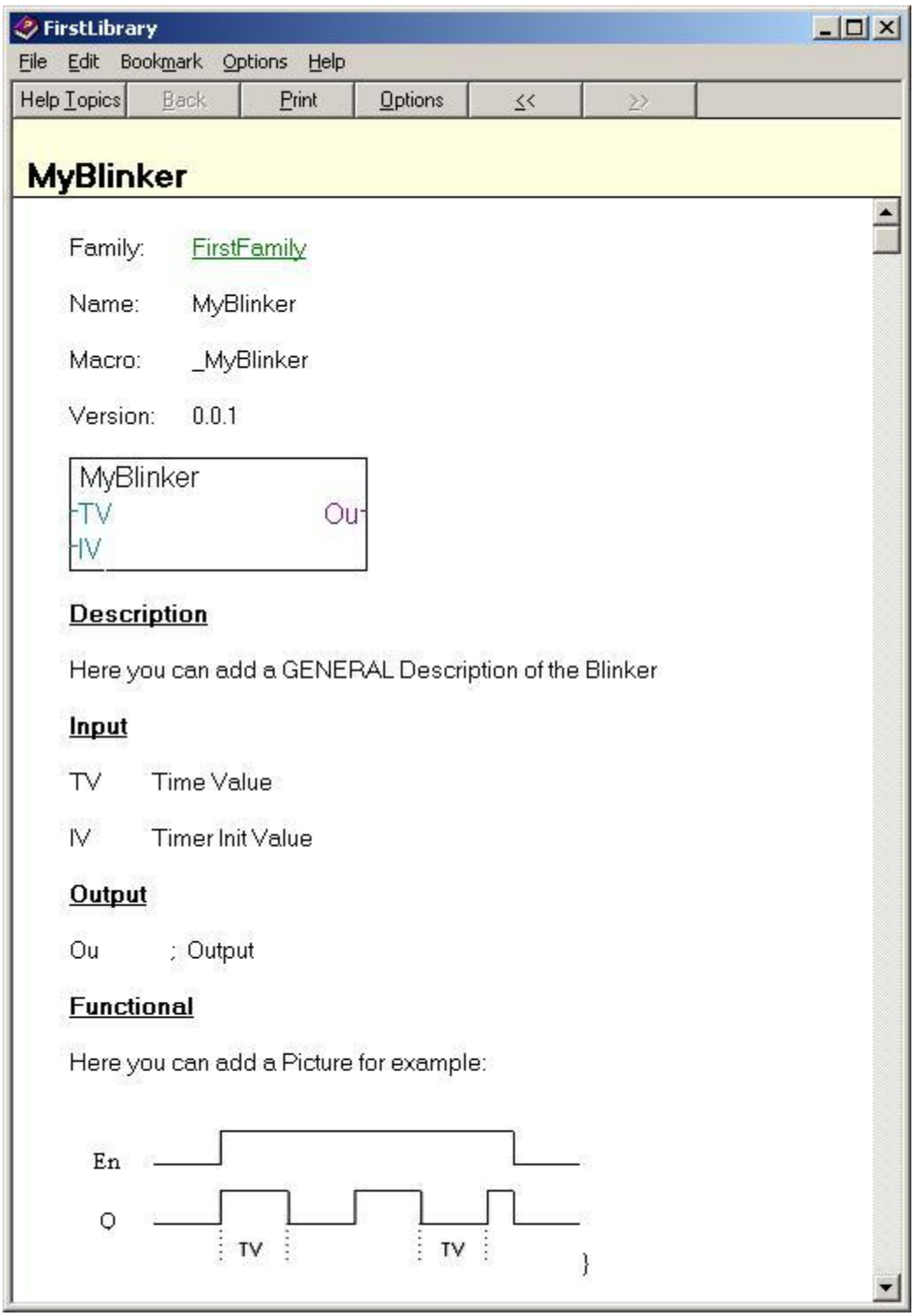


The message window will look like this:

6. Double click on the “FIRSTLIBRARY.HLP” which is also in the Output Directory:



7. Have a look at the FBOX Description:

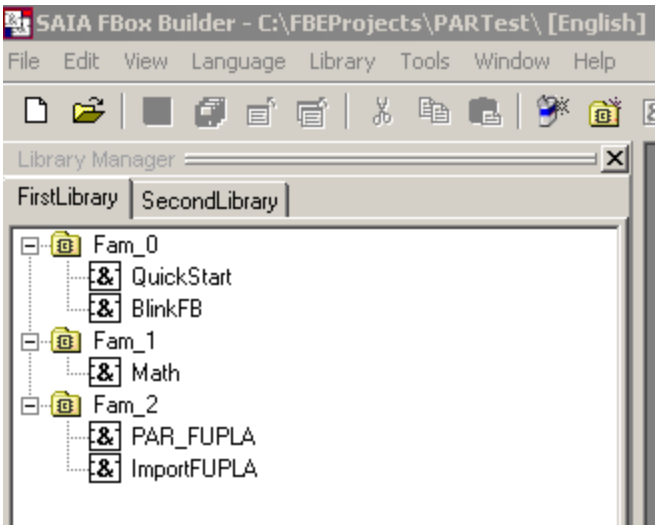


Contents

3 FBOX Builder Hierarchy

3.1 The Project Level

The FBOX Builder is organised in projects. The project level is the start point of any activity with the FBOX Builder. Basically a project is the only level that belongs only to the user. The user can organise freely his project without any constraints regarding FUPLA.
A Project defines a working directory where all the data needed by the FBOX Builder are stored. An output directory must also be given by the user to tell where the files generated by the compilation must be placed.
The Project contains one or more FBOX Library, which themselves contains one or more FBOX Family, which themselves finally contains one or more FBOXES.



As you can see in the title bar of the above picture, the language of the FBOX Library belongs to the project level.

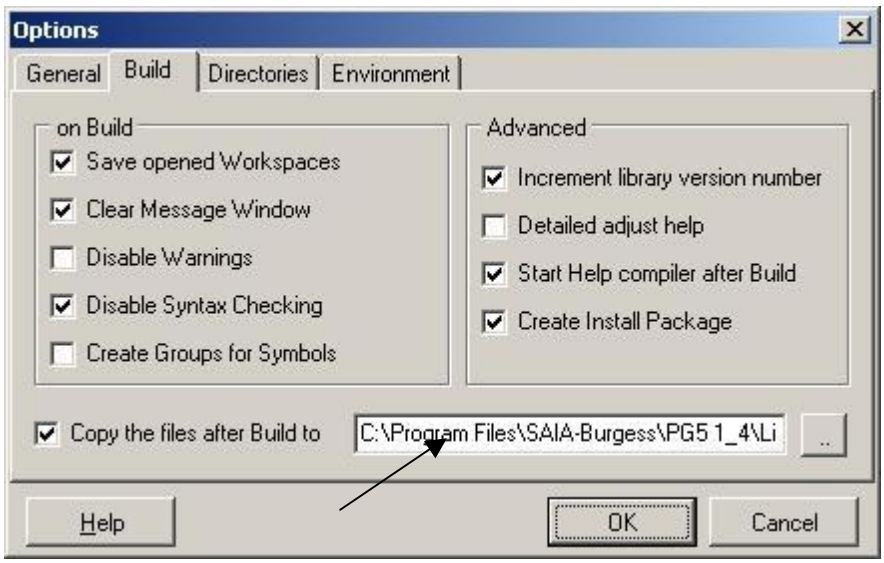
3.2 The Library Level

The FBOX Library contains FBOX Families. The Library level of the FBOX Builder is the same as the Library level defined in FUPLA. The FUPLA libraries are the following:

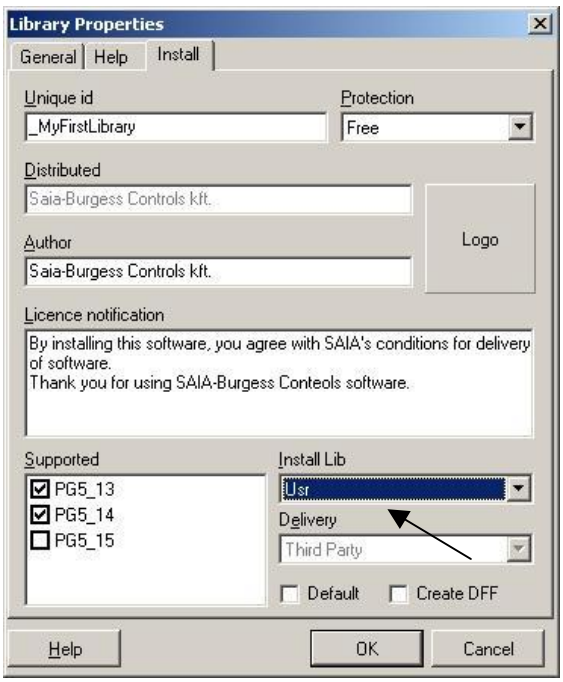
- Standard Library
- Application Library
- Ladder Library
- User Library

When a library development is finished, the user can move it to the FUPLA user library in the "…\libs\usr" directory and the library will appear in the FBOX Selector in FUPLA.

For example in the menu "View → Options" there is the possibility to choose to copy the Library files in the PG5 library directory of your choice.



Or you can also create an install package if you want to distribute your library and in that case the target directory is defined in the Library properties dialog.

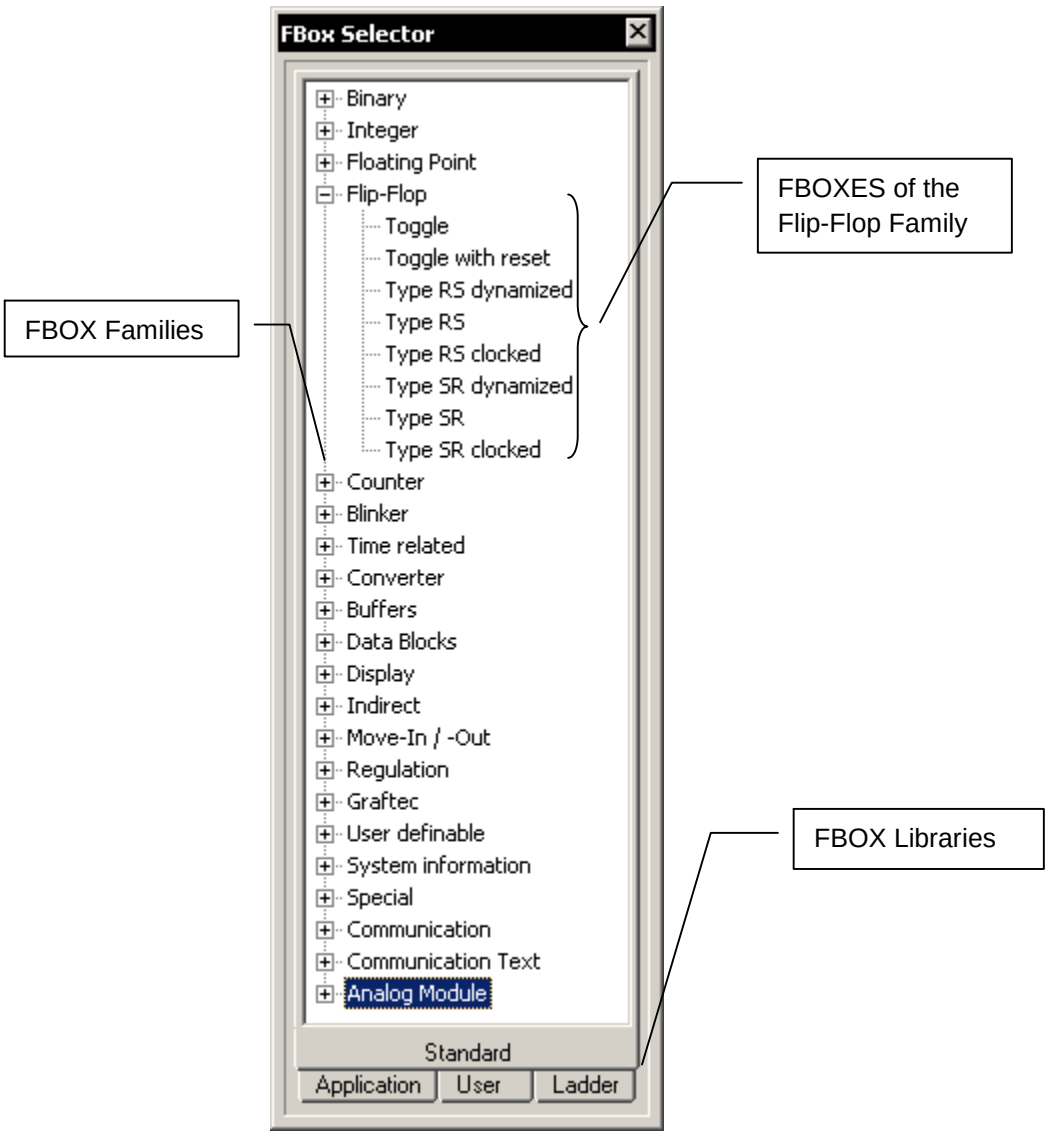


The compiler works at the Library Level; so when you compile, you compile a whole FBOX Library.

3.3 The Family Level

The FBOX Family contains FBOXES. The family level of the FBOX Builder is the same as the Family level defined in FUPLA.

As example, this is the family list of the FUPLA Standard library:



Contents

4 FBOX Basics

In this chapter you will see an example how to make a simple FBOX with the FBOX Builder. The “Adjust and View” variables and the language handling will be treated in the next chapter.

4.1 FBOX Development Process

To develop an FBOX Library, there are some steps to follow. Of course every developer has his own habits. This paragraph roughly describes the main steps to create an FBOX Library.
The FBOX Builder gives you several ways to build an FBOX. You can import an existing FBOX and modify it, you can import existing code as FB or FUPLA page or you can create an FBOX from scratch. Each of these solutions has its own particularity but all have basically the same main steps during the development.

The central FBOX file is the “DEF” file that defines the interface of the FBOX. Usually, to develop an FBOX, this file would be the start point. With the FBOX Builder the philosophy is the same, before writing the functionality of your FBOX, you have to define what is going to be its interface with the outside world and what internal data you gone a use to build the functionality. **So the first step is to define the “Input”, “Output”, “Constant” and “Adjust” variables**. The “Static” and “Dynamic” variables can be defined as needed during the definition of the functionality of the FBOX, of course all variables can evolve and are never fixed; at every time you can add new ones and remove others. This step will be done in the “Parameter Editor” if you begin an FBOX from scratch or it will be done in the source file of the FB for example; but in all cases, the “Parameter Editor” will greatly help you for this kind of work.

When the interface of the FBOX has been defined, **the second step is to write the behaviour of it.** This step can be done in IL code with the “Source Editor” of the FBOX Builder or with the text editor of your choice or with FUPLA.

At that stage, your FBOX is functional. As **third step,** you can **create a “Help File”** using the “Help Generator” of the FBOX Builder. This help file is based on the comments you entered for each FBOX parameter, the help text you entered for the Library and the Family and finally on the text, picture and link you add in the “Help” tab that belongs to the FBOX.

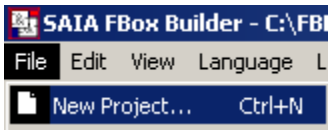
It is now time as **fourth step to test your Library**, to do so you have to build it and copy the FBOX files to the FUPLA directory “USR”. If you check in the option dialog “Copy the files after build to:” and “Start Help compiler after build” all the necessary files will be copied automatically to the right directory. Finally open FUPLA and create a small test program with your new FBOXES and check them out. This project can be added to Test projects in Info window’s “Files” tab for easier access.

The final step is the deployment of your Library. If you check the option “Create Install Package”, after a successful build an “EXE” file will be created. This file is created based on the Library Information you gave in the “Library properties” dialog in the FBOX Builder. This exe will simply copy the FBOX files in the directory chosen.

4.2 Create a Project

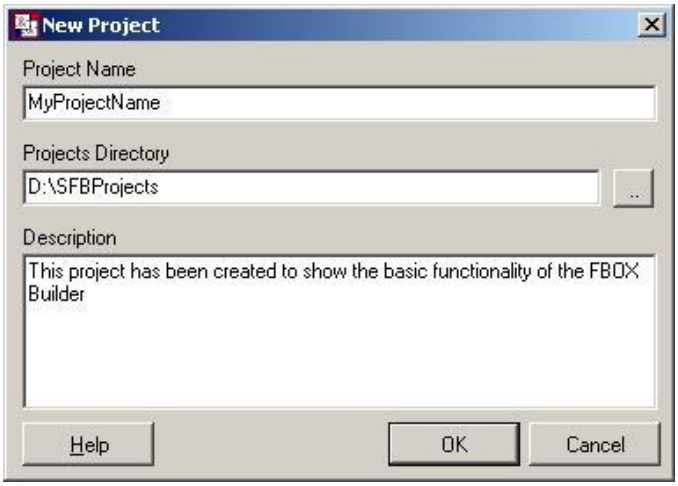
The project is the highest level of the FBOX Builder. A project can contain several FBOX Libraries, FBOX Families and single FBOXES in different languages.

To create a project you have to choose the following menu:



Or you can directly click on the following tool bar button:

The following dialog will appear:



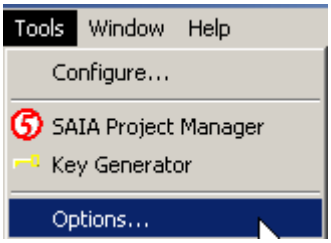
- **Project Name:** This entry is used by the FBOX Builder as a name for the root directory of all the data that belongs to this project.
- **Projects Directory:** This entry is used by the FBOX Builder as the root directory for the projects. For each new project you are creating a new root directory can be entered; this gives you another chance to organise your FBOXES.
- **Description:** This entry is only used internally, it will never appear in a help file or what ever output files.

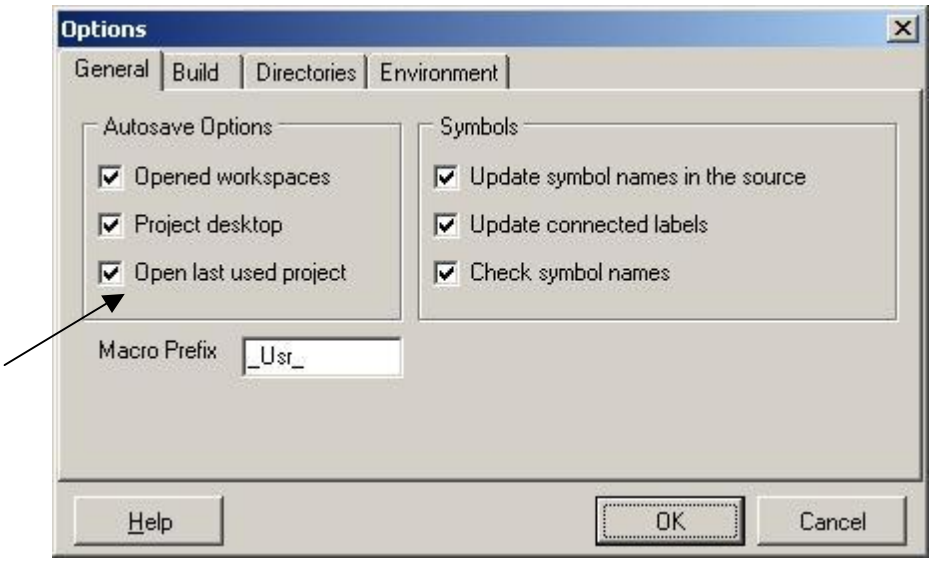
When you filled up all the entries, click on the OK button to accept or on Cancel to abort the current task.

Important: All the languages must be added when creating a project and should never be deleted. The Restore mechanism is not able to restore more languages than the “Actual” project languages number.

4.2.1 Project Tips

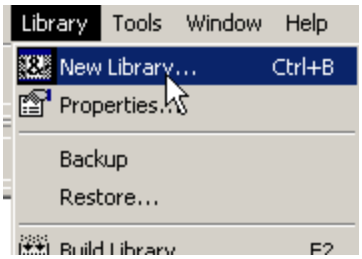
- You can choose in the options dialog to always open the last project used:





4.3 Create a Library

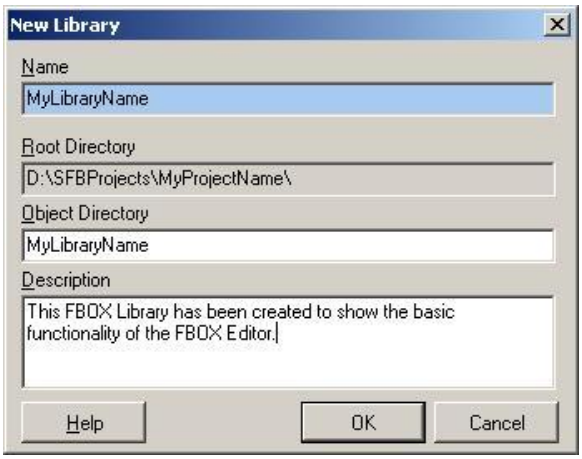
If you are currently creating a new project, the following dialog will automatically appear when you clicked on the OK button of the “New Project Dialog”. But if you want to add a new library to an existing project, you have to use the following menu:



Or you can use the tool bar button:

Or in right clicking on an existing library tab of the Library Manager, you get the following popup menu:

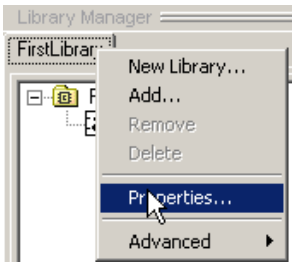
You will get the following dialog:



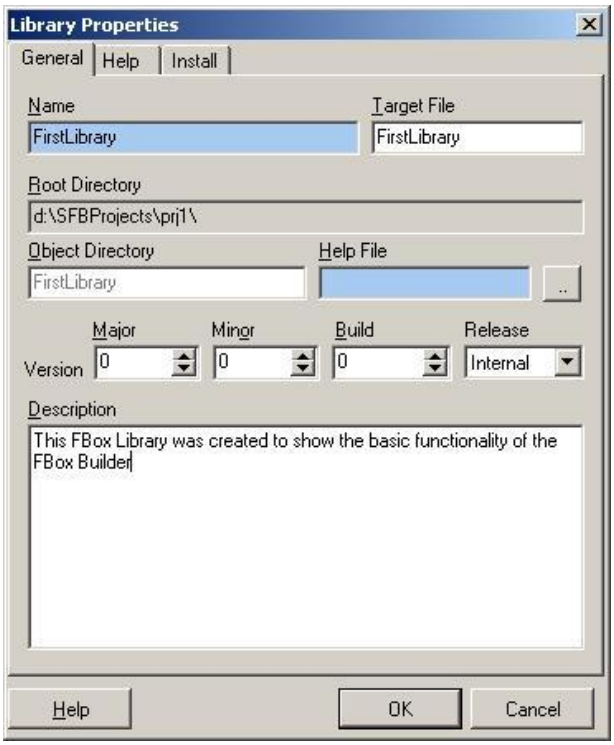
- **Name:** the FBOX Builder will use this entry as file name when generating the Library related files.
- **Root Directory:** The library you are creating will be place under the project directory you just have created.
- **Object Directory:** This entry is used by the FBOX Builder as root directory name for the Library. By default the library directory has the same name as the library.
- **Description:** This entry is used to describe the library. It is only used internally; it will not be part of the help file generated by the FBOX Builder.

4.3.1 Library Properties

All information that belongs to the library can be edited in the “Library Properties” dialog. You can obtain this dialog in right clicking on the Library tab and choose the “Properties…” menu:

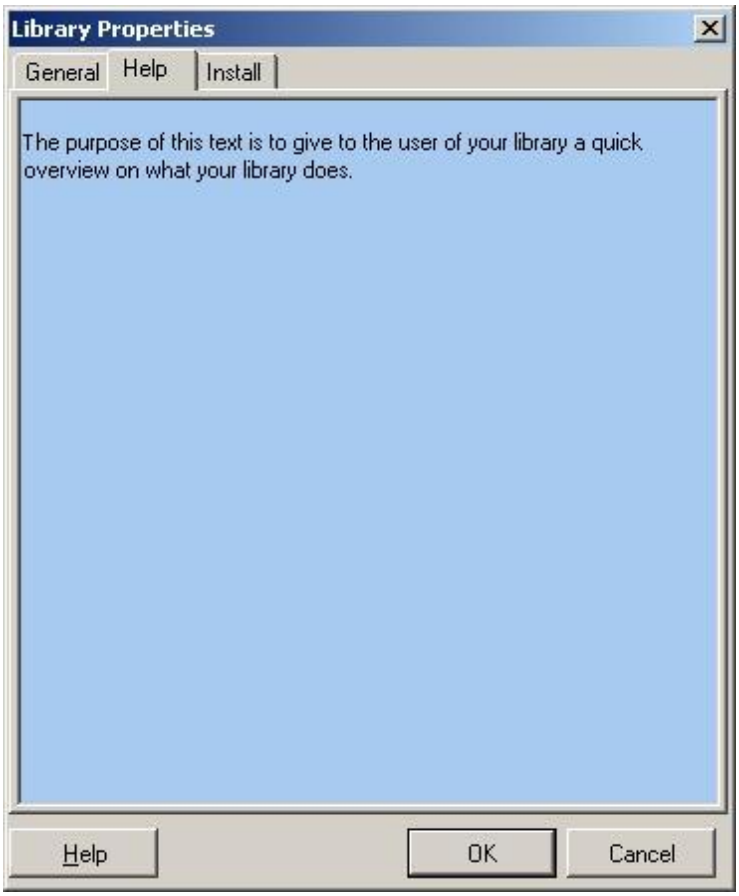


You will get this dialog:



- **Name:** This entry is Library name previously entered. The library name can be modified; the library tab of the “Library Manager” will adapted.
- **Target File:** Specify here the library filenames (.LIN).
- **Root Directory:** The location of the current library. This field is read-only.
- **Version:** This entry is used to manage the modification of the library. Each time you have to make changes in your library, you should increment the version. This way you can keep track of your modifications in the release notes generated by the FBOX Builder.
- **Release:** Internal means that the version will begin with '$' sign
  Beta means that the version will begin with 'b' sign
  Official means that the version will begin with 'v' sign
- **Description:** This is the library description previously entered.

We'll now have a look at the "Help" tab of this dialog:

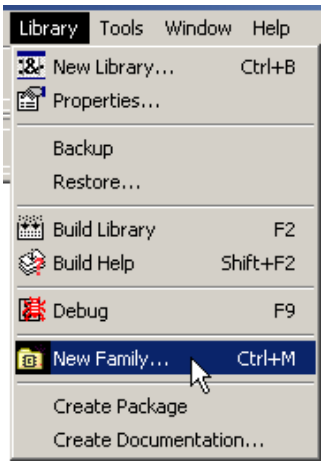


This tab is very simple; the text you'll write will be displayed as "Library Overview" in the help file generated by the "Help Generator" of the FBOX Builder.

In this chapter, we will not treat "Install" tab. Just for your information, the "Install" tab contains all the data needed by the "Library Installer" of the FBOX Builder. This feature will be explained later in the "FBOX Advanced" chapter.

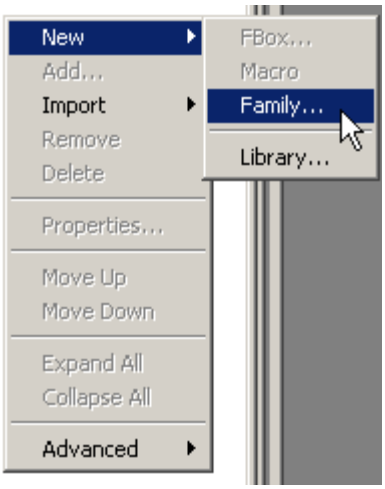
4.4 Create a Family

If you are currently creating a brand new project, the following dialog will automatically appear when you clicked on the OK button of the “New Library” Dialog. But If you want add a new family to an existing project, you have to use the following menu:

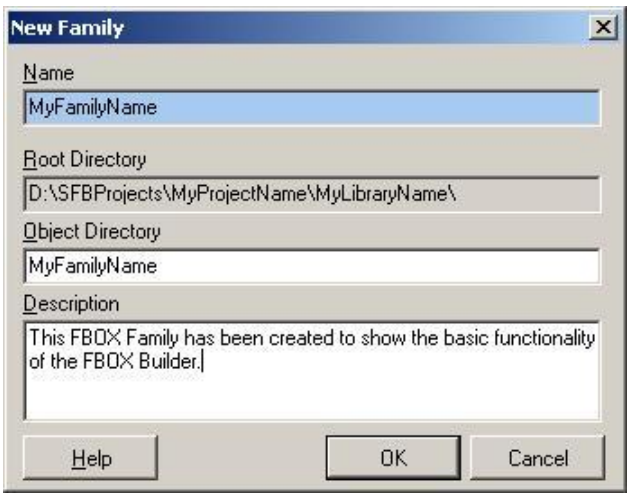


Or you can use the tool bar button:

Or in right clicking on an existing family folder of the Library Manager, you get the following popup menu:



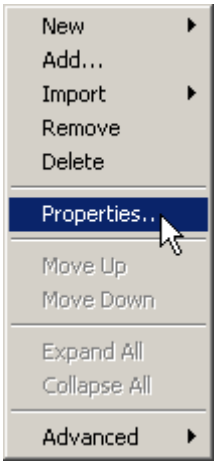
You will get the following dialog:



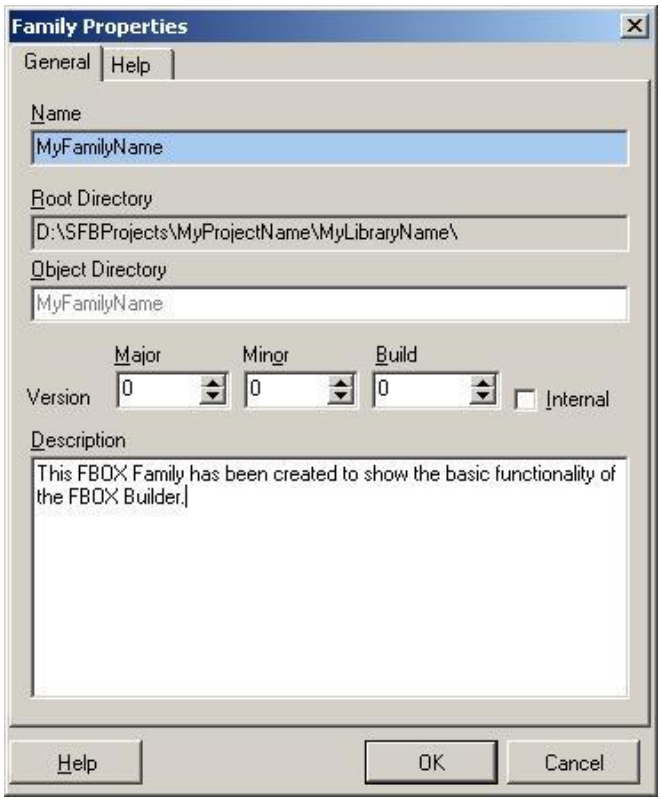
- **Name:** the FBOX Builder will use this entry as file name when generating the Family related files.
- **Root Directory:** The Family you are creating will be placed under the Library directory you just have created.
- **Object Directory:** This entry is used by the FBOX Builder as root directory name for the Family. By default the Family directory has the same name as the Family.
- **Description:** This entry is used to describe the Family. It is only used internally; it will not be part of the help file generated by the FBOX Builder.

4.4.1 Family Properties

All information that belongs to the family can be edited in the “Family Properties” dialog. You can obtain this dialog in right clicking on the Family folder and choose the “Properties…” menu:

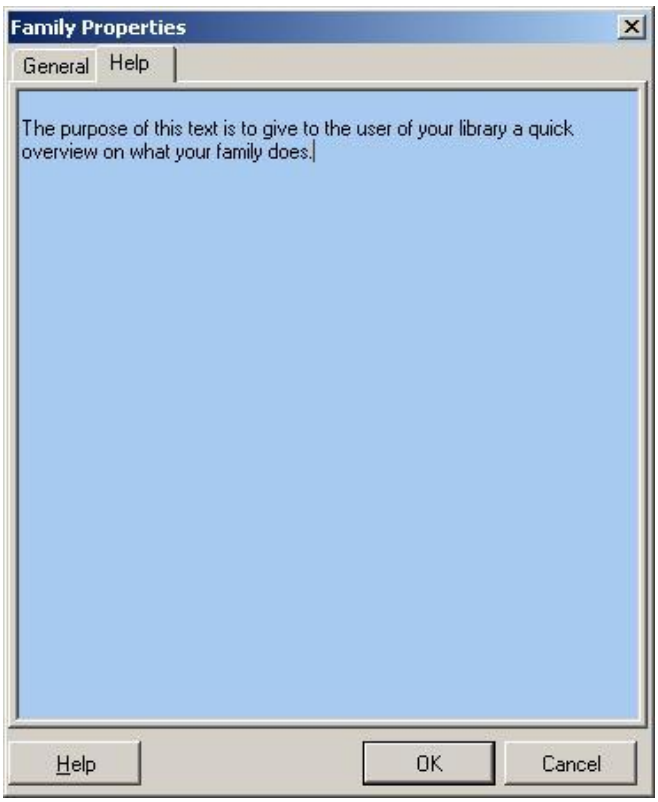


You will get this dialog:



- **Name:** This entry is Family name previously entered. The Family name can be modified; the family folder name of the “Library Manager” will be adapted but not the family directory itself.
- **Root Directory:** The location of the current family. This field is read-only.
- **Version:** This entry is used to manage the modification of the family. Each time you have to make changes in your family, you should increment the version. This way you can keep track of your modifications in the release notes generated by the FBOX Builder.
- **Internal:** If this check box is checked, the “Library Installer” of the FBOX Builder will display during the installation the “$” character before the version number.
- **Description:** This is the family description previously entered.

We’ll now have a look at the “Help” tab of this dialog:



This tab is very simple; the text you’ll write will be displayed as “Family Overview” in the help file generated by the “Help Generator” of the FBOX Builder.

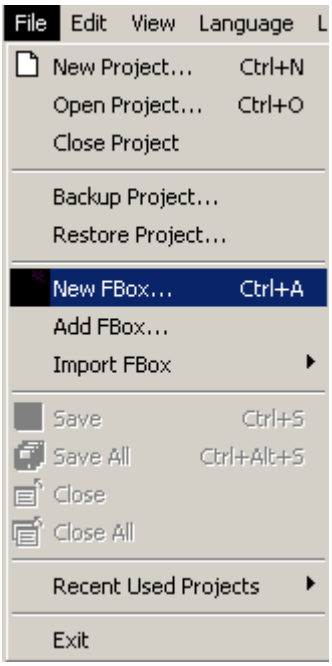
4.5 Create an FBOX

There are several possibilities to create a new FBOX with the FBOX Builder:

- Import an existing FBOX
- Import an existing FB or PB (Quick start example)
- Create an FBOX from scratch

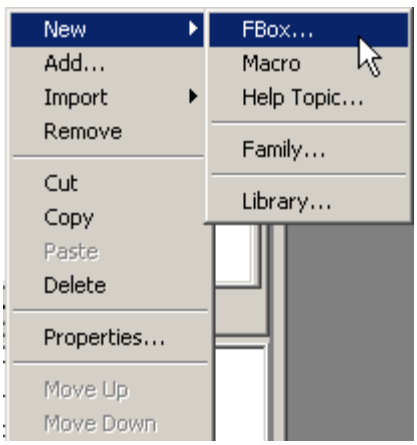
In this chapter, to show you the entire link between the different parameters of an FBOX, we'll begin a brand new FBOX from scratch.
As example, we'll develop a "Math" FBOX that will evolve through this chapter to the "FBOX Advanced" chapter also. We will give some more functionality to this FBOX all along the chapters.

You can create a new FBOX with the following menu:

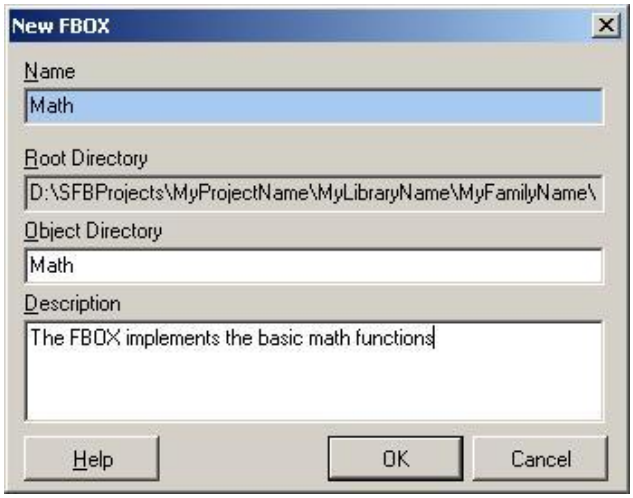


You can also use the following tool bar button:

Or in right clicking the family in which you want add the new FBOX:



You will get the following “New FBOX” dialog:



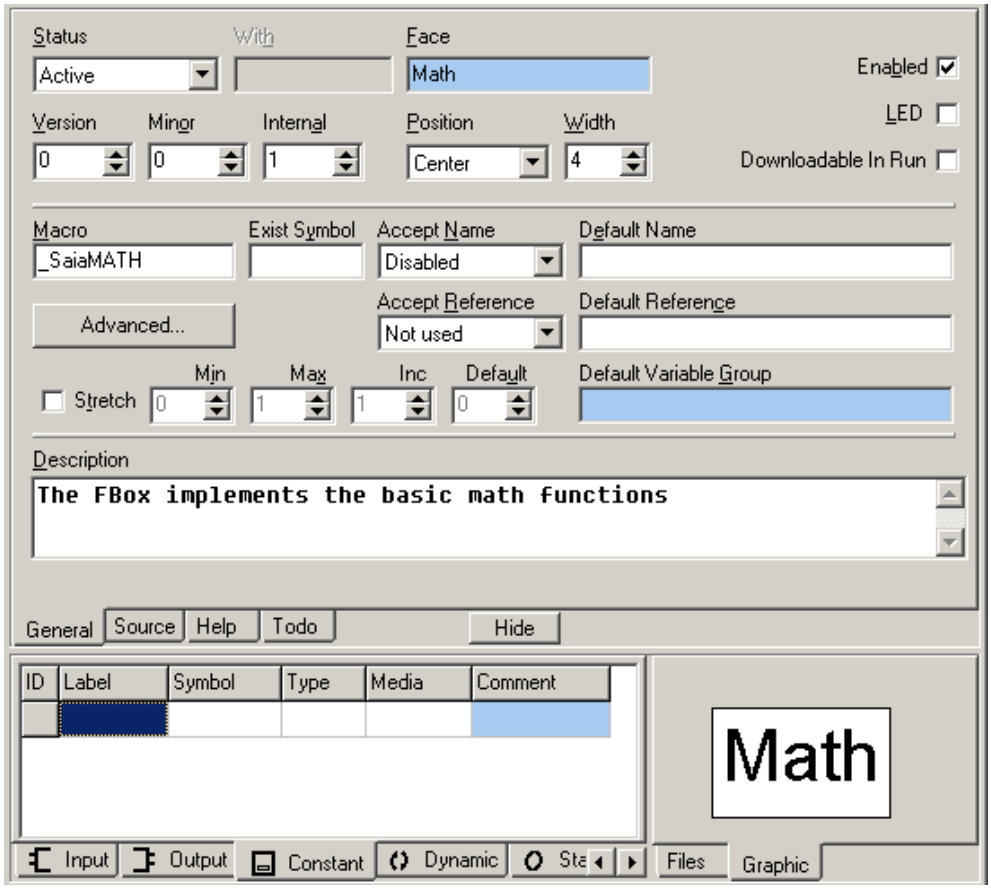
- **Name:** The name of the FBOX is very important, it will be used as default to be displayed on the FBOX Face, the “Help Generator” will use it as topic keyword and finally the FBOX Builder will use this name as default directory name to store the FBOX data.
- **Root Directory:** The FBOX you are creating will be placed under the Family directory you just have created.
- **Object Directory:** This entry is used by the FBOX Builder as root directory name for the FBOX. By default the FBOX directory has the same name as the FBOX.
- **Description:** This entry is used to describe the Family. It is only used internally; it will not be part of the help file generated by the FBOX Builder.

Click on the OK button to validate your entries or on Cancel to abort the process you began.

If you clicked on the OK button, a workspace that belongs to the created FBOX is displayed and a new FBOX node is created under the Family you have chosen for this new FBOX.

4.5.1 FBOX Workspace

The workspace is basically the tool to program and configure an FBOX. The workspace belongs to one and only one FBOX.



The upper part contains the following tabs:

- **General:** where you define the graphical interface of the FBOX.
- **Source:** where you edit the code (functionality) of the FBOX.
- **Help:** where you edit the Help topic that belongs to that FBOX.
- **TODO:** where the list of task which have to be done on the FBOX.
- optionally **: Schematic :** where the imported Fupla page is displayed.

The **Hide** button hides the Parameter Editor and the block info window (they are described below).

The lower part contains on the left side the **"Parameter Editor"** where you define the interface and the data needed by the FBOX. The right part of the workspace is called the **"Block info"** window and has the following tabs:

- **Files:** where you can see the FBOX attached files and where you can navigate into the workspace with double clicking on the node you want
- **Graphic:** where you see the graphical face of the FBOX you are building.

Important:

The first thing to do is to check the macro name of our FBOX because this name must be unique. A good habit is to use your company name as prefix of the macro name or at least an abbreviation like for example "Saia"; so the macro name of our FBOX will be "_SaiaMath"

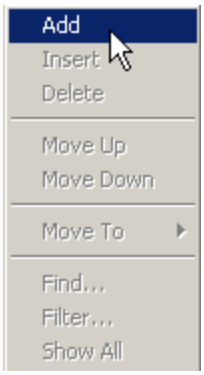
4.5.2 Define the FBOX Interface

As first step, we want to implement only the addition of two integer numbers. To do so we need to enter with two integers and go out with the result. We need also to define a bit what we want as graphical interface.

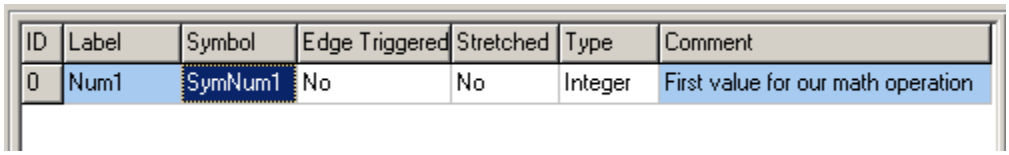
1. Add two inputs to the FBOX

   Select the “Input” tab of the “Parameter Editor”: 

   Right click on the “Parameter Editor” and choose the Add menu:



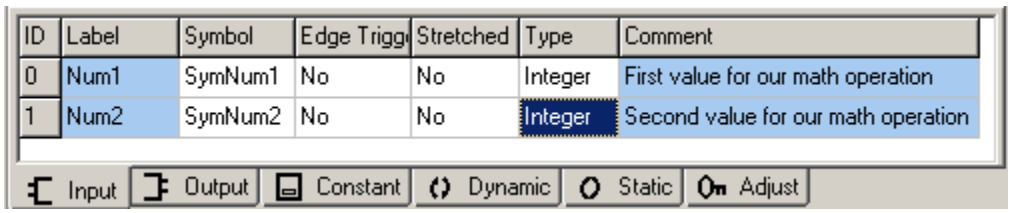
   And then fill up the different fields as follows:



ID	Label	Symbol	Edge Triggered	Stretched	Type	Comment
0	Num1	SymNum1	No	No	Integer	First value for our math operation

   - **Label:** The string displayed on the FBOX for this input entry.
   - **Symbol:** The string used in the source code to handle this input entry.
   - **Edge Triggered:** Used only in the case of a Boolean input to detect the raising or the falling edge of this entry.
   - **Stretched:** “Yes” means that this input entry is stretchable; in other words, the user can choose how many of this input he wants.
   - **Type:** This entry defines what kind of data format we handle for this math function
   - **Comment:** This comment describes the input entry and it will be used in the generated help file to explain the FBOX parameters.

   Add a second input for the second number to add, you should finally have something like that:

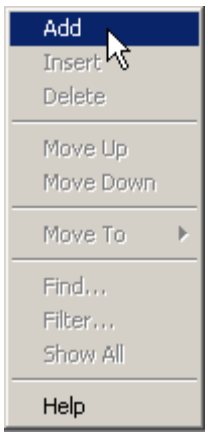


ID	Label	Symbol	Edge Trigg	Stretched	Type	Comment
0	Num1	SymNum1	No	No	Integer	First value for our math operation
1	Num2	SymNum2	No	No	Integer	Second value for our math operation

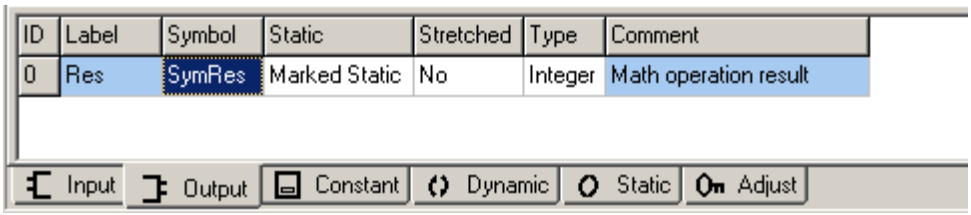
2. Add one output to the FBOX

   Select the “Input” tab of the “Parameter Editor”: 

   Right click on the “Parameter Editor” and choose the Add menu:



   And then fill up the different fields as follow:

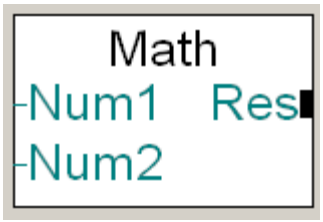


   - **Label:** The string displayed on the FBOX for this output entry.
   - **Symbol:** The string used in the source code to handle this output entry.
   - **Static:** Static means that the media that belongs to this entry is used only by this FBOX; in other words, this integer value can only be changed by the current FBOX and no other. The Marked Static simply asks FUPLA to display a small sign on the entry to show the user that it is static. With this option, FUPLA will reserve a static variable for this output and draw a small sign to show to the user. For the FBOX Developer, that means he does not need to write that output at each cycle, it is done automatically by FUPLA.
   - **Stretched:** “Yes” means that this output entry is stretchable; in other words, the user can choose how many of this output he wants.
   - **Type:** This entry defines what kind of data format we handle for this math function
   - **Comment:** This comment describes the output entry and it will be used in the generated help file to explain the FBOX parameters.

3. Play with the Graphical parameters

   On the “General” tab of the workspace you can increase the “Width” of your FBOX up to 10 and maybe choose the “Position” of the FBOX name to “Center”:

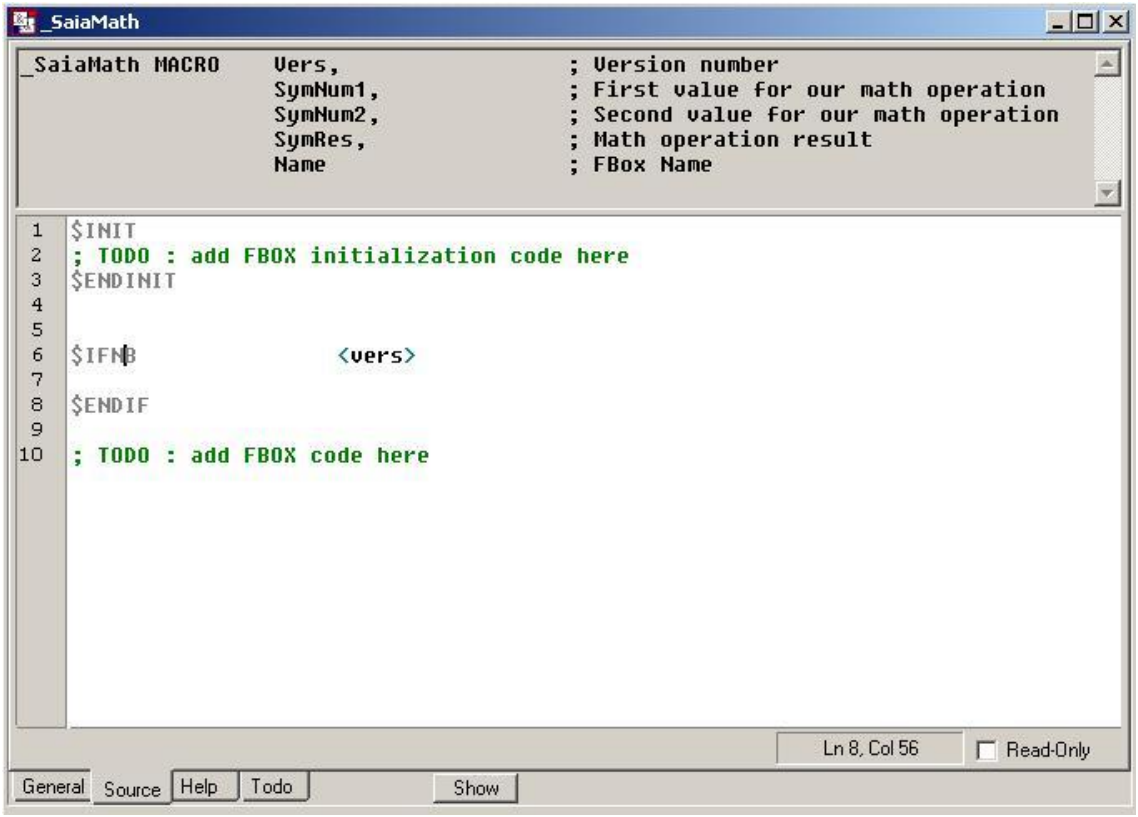
   The “Graphic” tab of the “FBOX Navigator” should display the following FBOX:



4.5.3 Write the FBOX Functionality

When the FBOX interface is defined, we must write the code that will give the behaviour of the FBOX.
Maybe we should first have a look at what the FBOX Builder has generated for us so far. Click on the “Source” tab of the Source workspace:

This is what you should get:



In the upper pan, you can see the macro header generated by the FBOX Builder. This part is read-only and it is entirely managed by the FBOX Builder. What you can see in this macro header:

- **_SaiaMath:** It is the name of the macro we have chosen in the previous steps.
- **Macro:** It is the reserved keyword for the macro declaration
- **Vers:** It is the version number of the FBOX that is passed to the macro if you want make further test related to the version.
- **SymNum1:** It is the symbol of the first input parameter we defined in the “Parameter Editor”.
- **SymNum2:** It is the second input parameter.
- **SymRes:** It is the symbol of the output parameter we defined in the “Parameter Editor”.
- **Name:** We did not change on the “General” tab the “AcceptName” entry. We left it to “Optional” that means the user can give a name to his FBOX and that name is passed to the FBOX macro where it can be used for example to define groups.

And in the lower pan is actually where you must write the code of the FBOX. In most cases, an FBOX has some data that needs to be initialised when the PCD starts; the initialise code must be placed in between the “$INIT / $ENDINIT” directives. The code that will be executed during the normal program cycle must be placed after the “$ENDINIT” directive.

For this first step in the “Math” FBOX development, we have decided to implement a simple “add” function that can be written in IL as follows:

```
ADD     SymNum1
        SymNum2
        SymRes          ;;Result of (SymNum1 + SymNum2)
```

For this function, we should initialise all the parameters with the “0” value; we should add the following lines as initialise code:

```
LD      SymNum1
        K 0
LD      SymNum2
        K 0
LD      SymRes
        K 0
```

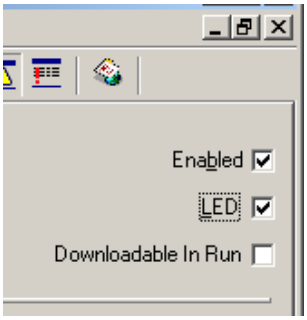
Here is the source you should get:

```
1  $INIT
2  ; TODO : add FBOX initialization code here
3                LD      SymNum1
4                        K 0
5                LD      SymNum2
6                        K 0
7                LD      SymRes
8                        K 0
9  $ENDINIT
10 $IFNB                 <vers>
11 $ENDIF
12 ; TODO : add FBOX code here
13
14               ADD     SymNum1
15                       SymNum2
16                       SymRes              ;;Result of (SymNum1 + SymNum2)
17
```

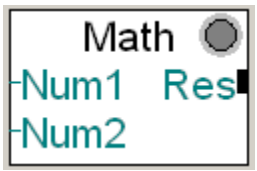
4.5.3.1 FBOX LED Handling

It is possible to show the internal status of an FBOX when you are online with FUPLA in adding a LED on its face, which is red in a certain state, green in another state. When you are offline, this LED stays grey.

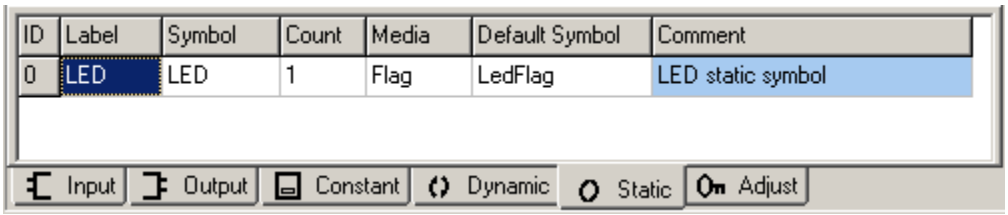
To add this LED you just have to select the “LED” checkbox that is situated on the “General” tab General of the workspace:



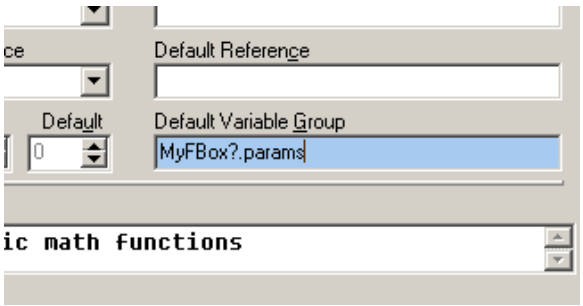
The result on your FBOX is the following:



This LED is linked to static flag, which is automatically added when you select this option. Click on the static tab, and fill the **Default Symbol** like this:



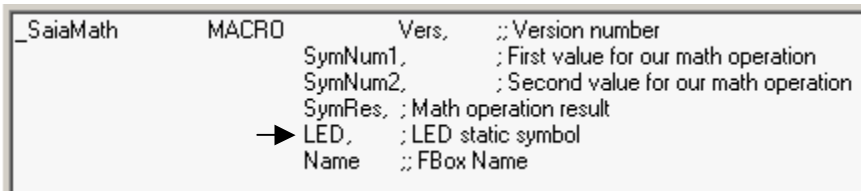
Now fill the Default Variable Group in the General tab like this:



Now when an FBox like this is added to a Fupla page, the led symbol will appear in Symbol Editor. The symbol's name will be: MyFBox0.params.LedFlag. After adding an other FBox, a MyFBox1.params.LedFlag symbol will be added to Symbol Editor and so on.

Note: If the Default Variable Group is not filled, no symbol will be added to Symbol Editor, even if they are defined in the Static tab - Default Symbol cell.

And you can also have a look in the “Source” Source tab of the workspace, now the macro header has one more parameter called “LED”.



In changing the state of this “LED” flag in the macro, you change the state of the LED of the FBOX. The FBOX LED will be red if this flag is SET and green if this flag is RESET. We can check if the operation we did, gives an overflow and if yes, we must SET the “LED” flag with the following code:

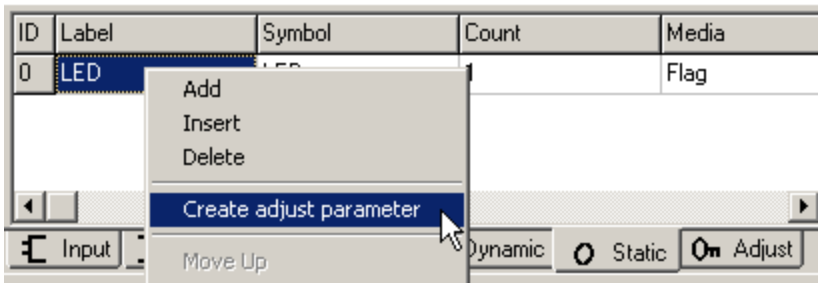
```
                ACC     E
                JR      L NoError
                SET     LED

NoError:
```

The “ACC E” line place the accumulator to high if an error occurs or to low if everything is normal.
The second line “JR L NoError” means that if the accumulator is low jump to the “NoError” label otherwise execute the “SET LED” code line.

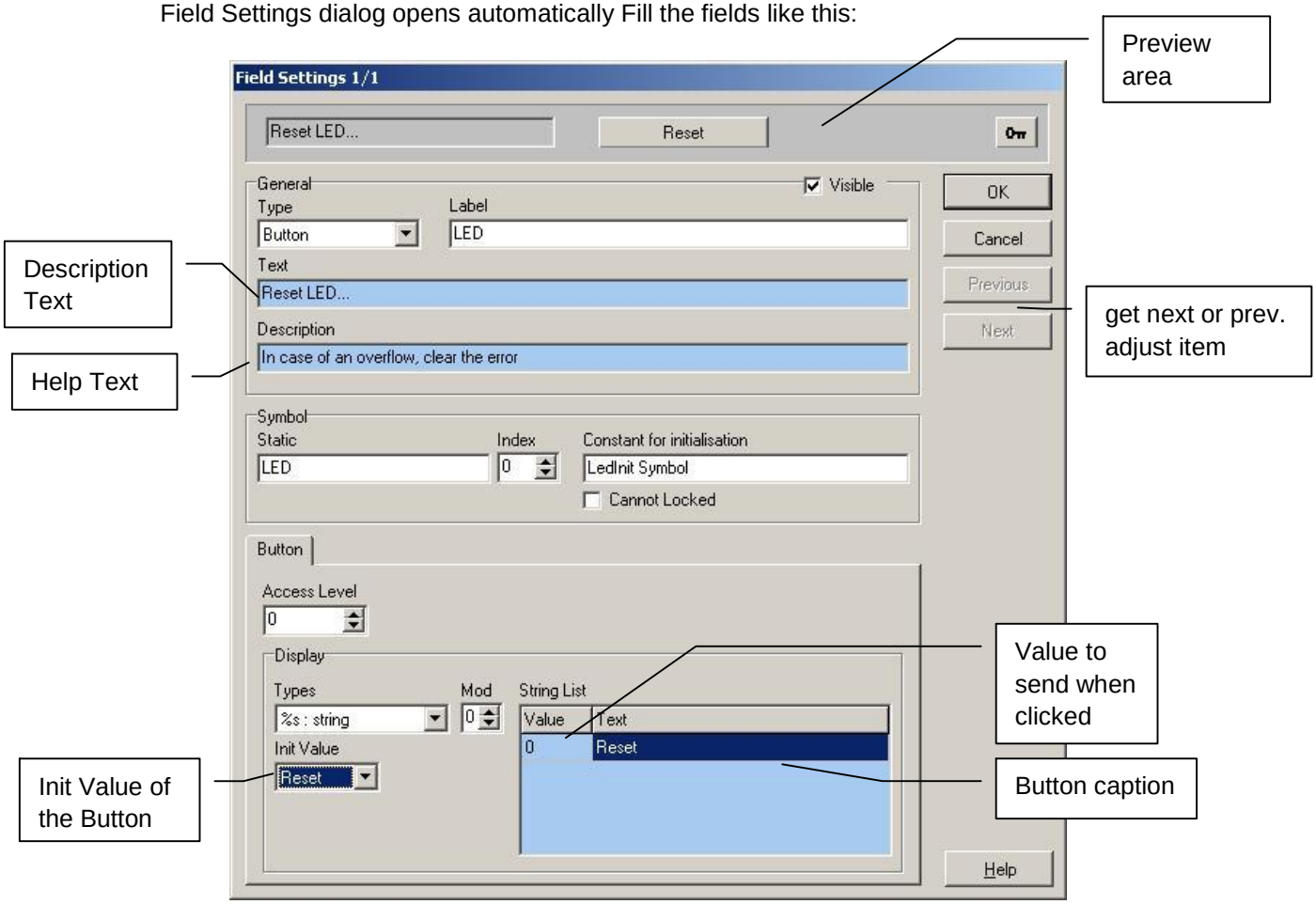
In case of an overflow, now the LED of the FBOX will be red. The only way to make it green again is to reset the “LED” flag. Usually to do so, the best way is to add a “Reset” button in the FBOX Adjust and View dialog window. The Adjust and View variables will be treated in the next chapter called “FBOX Advanced”. Anyway we will finish that FBOX and introduce you at the same time without going in the details into the Adjust and View Variables.

4.5.3.2 Add a LED “Reset” Button

To add a button, we must add an Adjust variable in the “Parameter Editor”. The easiest way is to use "Create adjust parameter" context menu item in Static tab.



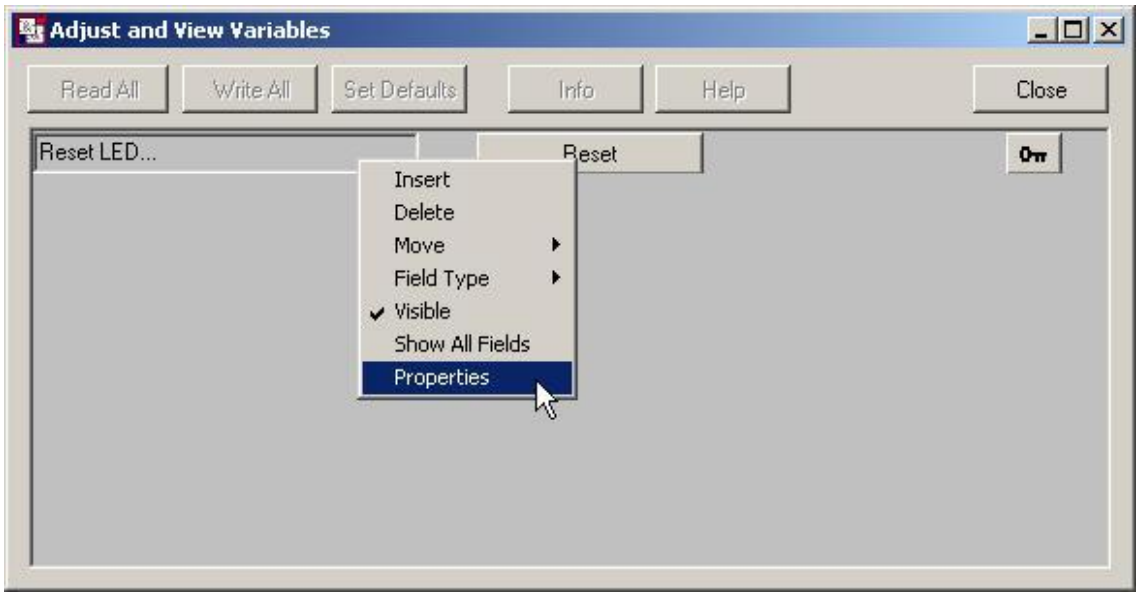
Field Settings dialog opens automatically Fill the fields like this:



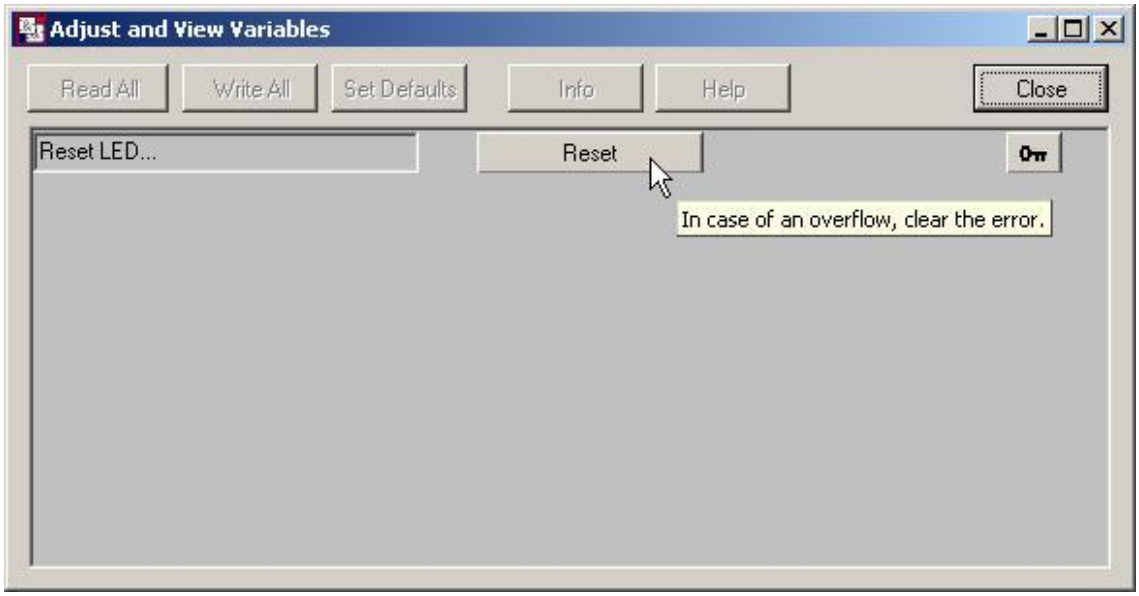
Check the parameters in and fill the remaining fields

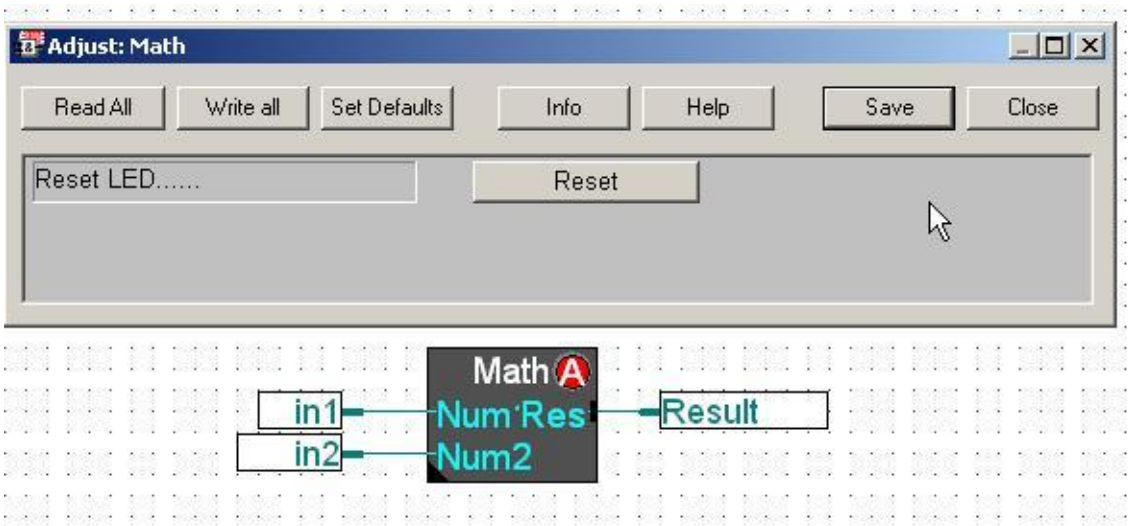
ID	Label	Symbol	Type	Visible	Default Symbol	Comment
0	LED	LedInit Symbol	Button	Yes		Reset the LED

The first result on the FBOX face is the small black down left corner, which means if you double click on it a dialog will appear as follows:



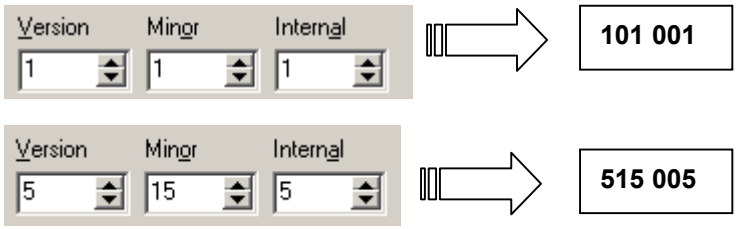
So now the result is when you are online with FUPLA and an error occurs, the LED of the FBOX will get red and you can reset that error in clicking on the “Reset Button” we just defined:





4.5.3.3 FBOX Version Handling

The version that you have defined in the “General” tab of the workspace is translated in the following format:



The range of the version number is 0 -> 599.999. Basically the number version is calculated with the following formula:

VersionNumber = {[(Version * 100) + Minor] * 1000} + Internal

To check the version of the FBOX you can use the following code lines. The two first lines have nothing to see with the version, but take it as a small tip to avoid the warning on each variable that you didn’t use in the macro when you compile your FBOX in PG5.

```
$IFNB <Name#LedAdj>
$ENDIF

$IF Vers<10001
        $Error Math: Old Fbox not anymore supported.
        exitm
$ENDIF
```

The instruction “EXITM” simply skips the remaining code and directly exit of the macro.

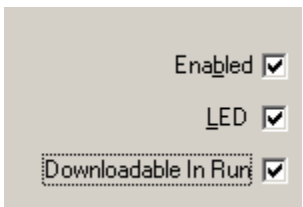
You can also check for example the PG5 version that your library or FBOX supports. It exists pre-defined symbols that give you the chance to check parameters like Library versions, PG5 version and so on. For more details on this subject, have a look at the “SAIA Instruction List Help” under “Pre-Defined Symbols”.

4.5.3.4 Using “Block” Directives

When developing FBOX, sometimes you need that a code part is executed in a certain block like special “XOB” or even if you develop an FBOX that must run with GRAFTEC you might want that a special piece of code runs in a COB.

The directives “$COBSEG” and “$XOBSEG” allows you to place a part of your code in the block of your choice.

Note: If a part of code uses an XOB for initialisation, then you should uncheck the "Downloadable in Run" checkbox on the "Global" tab in order to inform Fupla, that the project which is using that FBox, cannot be downloaded in run.

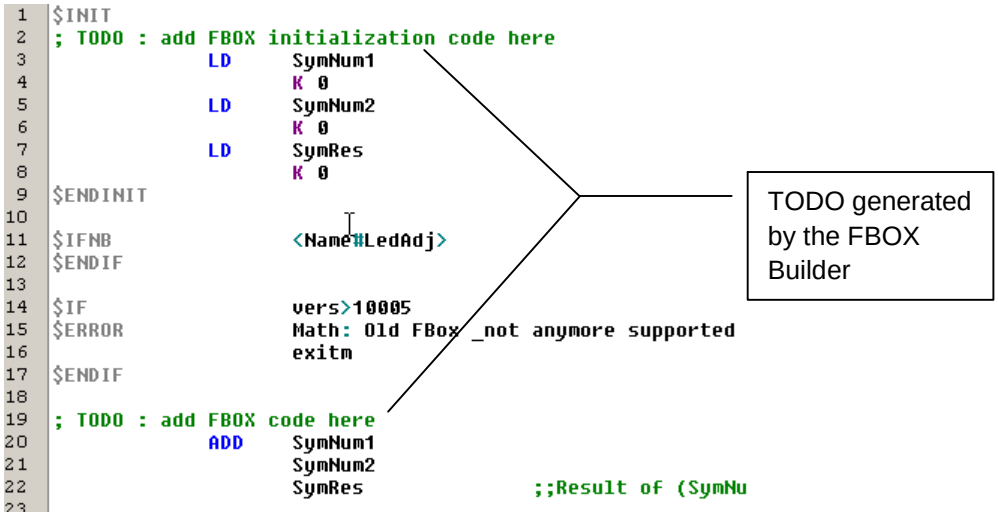


For more details on this subject, have a look at the S-Edit help.

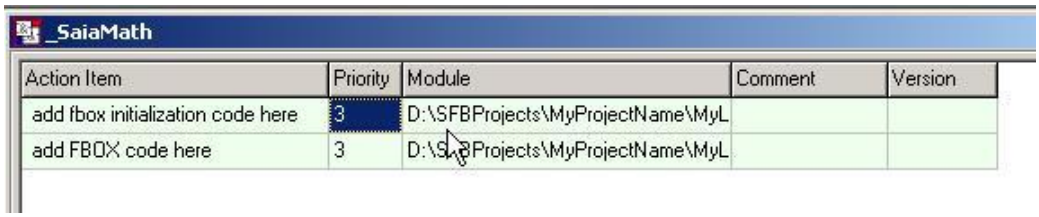
4.5.4 Using TODO and History

The TODO and History functionality of the FBOX Builder is meant to help the FBOX developers during his working process. It is simply a list of tasks that must be done and when those tasks are done the go in the History list.

In our example, there is two TODO items generated automatically by the FBOX Builder:



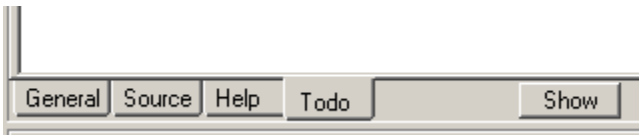
Have a look at the TODO list in clicking the Todo tab of the workspace:



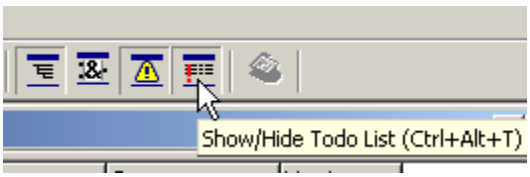
- **Action Item:** Description of what needs to be done.
- **Priority:** Level of priority that needs to be done. The levels are 1, 2 or 3.
- **Module:** The file path in which the TODO is.
- **Comment:** Sentence that belongs to this item.
- **Version:** Current version of the FBOX that is concern by this item.

4.5.4.1 The TODO types

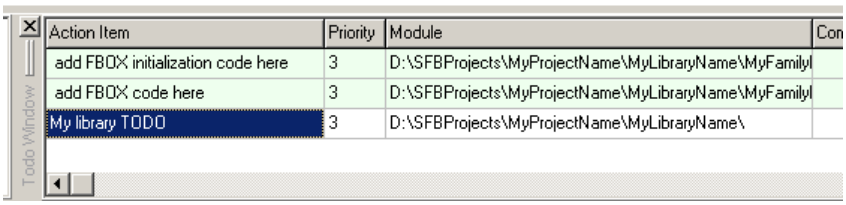
1. There is a TODO window if you click on the Todo tab in FBOX main window, it is possible to add TODO s there using the context menu.
Note : Those TODOs will appear only in this FBOX's Todo window.



2. You can use toolbar icon to display the Todo docking window. This window shows all Todo's of the FBOXes in the library



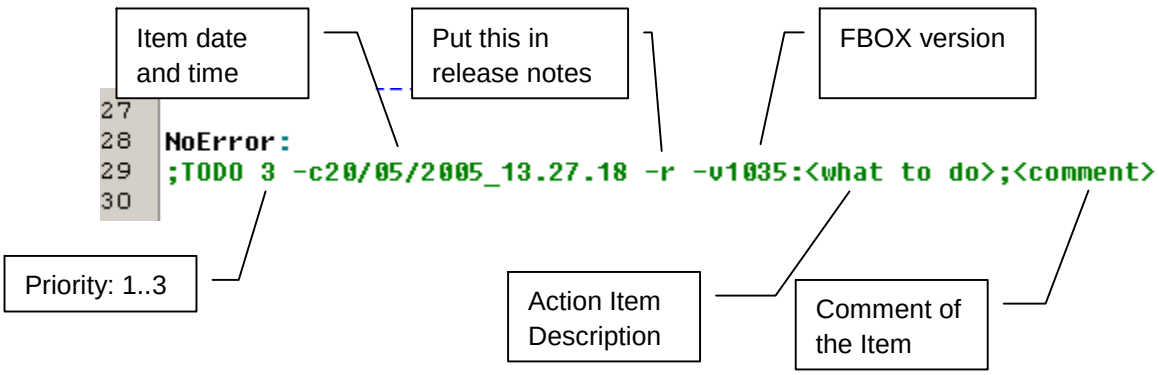
Use the context menu - *add* to add a TODO for the library.
You can check the scope under Module label



3. And it is possible to add TODOs in editor too
This item's background is light grey.

1. Call context menu with right mouse click and choose “Insert Todo” menu:

2. You will get this line in the source editor:



You just have to edit this line, as you need. The Item of the list will be updated.

Tip: Double clicking an item in the list, brings you to the item in the editor.

4.5.4.2 What about History items

As described in the previous lines, a TODO that is marked as DONE becomes a history item.
To mark a TODO Item DONE, you have to right click on the TODO line in editor or in Todo window to call context menu and select "Done".

Our example will look like this after marking the both TODO as DONE:

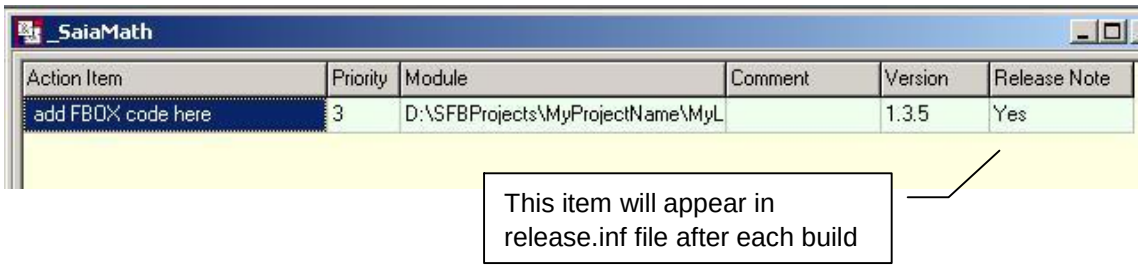
```
;DONE -c20/05/2005_13.44.48 -r -v1035: add FBOX code here
              ADD     SymNum1
                      SymNum2
                      SymRes                  ;;Result of (SymNu
```

The lines are now marking with the tag “DONE” with the date and time of the operation. These two items are not anymore part of the TODO list but now they are part of the history list. You can check that out the following way:

1. Click on the “Todo” tab of the workspace: 

2. To see the History list, right click in the TODO List area and choose the “Show History” menu:

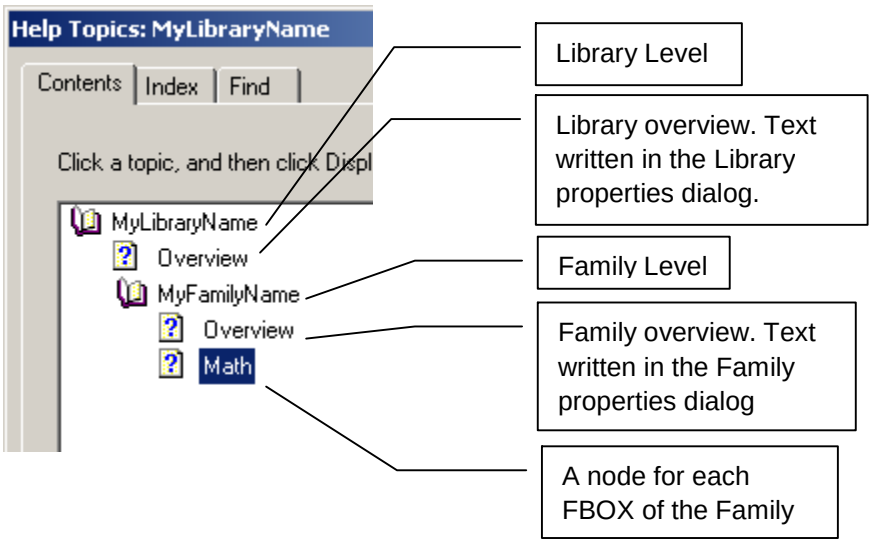
3. You get the History list as the following:



4.5.5 Create the FBOX Help Topic

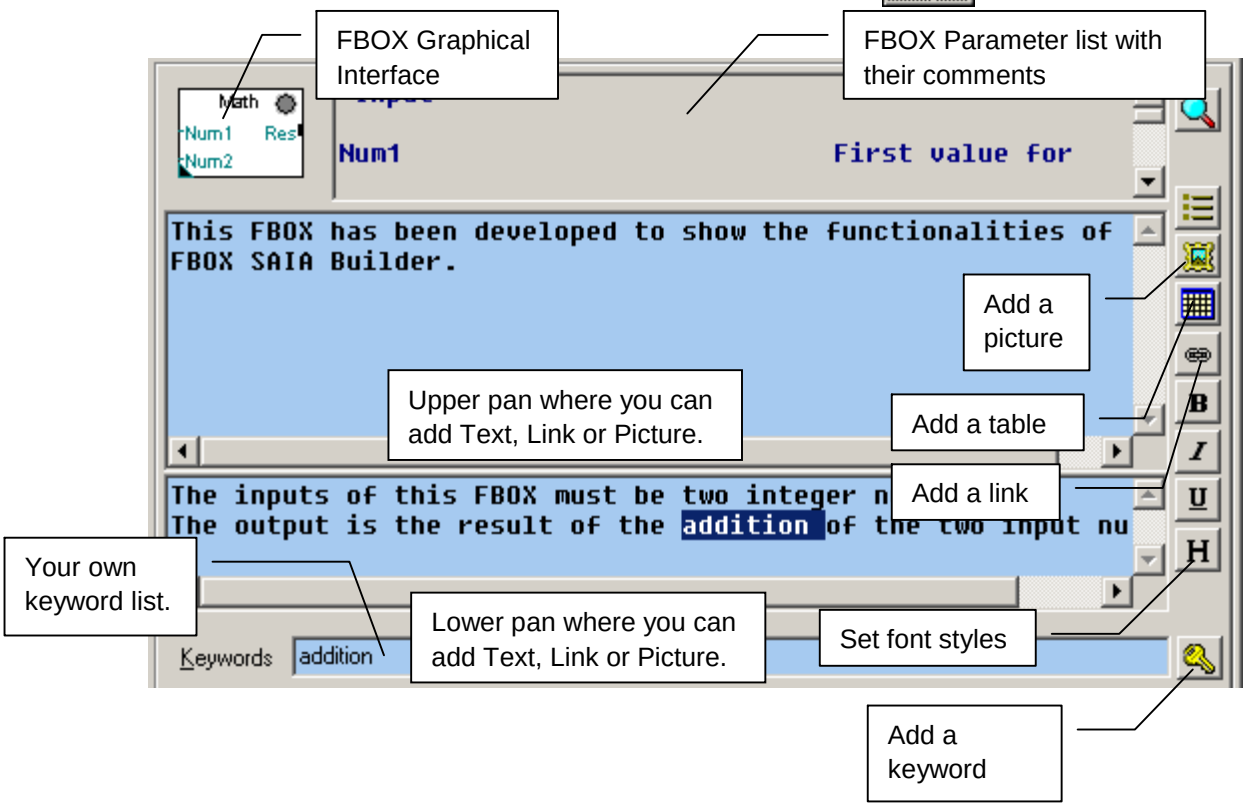
As we have seen in the previous paragraph, the FBOX Builder is able to generate a help file that belongs to the library level.

The help file has a structure that is pre-defined by the FBOX Builder, which looks like this:



The Library and the Family level are very simple, basically the only hand you have on them is the text you entered.

For the FBOX level a structure is pre-defined but you have quite a lot of possibility to customize your help. Click on the “Help” tab of the workspace: 

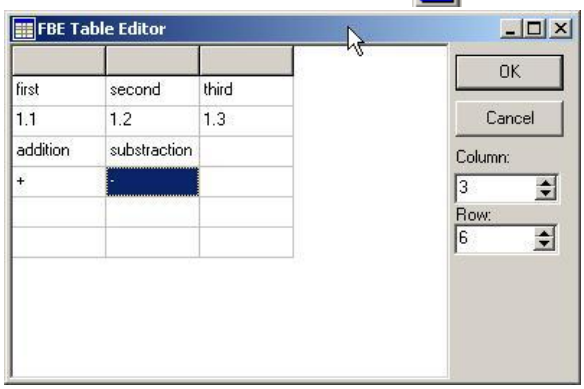


4.5.5.1 Add a New Picture

You can add the picture you want in the upper or lower part, simply place the cursor where you want insert the picture and click on the  button and then browse for your picture.

4.5.5.2 Add table

You can add tables in the help too. Use  button to call Table Editor



It looks like this in the editor and like this in the help:

```
{table
first   second  third
1.1     1.2     1.3
addition        substraction
+       -


\table}
```

first	second	third
1.1	1.2	1.3
addition	substraction	
+	-	

4.5.5.3 Add a new Link

There are links that are automatically generated like the link between the FBOX and the Family or between the Family and the Library.
But you can add your own links between what ever you want. To do so, you have to write text you want to display as link, select it, click on the  button and finally replace the "<MacroName>" to topic ID you want link to.
The topics ID are defined as follow: **IDH_Keyword**
For example the link to our library looks like this**: IDH_MyLibraryName**

4.5.5.4 Font style buttons

Select the text you want to format and press:

 for Bold  for Italic  for underlined  for Header style

4.5.5.5 Add a new Keyword

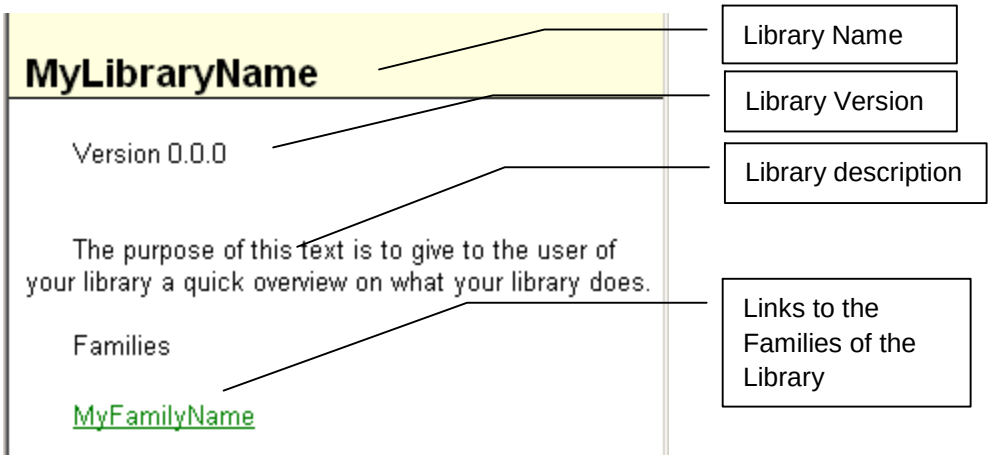
The default keyword is the Macro Name and the FBOX Name, one of these is used by FUPLA to call the right help topic according to the current FBOX.

But you can add some other keyword that you can use inside the help to make your own link to it. To add a new keyword, you can directly type in the keyword edit field or you can select a word in the help editor and then click on the  button.

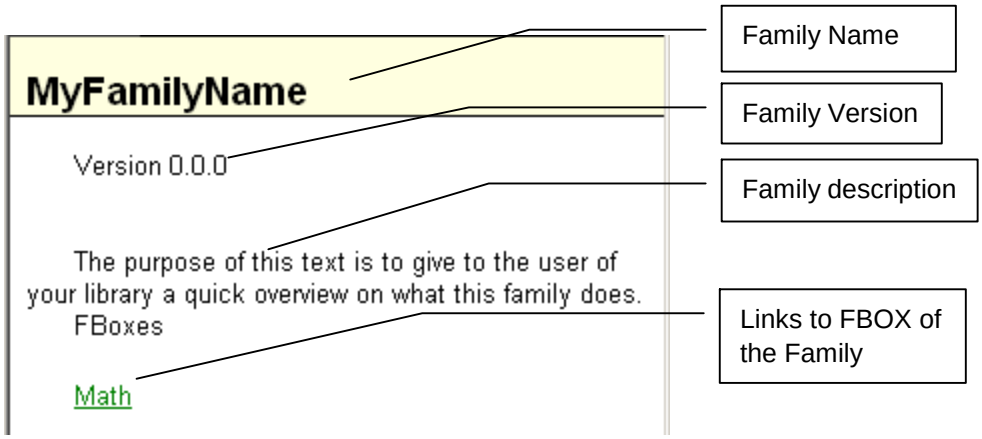
4.5.5.6 Help Files Pre-defined structure

Each level of the Help has also a pre-defined structure. In the following picture, you will see the look of each level:

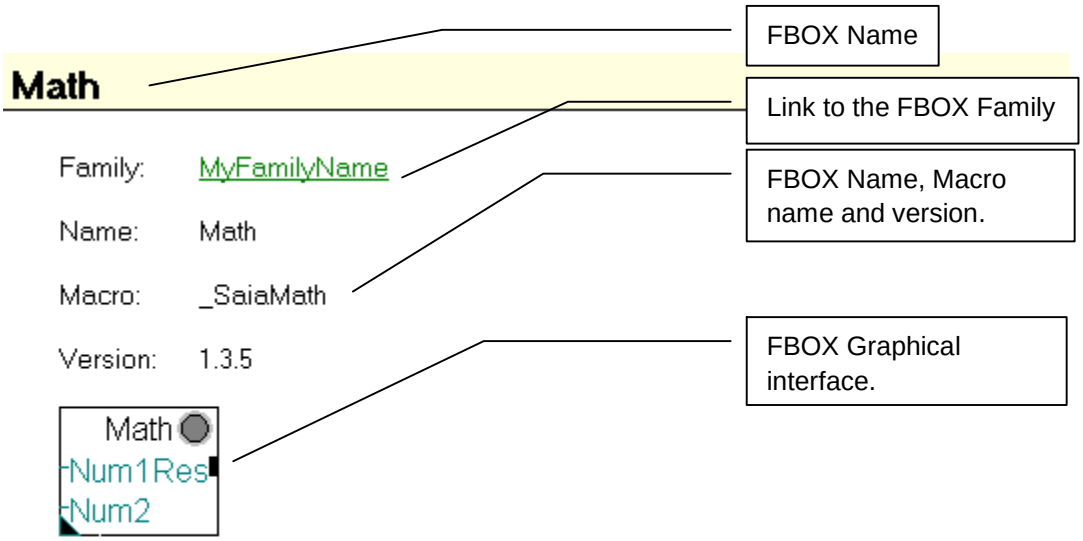
1. The Library overview:



2. The Family overview:



3. The FBOX description:



Description

This FBOX has been developed to show the functionalities of the SAIA FBOX Builder.

The inputs of this FBOX must be two integer numbers.
The output is the result of the addition of the two input numbers.

first	second	third
1.1	1.2	1.3
addition	substraction	
+	-	

Input

Num1 First value for our math operation

Num2 Second value for our math operation

Output

Res Math operation result

4.6 Generate Output Files

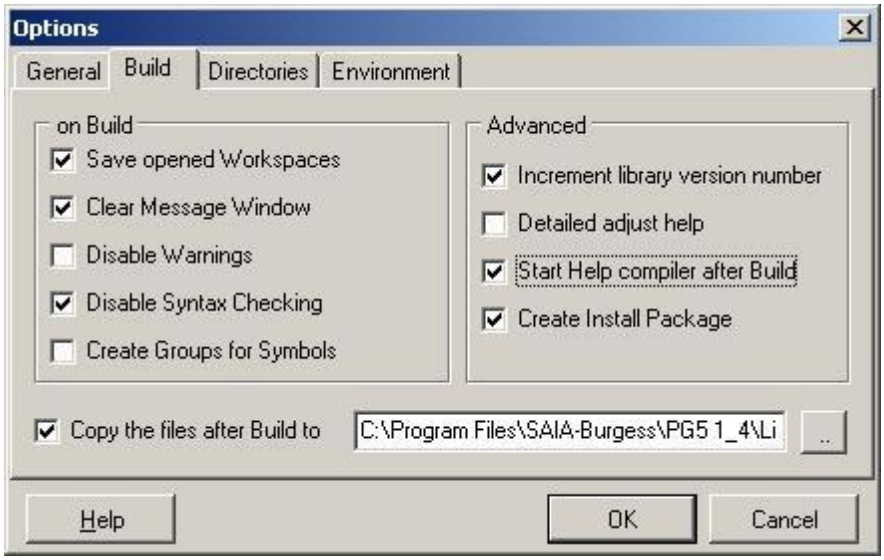
Now that everything is ready, the FBOX functionality is defined, the help files also, we can generate all the files needed by FUPLA to fully run our FBOX.

There are two different processes that you must start, the build of the FBOX Library itself and the build of the help file.

You start the first process in clicking on the tool bar button: 

You can start the second process in clicking on the tool bar button: 

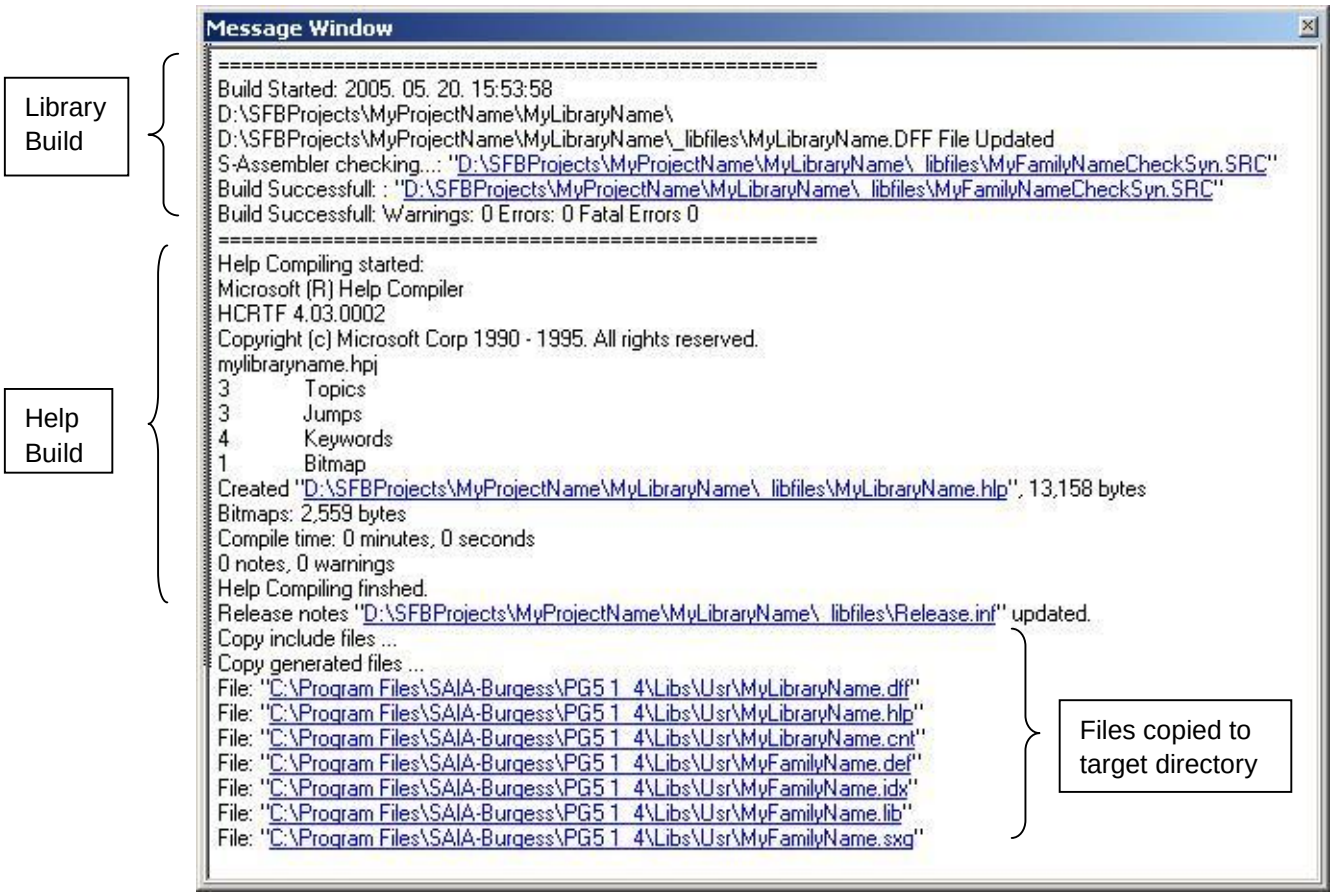
But you can also select the option “Start Help compiler after Build” (View -> Option)



Before generating the files, you should select also the “Copy the files after Build to ” and select the “Usr” directory of the PG5 library folder. If this option is not selected, you’ll have to copy by hand all the generated files in this directory to be able to use your FBOX in FUPLA.
The "Create Install Package" results an installable executable for installing the FBOX library.

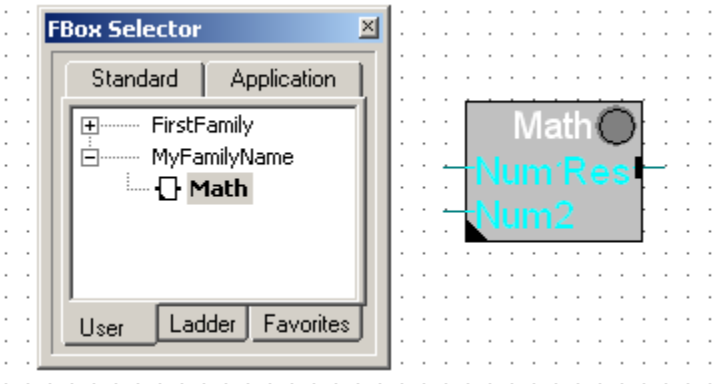
If the option “Disable Syntax Checking” is checked, it means that only the FBOX Builder checks the FBOX definition; otherwise the PG5 assembler checks each FBOX.

So after the Build process is terminated, you get the description of what happened during this process in the message window. You should get the following messages:



Tip: To open a file listed in the Message window just double click on it.

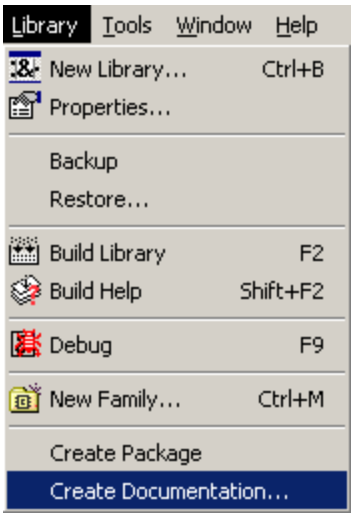
Finally open FUPLA, select the “User” library, the “MyFamilyName” family and the “Math” FBOX, you get the following result:



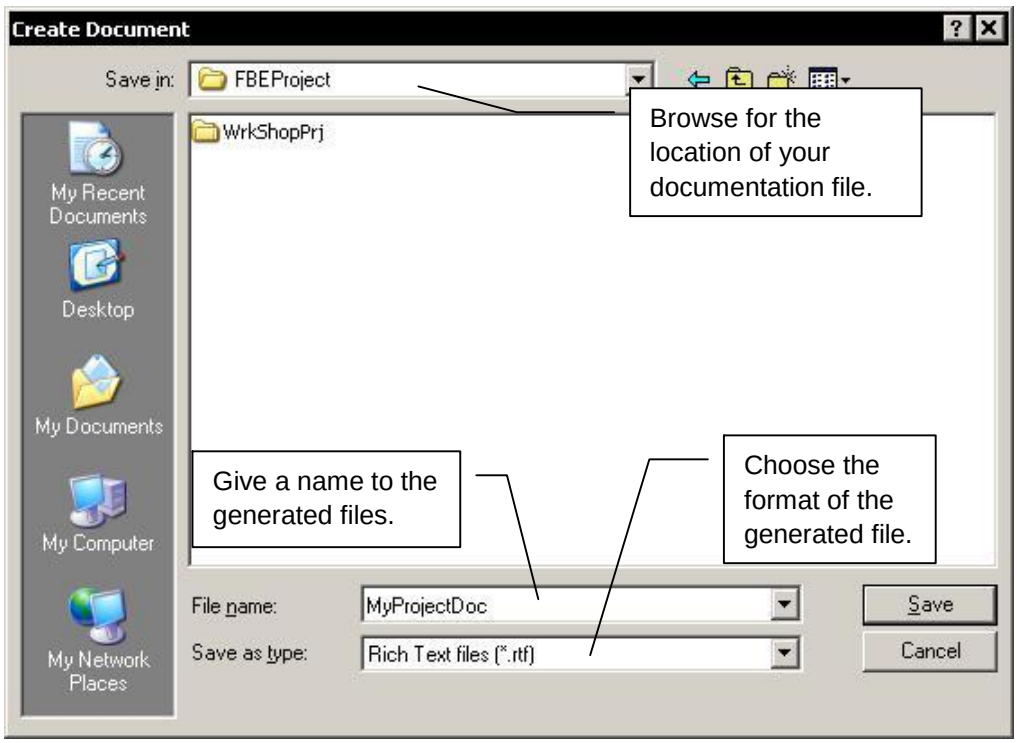
4.7 Documentation Generator

The FBOX Builder allows you to generate the documentation that belongs to your project. You have the choice between generating your documentation in RTF or in PDF format.

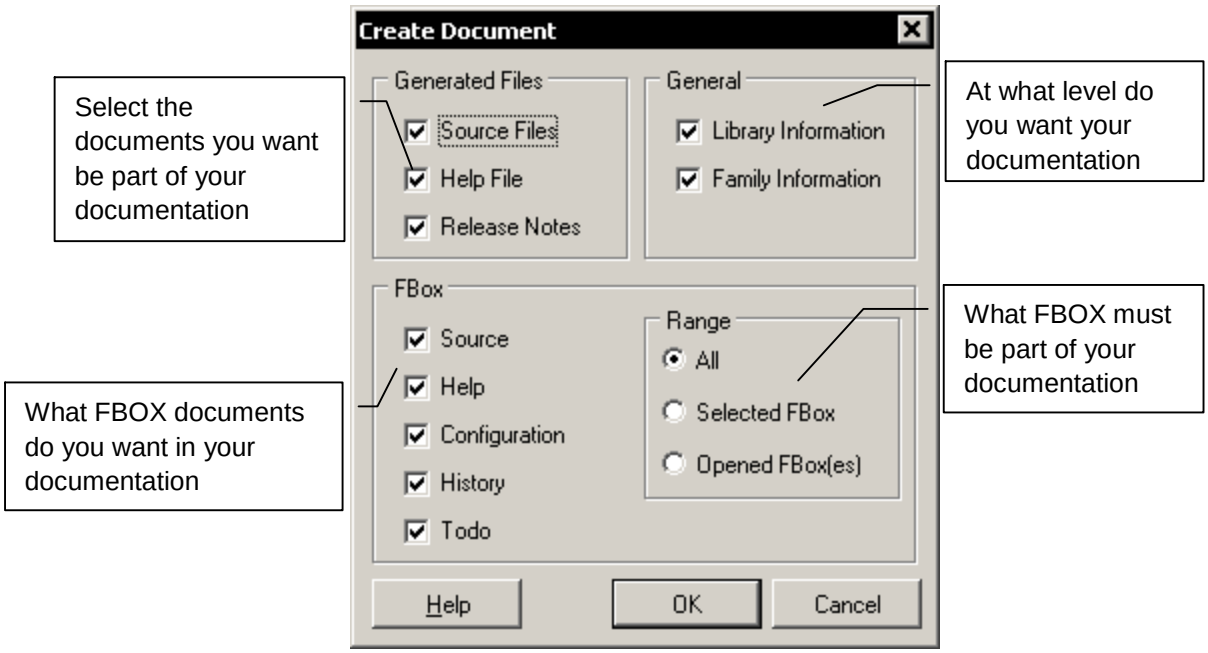
To generate your documentation, you just have to select the “Create Documentation…” sub-menu in the main “Library” menu.



You get the following dialog:



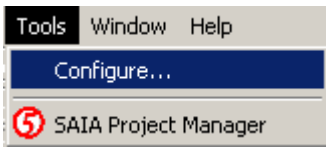
After having clicked on the save button, the following dialog popped up:



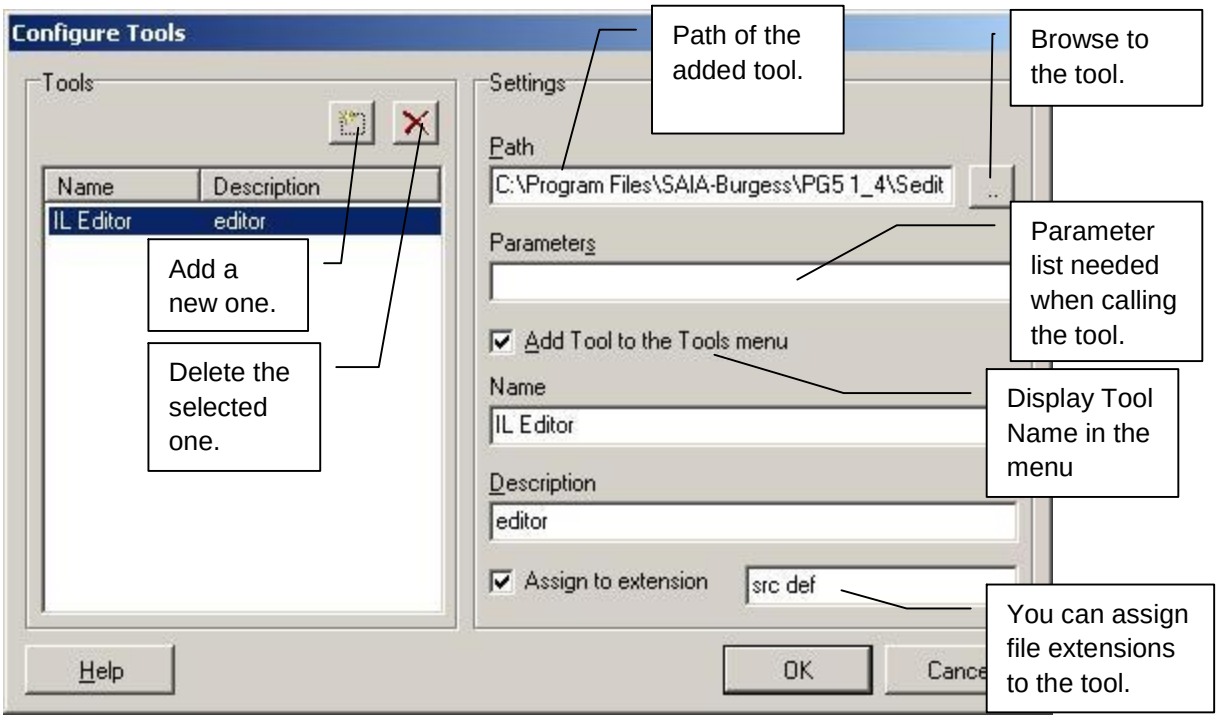
4.8 Add-Tools to FBOX Builder

FBOX Builder gives you the opportunity to add your own tool in its environment. For example you can add the Text Editor of your choice and for example configure the extension that belongs to it.
As example, we will add S-Edit to our environment:

1) Call the configuration dialog in clicking on the following menu:



You get the following dialog window:



2) Now when inside the FBOX Builder you double click on a “SRC” file, S-Edit will be called to display this file.

To add more than one extension to an add-on tool, you just have to add it more than once with different extensions but with the “Add Tool to the Tools menu” checkbox unchecked.

Contents

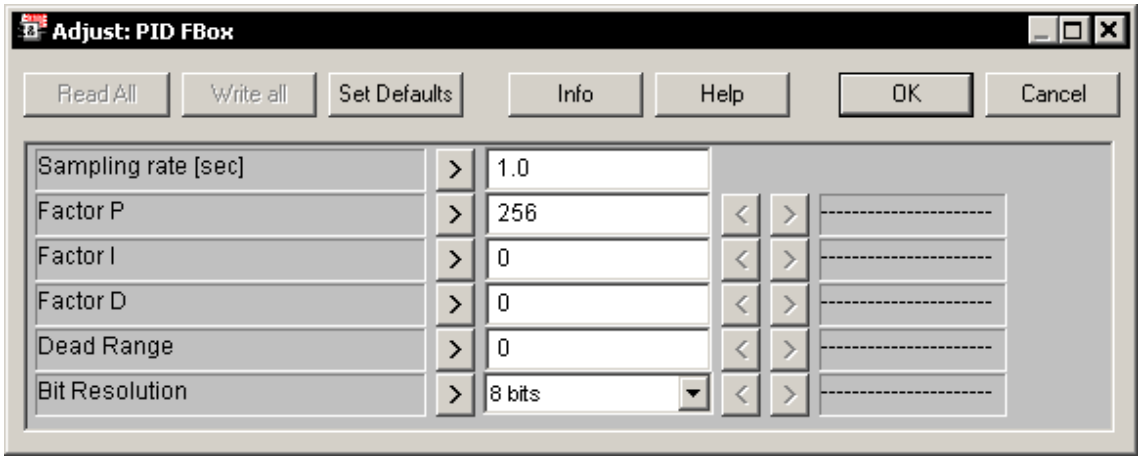
5 FBOX Advanced

In this chapter you will see with an example the advanced features of the FBOX Builder, the basics was treated in the previous chapter.
To avoid symbols naming clash with existing FBOX libraries, you can contact SAIA-Burgess Controls Call Centre to get a list of registered symbols prefixes that you must not use; or if you plan to develop and distribute in a large scale your own library, you can apply at the SAIA-Burgess Call Centre to register your own prefix.

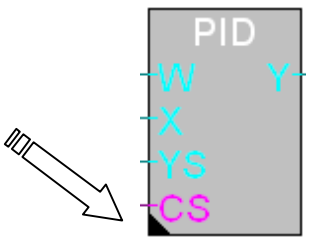
SAIA-Burgess Controls Call Centre: pcdsupport@saia-burgess.com

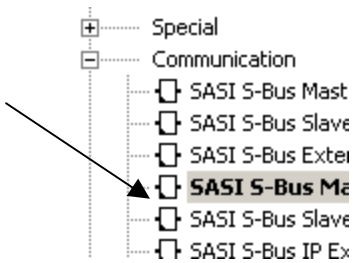
5.1 Adjust and View Variables

The Adjust and View variables are maybe the most complicated part of the FBOX Development but also the most powerful feature. The FBOX Builder greatly supports you to make full using of the huge possibilities that is given to you with the Adjust and View Variables.
This kind of variables gives a chance to the FBOX user to modify the some FBOX parameter before compiling the FUPLA program or even during the run of the PCD program. When declaring an Adjust and View variable, the FBOX has a dialog window that belongs to it. Each line of this dialog corresponds to one Adjust and View variable. Here below you have an example of what could be on this dialog window:



An FBOX which has a dialog window attached to it has its down left corner black as shown below it is displayed in FBOX selector too:





You can open the dialog in double clicking on the FBOX.

There are several types of adjust variables as described below:

Online View variables are static variables that can be visualised during the execution of the PCD program. They can be Flags, Timers, Counters or Registers.

Online Adjust variables are static variables that can be modified and visualised during the execution of the PCD program. They can be registers, flags, counters or timers.

The FBOX user when programming in FUPLA initialises **Offline Adjust variables**. This kind of variables does not need any static variables because they are passed as constant to the FBOX macro.

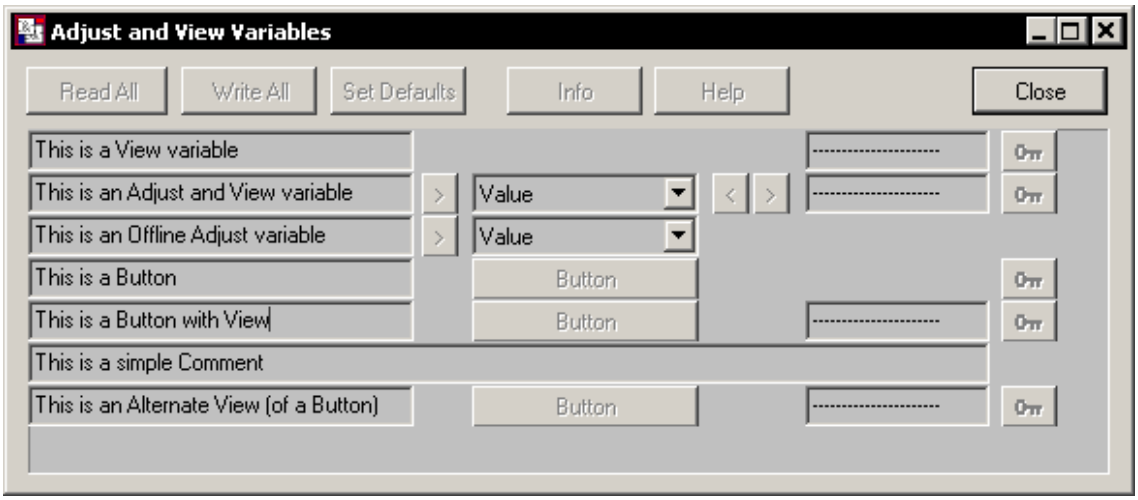
Button is associated to static variable; they can be registers, flags, counters or timers. When a user presses an FBOX button, FUPLA transfers its default value to its corresponding static variable.

Button with View is button like described in the previous lines, the only difference is that a View belongs to it to visualise the content of the static variable that is link to the button in question.

Alternate View variable is used to show in the view window a different variable as the one, which was previously defined in the current adjust variable. This feature is exclusively used today for button with view.

Comment is used in an Adjust Dialog window to comment or create sections in a list of adjust and view variables. Double clicking on these fields gives the description associated with the variable.

Here is how those variables are represented in the dialog window:



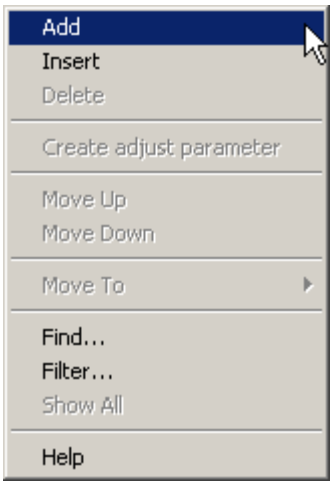
After this quick overview of the possibilities, it's time to go in the details of each variable type.

5.1.1 Online View variables

It's now time to make our example FBOX called "Math" a little bit more efficient. We can for example add three View variables to have a look at the numbers of the mathematical operation, entry numbers and the result. As reminder, our example does just an addition of two integer numbers.
As described in the previous paragraph, an online View variable is linked to a static variable; the best way to add an online View variable to our FBOX is to first add a static variable and based on this one create a View variable as follow:

1. Select the "Static" tab of the "Parameter Editor" 

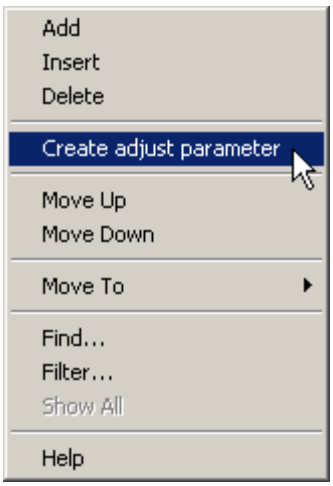
2. Right click on the "Parameter Editor" and select the "Add" menu:



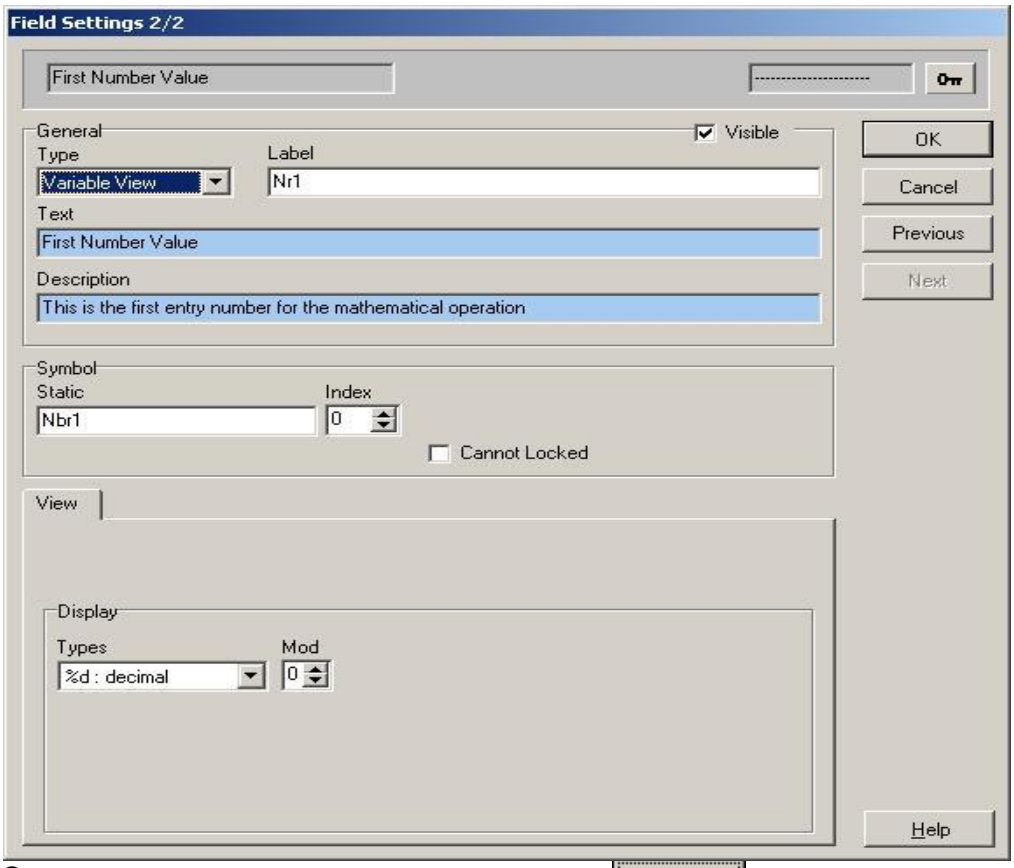
3. Enter the following lines:

1	Nr1	Nbr1	1	Register	Num1Static	Static register of the first number
2	Nr2	Nbr2	1	Register	Num2Static	Static register of the second number
3	Res	Result	1	Register	ResultStatic	Static register of the result

4. For each of these lines, make an Adjust and View variable in right clicking on each line and select the "Create adjust parameter":



5. Each time the Field settings dialog is opening. We must complete the properties dialog of each View variable:



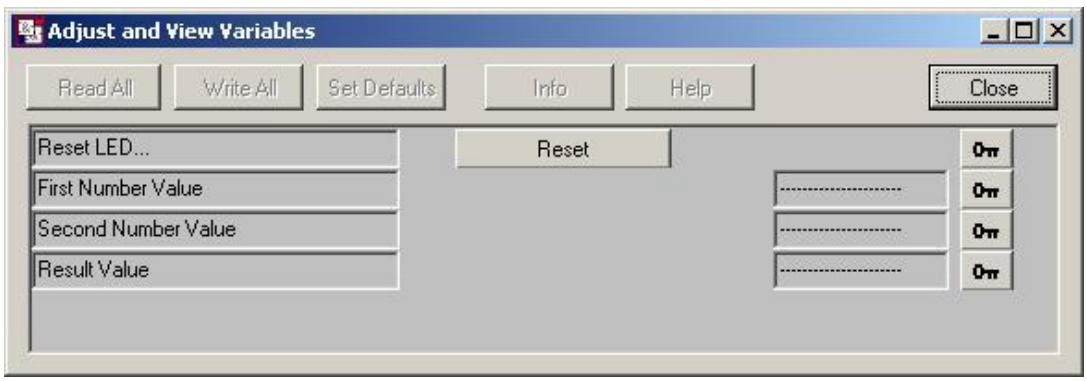
6. C
lick on the “Adjust” tab of the “Parameter Editor”

7. Complete the adjust lines as follow:

1	Nr1	Nbr1Adj	Variable View	Yes	View variable number 1
2	Nr2	Nbr2Adj	Variable View	Yes	View variable number 2
3	Res	ResAdj	Variable View	Yes	View variable result

You can see that the links with the static variables are labels Nr1, Nr2 and Res that must be the same as the static one.

If you double click on the FBOX graphical interface, you get this dialog:



The “LED” was defined in the previous chapter.

8. Now that everything is defined, you have to write the code that is going to write the values in these static registers we have just declared. Go in the source editor, the macro header looks like this:

```
_SaiaMath MACRO     Vers,                ; Version number
                    SymNum1,             ; First value for our math operation
                    SymNum2,             ; Second value for our math operation
                    SymRes,              ; Math operation result
                    LED,                 ; LED static symbol
                    Nbr1,                ; Static register of the first number
                    Nbr2,                ; Static register of the second number
                    Result,              ; Static register of the result
                    LedInit,             ; Reset the LED
                    Name                 ; FBox Name
```

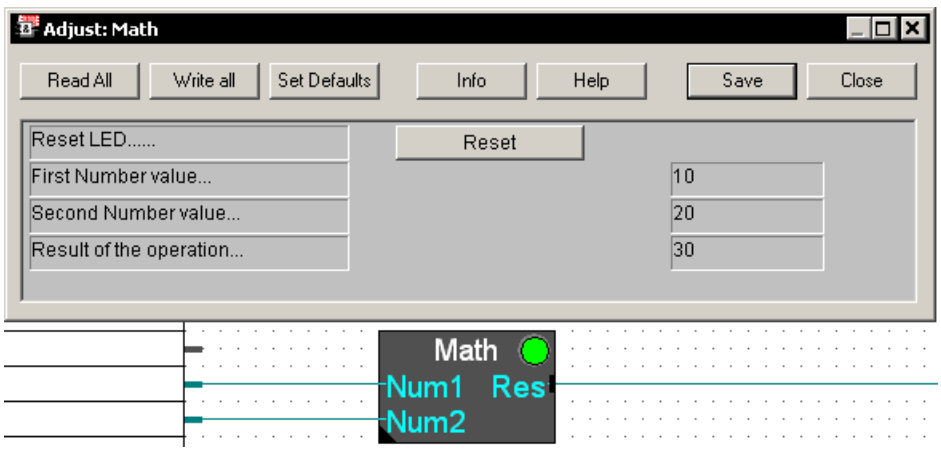
View Variables

Copy each number in the View variables as follows:

```
19  ;done -c20/05/2005_13.44.48 -r -v1035: add fbox code here
20                  ADD     SymNum1
21                          SymNum2
22                          SymRes              ;;Result of (SymNum1 + SymNum2)
23
24
25
26  ;;Copy in the View Variable static registers
27                  COPY    SymNum1
28                          Nbr1
29                  COPY    SymNum2
30                          Nbr2
31                  COPY    SymRes
32                          Result
33
34
35                  ACC     E
36                  JR      L NoError
37                  SET     LED
38
39  NoError:
40  ;TODO 3 -c20/05/2005_13.27.18 -r -v1035:<what to do>;<comment>
```

Write the values in the static registers that belongs to the View variables.

9. You can now check the result of your work in going in FUPLA, place your FBOX, download it and go online with your PCD. In FUPLA you should get something as follows:

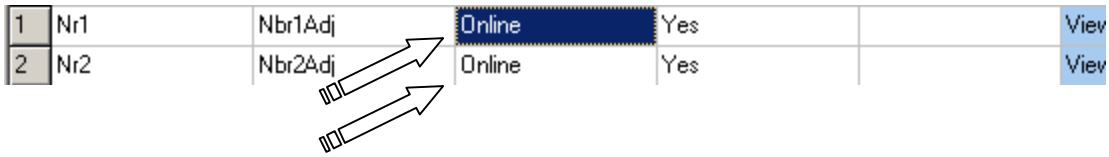


5.1.2 Online Adjust variables

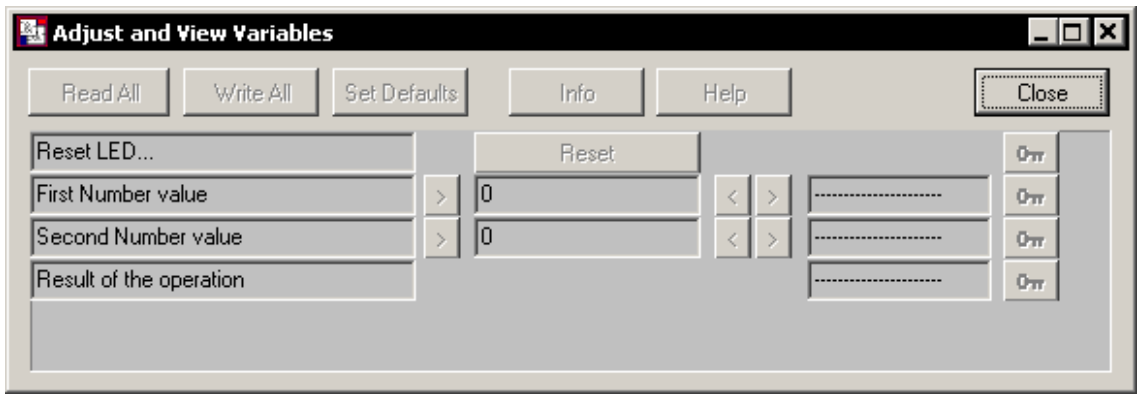
In the previous paragraph called “Online View variables”, we have described how you can link static to View variables to display them in the Adjust and View dialog window. In this chapter do not want just to be able to visualise the variables, but also to be able to modify their values. Of course we don’t want to modify the result so we will just adapt the to entry numbers, in fact the modification is very simple.

1. Click on the “Adjust” tab of the “Parameter Editor” 

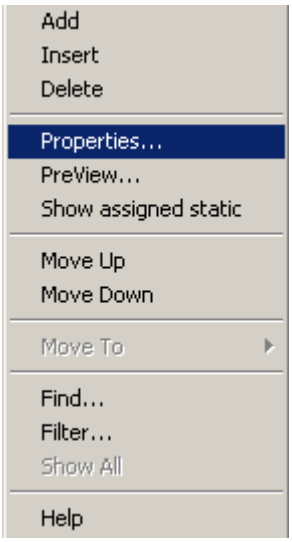
2. Modify the lines that correspond to the two entry numbers as follow:

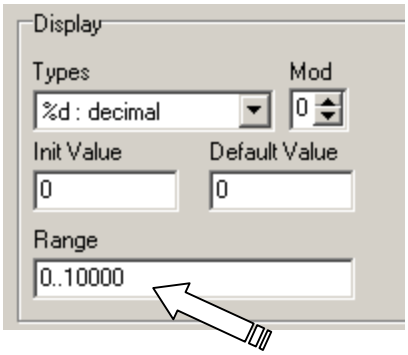


3. You get the following dialog window when clicking on the FBOX graphical interface



This is it, but you could just maybe add a “Range” of accepted value in the properties dialog of each Adjust and view Variable that you get in right clicking on the “Parameter Editor” and choose the “Properties” menu as follow:





4. You have to modify the source code of your FBOX, as you remember, we add two FBOX Inputs that was registers as entry values, to modify the entry values we must copy the values of our Adjust variables in the Input registers. The code looks like that:

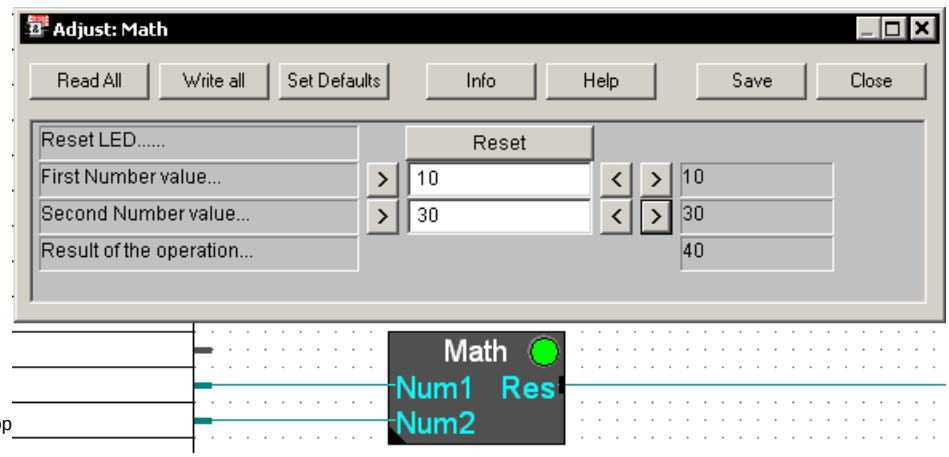
```
25
26 ;done -c20/05/2005_13.44.48 -r -v1035: add fbox code here
27                 COPY    Nbr1
28                         SymNum1
29                 COPY    Nbr2
30                         SymNum2
31
32                 ADD     SymNum1
33                         SymNum2
34                         SymRes              ;;Result of (SymNum1 + SymNum2)
35
36                 COPY    Result
37                         SymRes
38
39 ;;Copy in the View Variable static registers
40
41
42                 ACC     E
43                 JR      L NoError
44                 SET     LED
```

You should also initialise the static variables Nbr1, Nbr2 and Result during the start up of the PCD as follow:

```
 9                  LD      Nbr1
10                          0
11                  LD      Nbr2
12                          0
13                  LD      Result
14                          0
15
16 $ENDINIT
```

As you can see, the two “Inputs” of the FBOX Are useless now, we could remove them!

5. You can now check the result of your work in going in FUPLA, place your FBOX, download it and go online with your PCD. In FUPLA you should get something as follow:



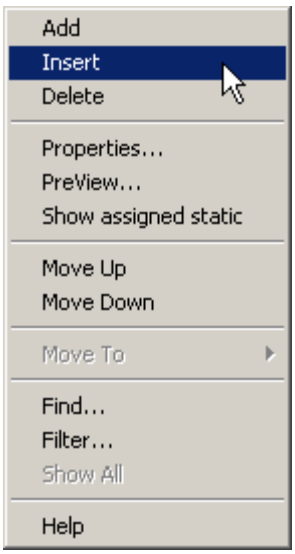
5.1.3 Offline Adjust variables

This kind of variables are not linked to a static variable but are passed as constant to the macro that belongs to the FBOX. The FBOX user has to select before the compilation of his FUPLA program the value he wants for his parameters.

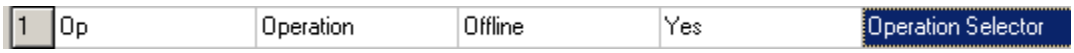
What we could do in our “Math” FBOX is to add a choice for the user. We want to add a multiplication operation. It would be better of course to be able to do this change online but we have already treated the online adjust variable in the previous paragraph.

1. Insert a new adjust variable, click on the “Adjust” 

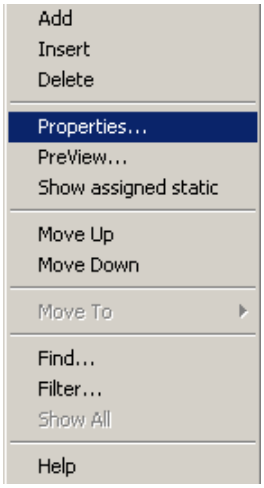
tab, select the line where you want the insertion, right click on the “Parameter Editor” and finally click on the Insert menu as shown below:

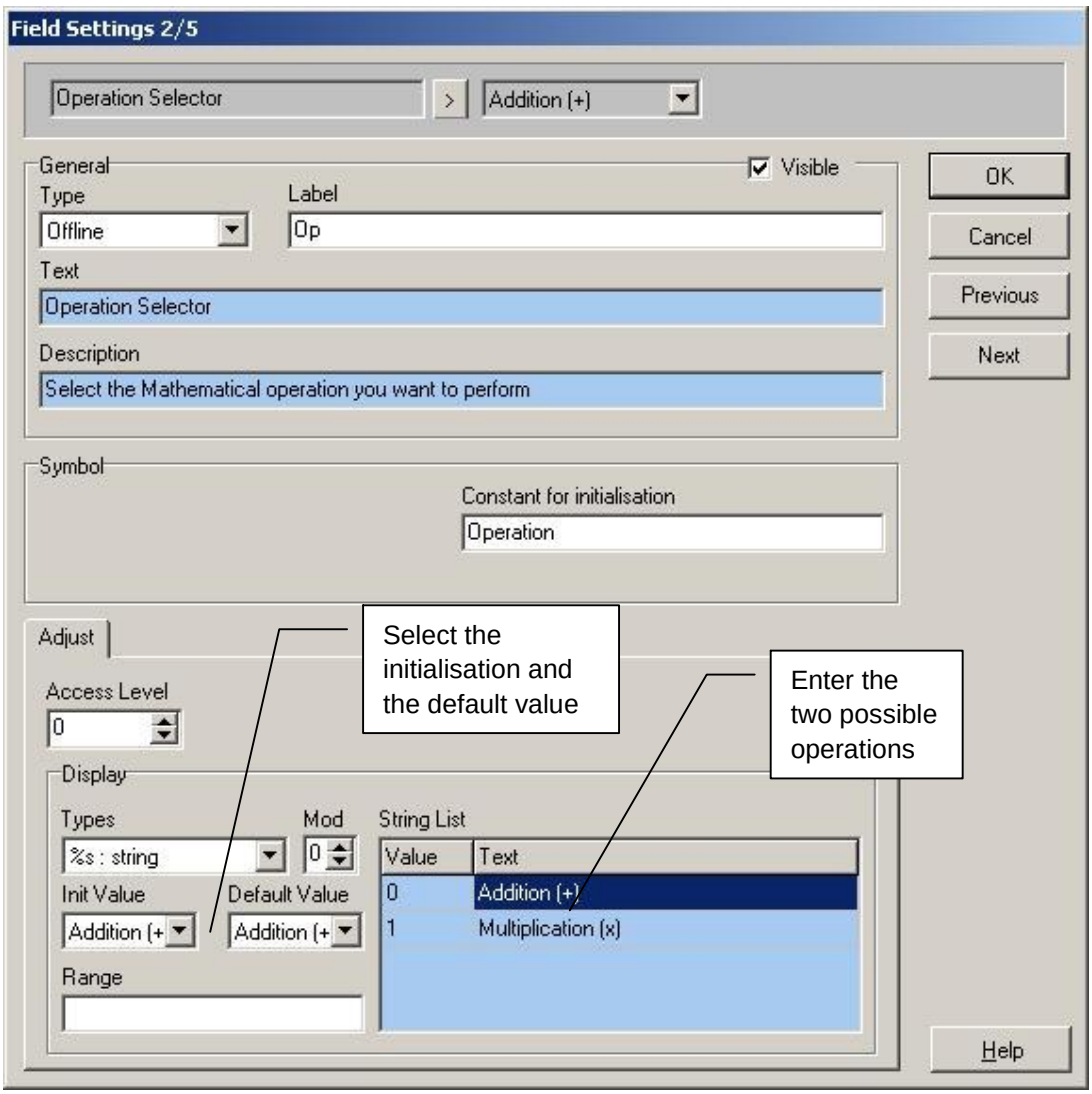


2. Enter the following line in the “Parameter Editor”:



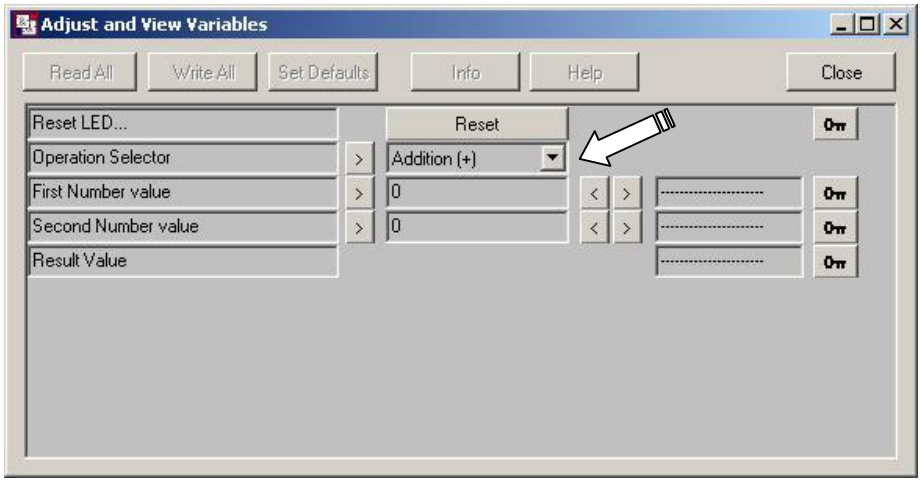
2. Edit the properties of this Offline Adjust variable, right click on it and choose the "Properties" menu





Note: If the string values are not continuous (e.g. 0, 1, 5, 6, 8) then Range field must be filled too, based on which item should be visible.

You should get the following dialog window:



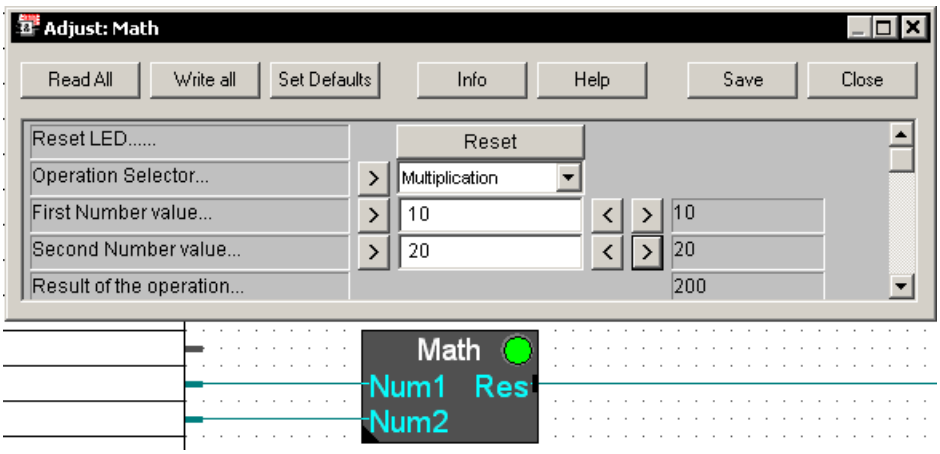
3. Now the definition is finished, we must implement the modification in the source code. The macro header looks like this:

```
_SaiaMath MACRO     Vers,                ; Version number
                    SymNum1,             ; First value for our math operation
                    SymNum2,             ; Second value for our math operation
                    SymRes,              ; Math operation result
                    LED,                 ; LED static symbol
                    Nbr1,                ; Static register of the first number
                    Nbr2,                ; Static register of the second number
                    Result,              ; Static register of the result
                    LedInit,             ; Reset the LED
                    Operation,           ; Operation selector
                    Nbr1Adj,             ; View variable number 1
                    Nbr2Adj,             ; View variable number 2
                    Name                 ; FBox Name
```

Because the choice as to be done before the compilation, we can work with the directive “$IF” to choose between one piece of code or another. Modify the source code as follows:

```
29                  COPY    Nbr2
30                          SymNum2
31
32  $IF                     Operation = 0       ;Addition (+)
33                  ADD     SymNum1
34                          SymNum2
35                          SymRes              ;;Result of (SymNum1 + SymNum2)
36  $ELSEIF                 Operation = 1       ;Multiplication (x)
37                  MUL     SymNum1
38                          SymNum2
39                          SymRes              ;;Result of (SymNum1 x SymNum2)
40  $ENDIF
41
42
43                  COPY    SymRes
44                          Result
45
```

4. You can now check the result of your work in going in FUPLA, place your FBOX, download it and go online with your PCD. In FUPLA you should get something as follows:

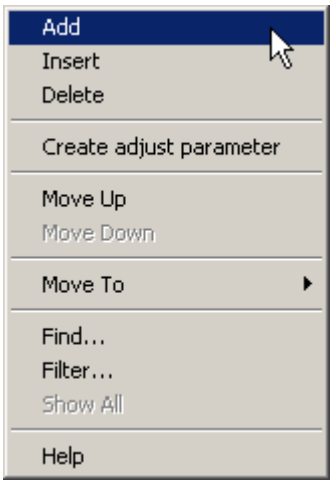


5.1.4 Button

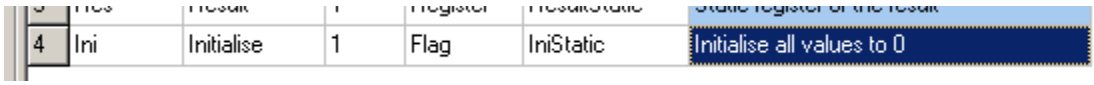
The purpose of the button is to send a pre-defined value to a static variable that belongs to it. The button is a type of an online variable so the first thing to do to define a button is to add a static variable in the “Parameter Editor”. To carry on the improvement of our “Math” FBOX, we can add a button to it to initialise every operation register, Num1, Num2 and the Result.

1. Add a static variable, click on the “Static” 

tab of the “Parameter editor”, right click in the “Parameter Editor” and choose the “Add” menu.

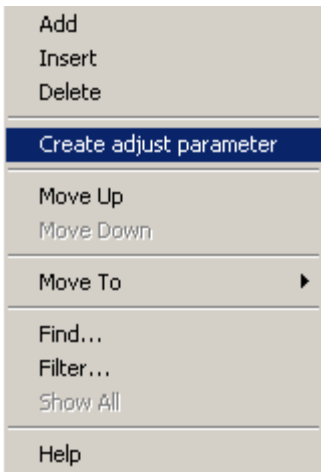


2. Complete the added line as follow:

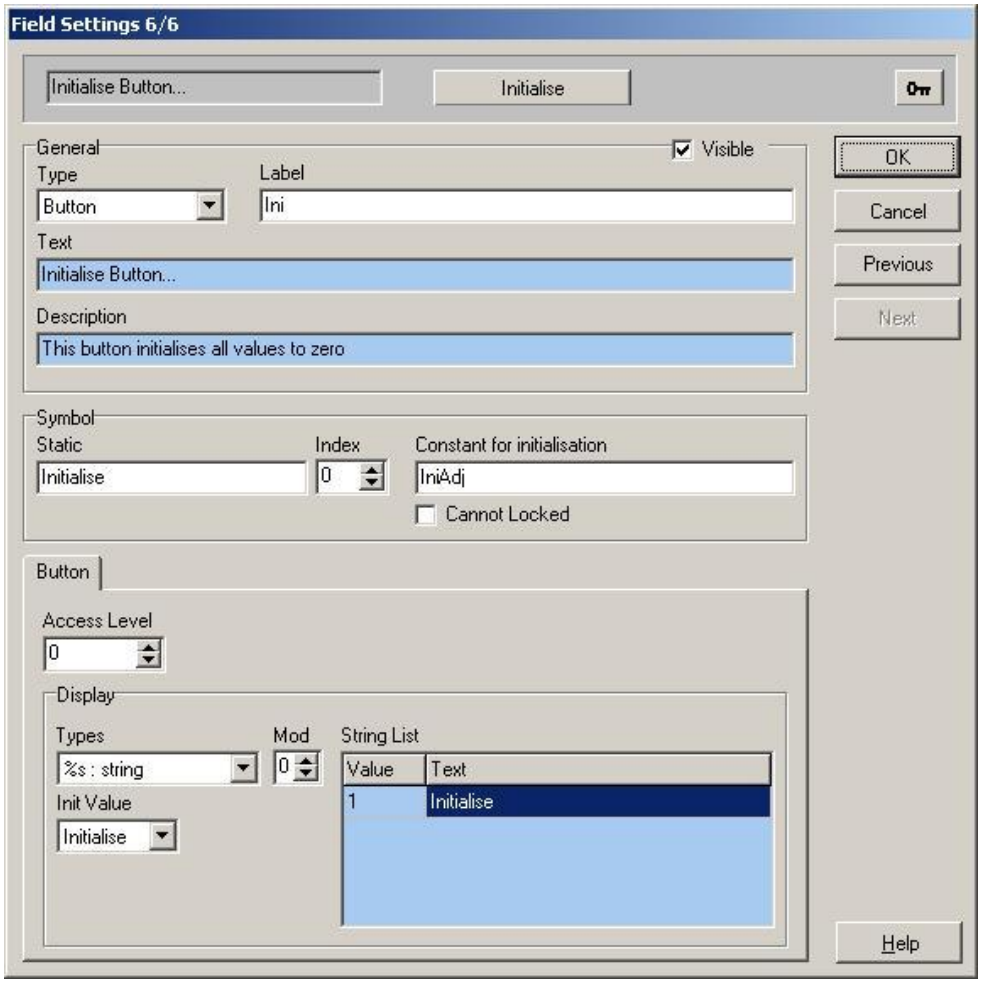


4	Ini	Initialise	1	Flag	IniStatic	Initialise all values to 0

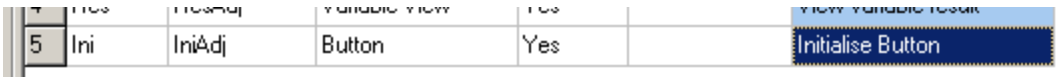
3. Based on this static variable, create the belonging Adjust variable in right clicking on the line you just added and select the “Create adjust parameter” menu.



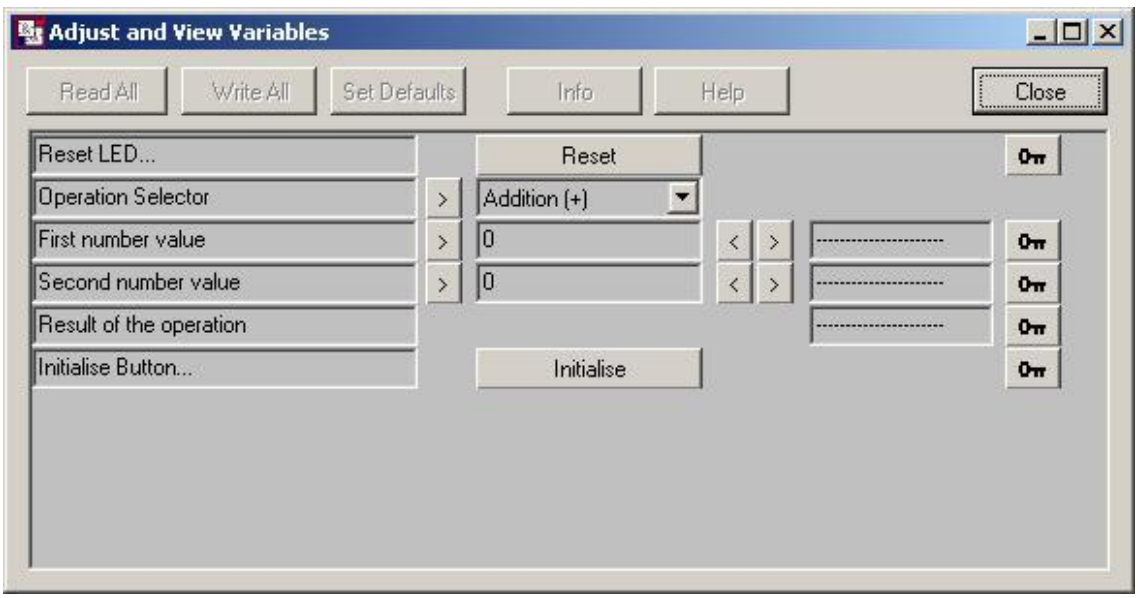
4. Fill up the properties dialog of this Adjust as follows:



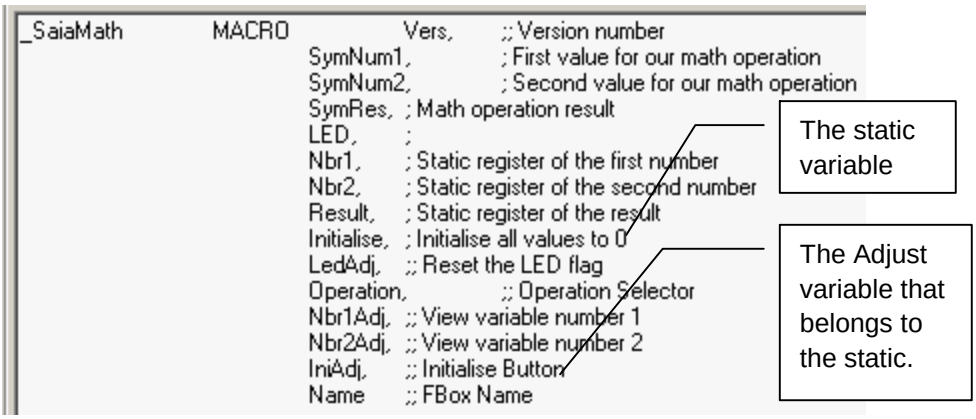
5. Modify the Adjust variable as follow:



6. You should get in double clicking on the FBOX graphical face the following dialog window:



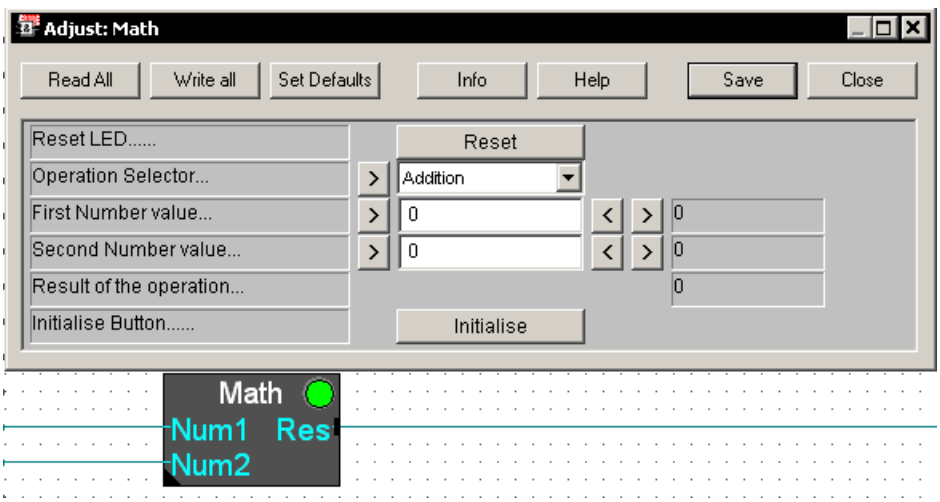
7. Now that everything is defined, we have to add the code to implement this functionality. The macro header looks like this:



8. Modify the source code as follows:

```
43                COPY    SymRes
44                        Result
45
46  ;;Initialise all registers on Init Button press
47
48                STH     Initialise
49                JR      1 DoNotInit
50                RES     Initialise
51                LD      SymNum1
52                        0
53                LD      SymNum2
54                        0
55                LD      SymRes
56                        0
57                LD      Nbr1
58                        0
59                LD      Nbr2
60                        0
61                LD      Result
62                        0
63  DoNotInit:
64
```

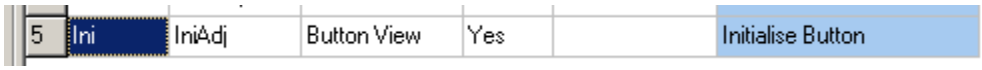
9. You can now check the result of your work in going in FUPLA, place your FBOX, download it and go online with your PCD. In FUPLA you should get something as follow:



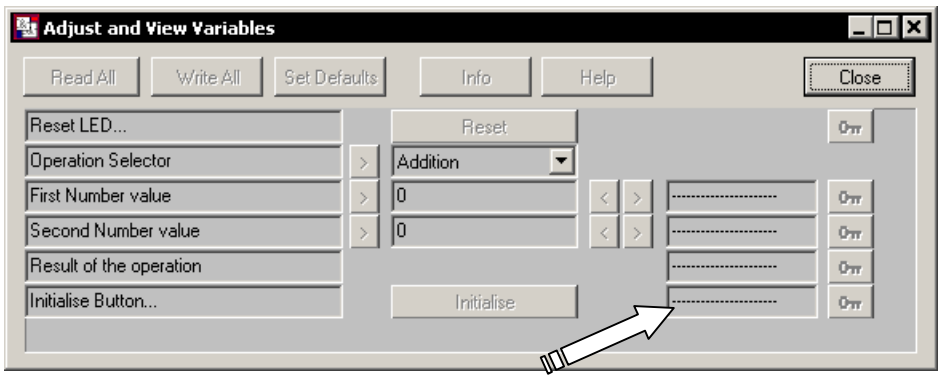
5.1.5 Button with View

There is a possibility to add to a button a View field to visualise the value of the static variable that belongs to the button. In our example, we can modify the button we have created in the previous paragraph as follow:

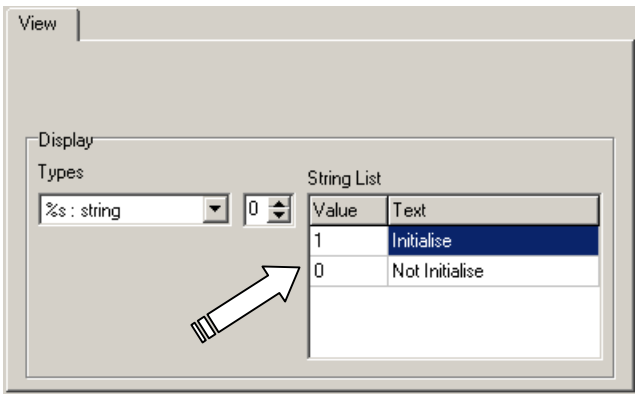
1. Modify the Button adjust line as follows:



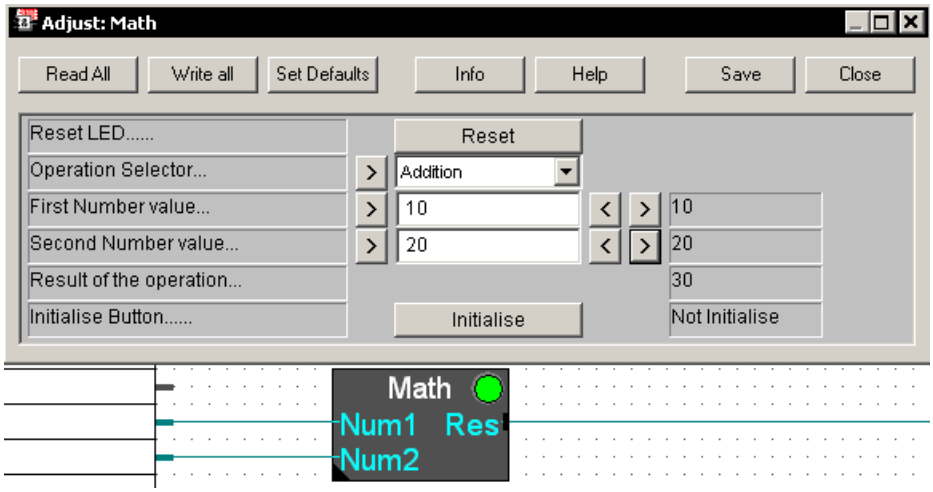
2. You should get the following dialog window:



3. Add a string corresponding to the “0” state of the static flag that belongs to the Button with View. Open the properties dialog window and modify it as follows:



4. You can now check the result of your work in going in FUPLA, place your FBOX, download it and go online with your PCD. In FUPLA you should get something as follow:

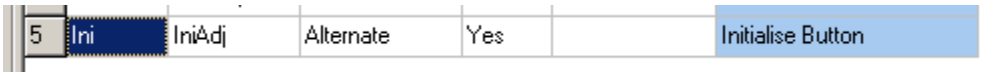


It is quite difficult to not say impossible to see the “Initialise” state because the time of this state is very short.

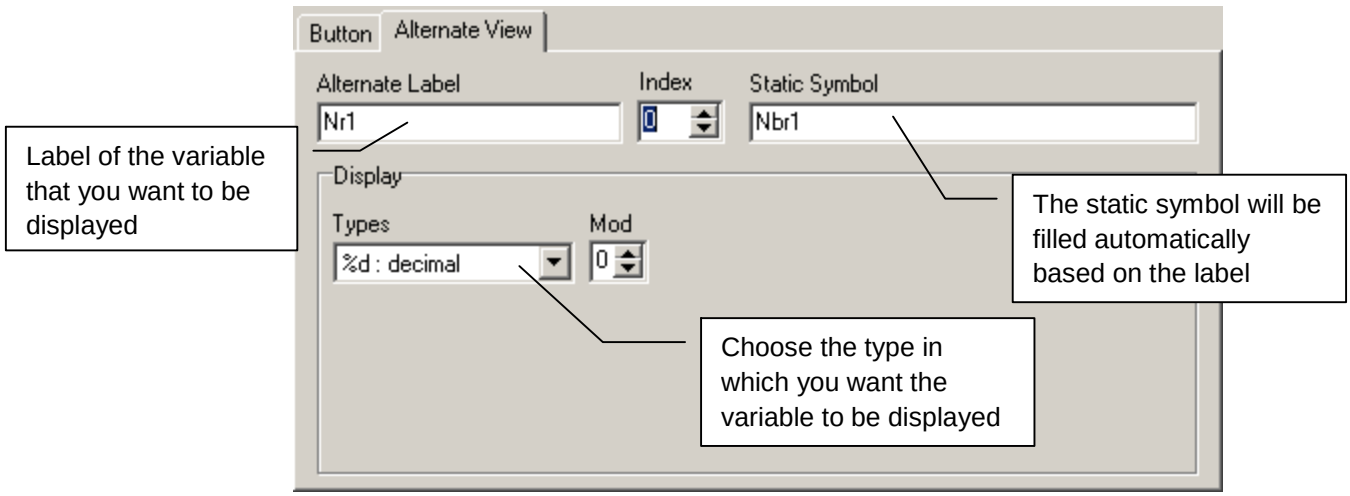
5.1.6 Alternate View

As said before, this type of Adjust And View variable is exclusively linked to a Button with View variable. The Alternate view simply means that you do not see the value of the static variable attached to the button but another variable of your choice. To carry on with our “Math” example, we will change the pre-defined Button with View with an Alternate View in which only on example purpose, we will display the first entry number of the mathematical operation.

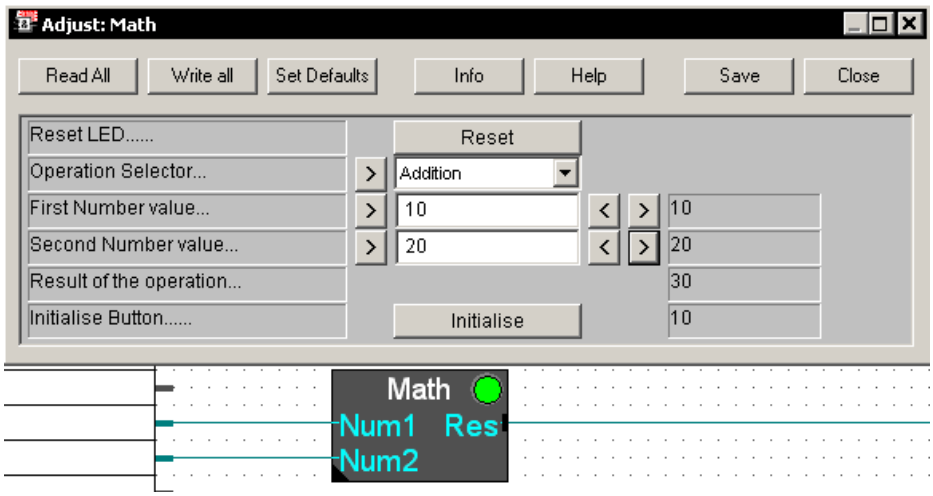
1. Modify the Initialise Button like follow:



2. Edit the properties of this Adjust variable the following way:



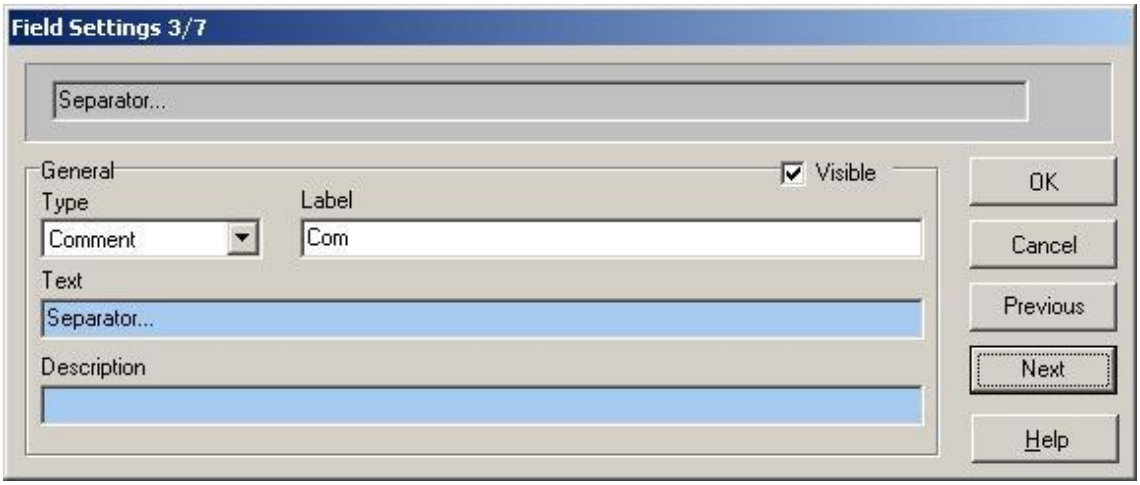
3. You can now check the result of your work in going in FUPLA, place your FBOX, download it and go online with your PCD. In FUPLA you should get something as follow:



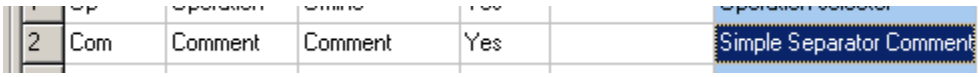
5.1.7 Comment

The Comment variable is straightforward, it is used either to describe another variable or just as separator. To have a better layout of our example dialog window, we want to add a comment just after the “Operation Selector”, you have to proceed like following:

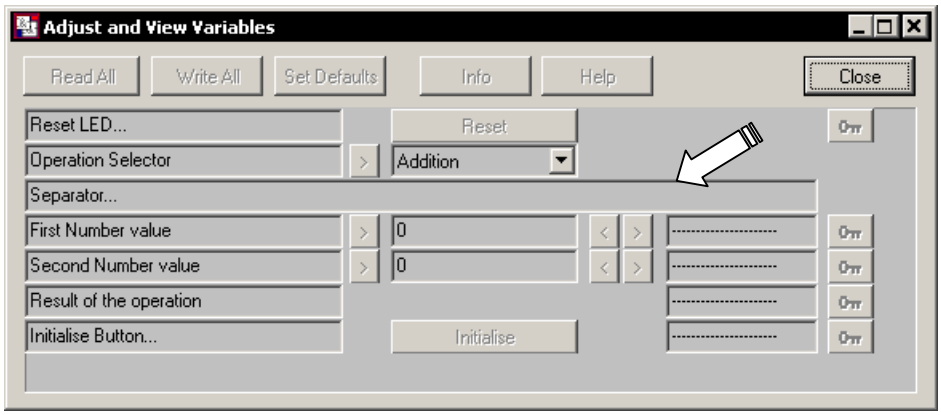
1. Insert a Comment just before the “First Number value” of the “Adjust” tab and edit the properties dialog as follows:



3 The adjust line should look like this :

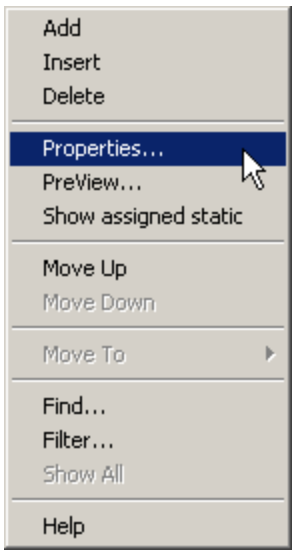


3 And you should get this dialog window:

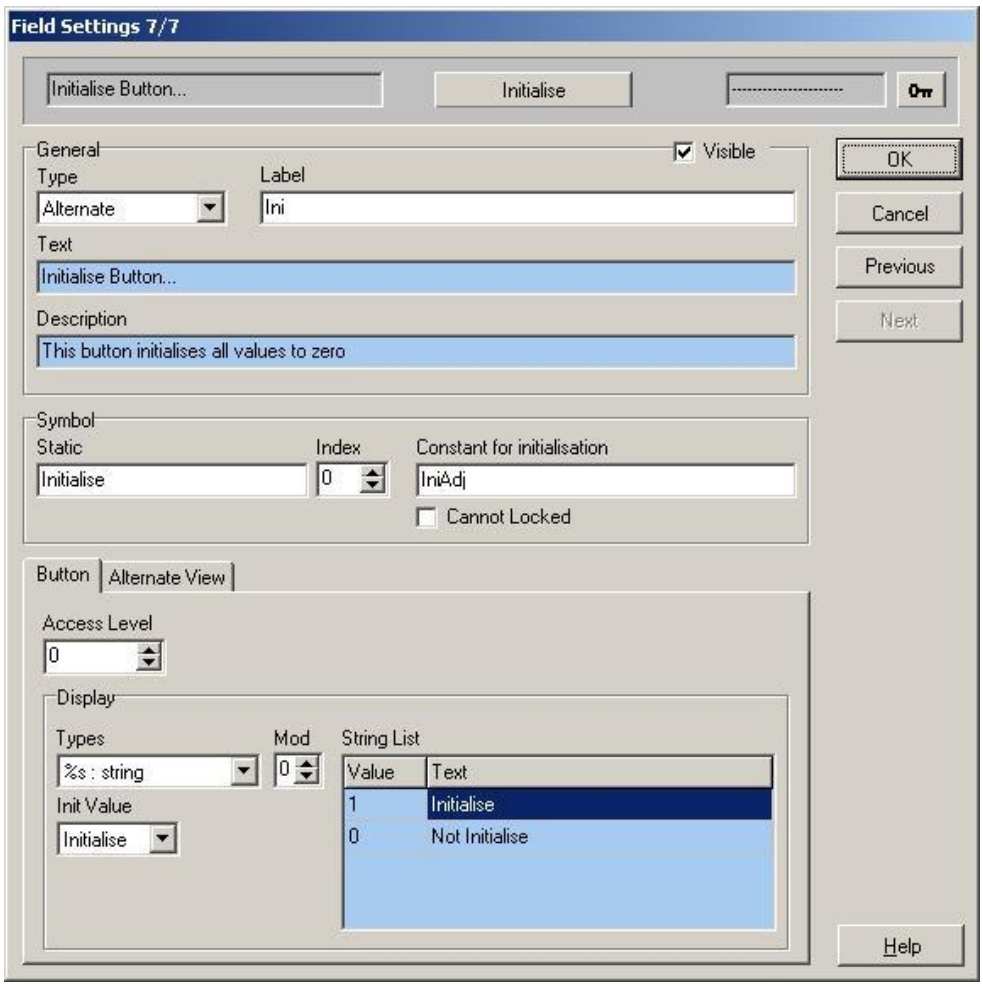


5.1.8 Adjust Variables Properties in Details

As example we want to have a look at the properties of an “Alternate View” because it covers everything. Maybe first thing to say is how to open the properties dialog of any Adjust And View variables; always select the variable you want, right click on it and choose the “Properties” menu:



The following dialog appears if you have selected an “Alternate View” variable:



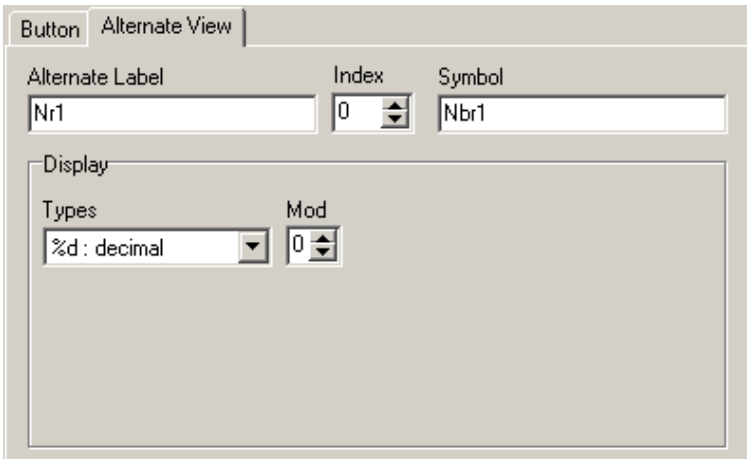
- The upper part is the preview.
- **Visible:** Visible or hidden on the dialog window.
- **Type:** Type of variables as described in the previous paragraph.
- **Label:** Adjust variables need a static variable. The link between this static and the Adjust variable is the “Label”, both must have the same Label.
- **Text:** Comment of the variable, visible on the left side of the FBOX dialog window
- **Description:** Help text that is displayed when you click on the comment text.
- **Static:** The static symbol defined in static tab 

- **Index:** The number beside the Static is the index in case the static variable was defined as an array of variable.
- **Constant for initialisation:** A symbol is needed to initialise the adjust variable.
- **Cannot Locked:** If this check box is not checked, this button  is displayed in the FBOX dialog window and gives the possibility to lock the address of an FBOX static variable. Only online adjust variable can be locked on an absolute address (offline variables are not associated to a PCD media they are constants given to the macro at compiled time.). This checkbox specify if an online can or cannot be locked on an absolute address.
- **Access Level:** Limit write access to an online Adjust variable while the PCD program is in running.
- **Types:** The format with which the value should be displayed.
- **Init Value:** The initial value.
- **Default Value** (not displayed on the above picture)**:** If no value is defined, the default one is taken.
- **Range** (not displayed on the above picture)**:** Range of accepted value as for example:

10..20	range: 10 to 20.
10, 15..20	range: 10 or 15 to 20.
10..12,15..20,0..5	range: 10 to 12 or 15 to 20 or 0 to 5.

- **String List:** List of values with their associated strings.

And in the alternate tab:

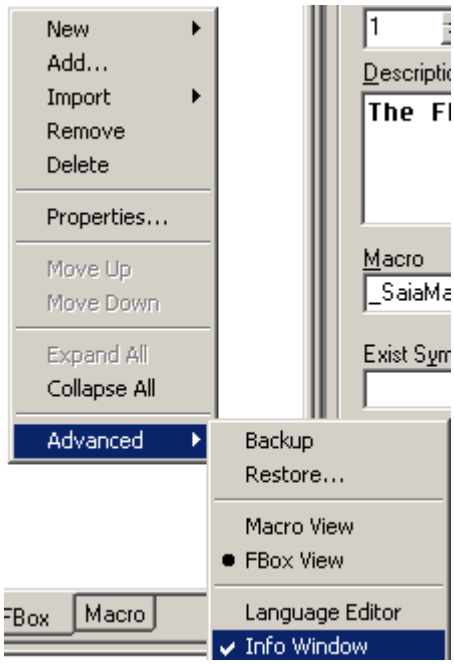


- **Alternate Label:** label of the static variable to be displayed in the Alternate View.
- **Index:** The number beside the Static is the index in case the static variable was defined as an array of variable.
- **Symbol:** It is filled automatically based on label
- **Types:** The format with which the value should be displayed.

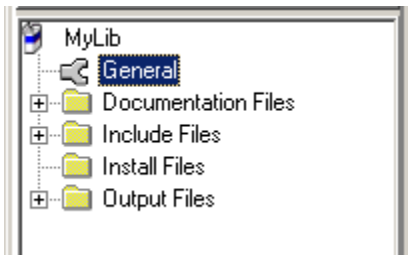
5.2 Library Information

The FBOX Builder ease the distribution of FBOX Library, the “Library Installer” gives you the chance to make an install package for the user of your libraries.

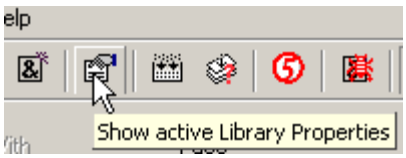
The Installer program needs some information to customize the installation process. First open Library Info Window using context menu of Library manager window:



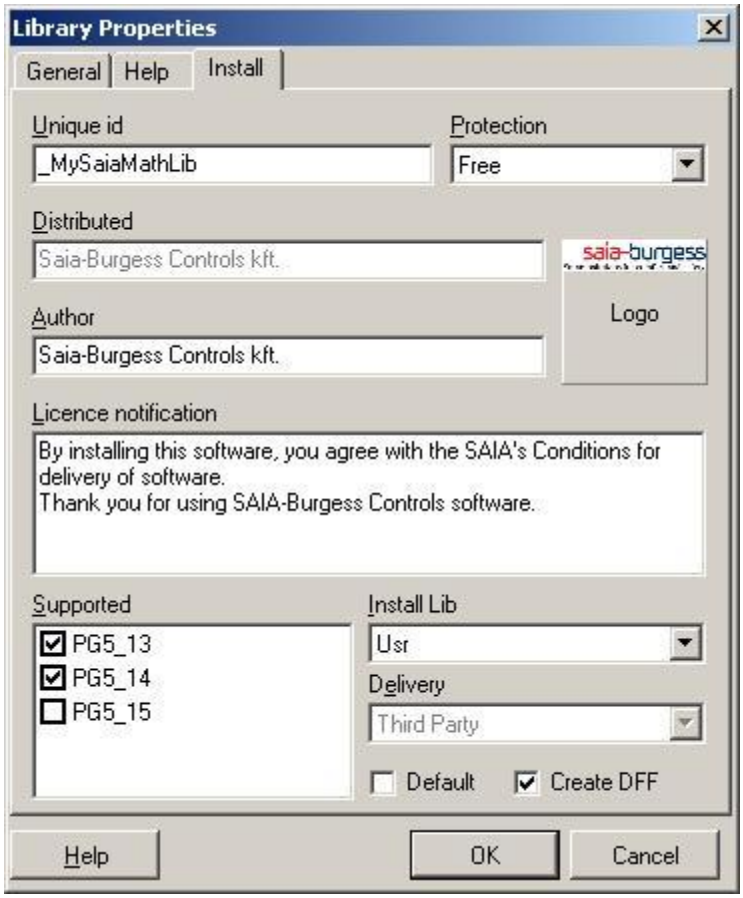
Now double click on the “General” node of the “Info View” :



Or use the appropriate toolbar icon:

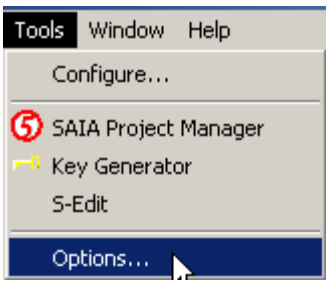


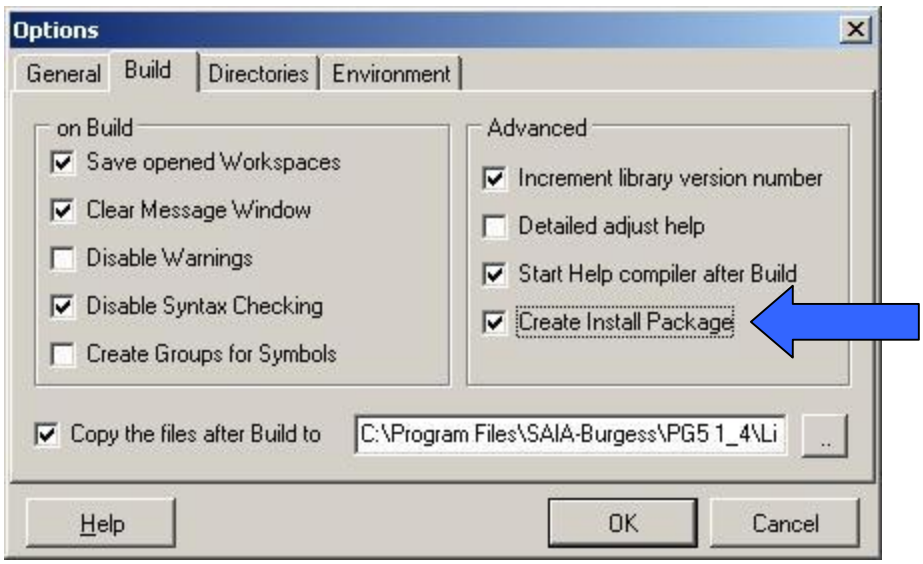
You get the following dialog; select the “Install” tab: 



- **Unique id:** Enter a unique identification string, this string is checked in all library files when installing a new library to be sure that you are not going to overwrite an existing library.
- **Protection:** Select the kind of protection you want to distribute your library. You can distribute “Encrypted” source file, Licensed library (working with the PG5 license mechanism) or you can distribute your library free.
- **Distributed:** This field is read only. It is filled based on the registered user of FBOX Builder.
- **Author:** Enter the name of the Library developer, this text will appear during the installation process.
- **Logo:** Click on the “Logo” button to browse to a picture that you want to be displayed during the installation process. (BMP, ICO, EMF and WMF types are accepted)
- **Licence notification:** Enter a text that you want the user to read during the installation process.
- **Supported:** Select the PG5 (PG4) versions that supports your FBOX Library. If the Installer does not find any PG5 of the selected version, an error message dialog will be popped up; then the user has the choice to cancel the installation or to browse for a directory to install the library anyway.
- **Install Lib:** Select in which PG5 directory your library should be installed. (STD, APP or USR USR is the default)
- **Delivery:** Only for SAIA internal use.

Now that you enter all the needed data, it is the time to create this Install package. Basically you just have to choose the “Create Install Package” option as follow:

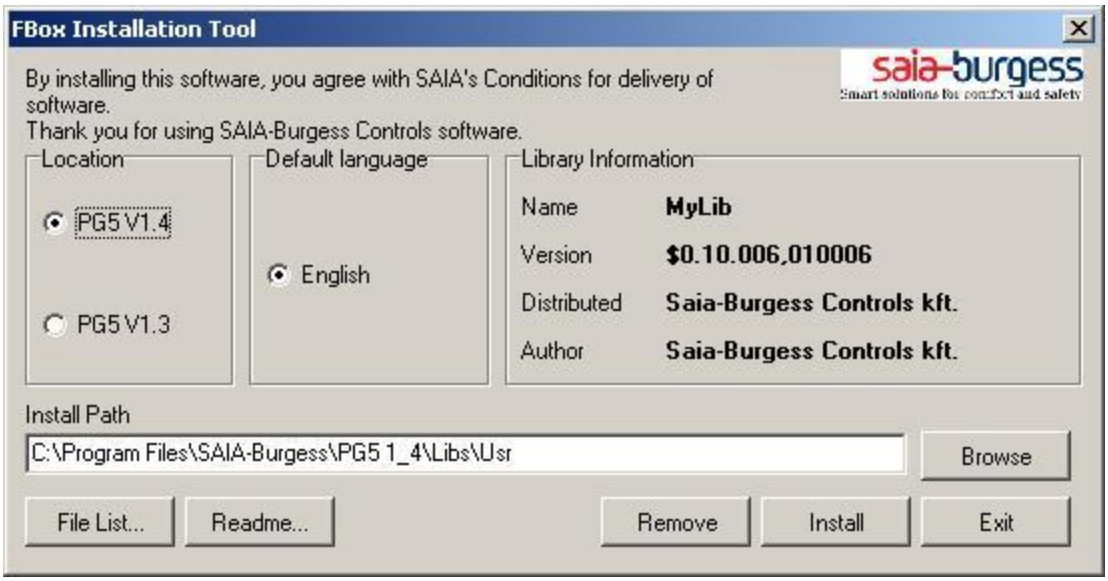




This way when you build you FBOX Library the Install Package will be automatically created. The package is an executable file, its name is based on the Library Name, the Language and the version. The Following message will be written in the “Message Window”:

```
Create package installer...
Install package "D:\SFEWork\MyProjectName\MyLibraryName\_libfiles\MyLibraryName_English_V0_0_0.exe" created.
```

If you double click on it, the executable file will be started and you can see the link with the “Library Properties Install” and the Install package:



The file list is simply the files that will be installed and the readme is based on the Library description you wrote in the Library Properties Dialog.

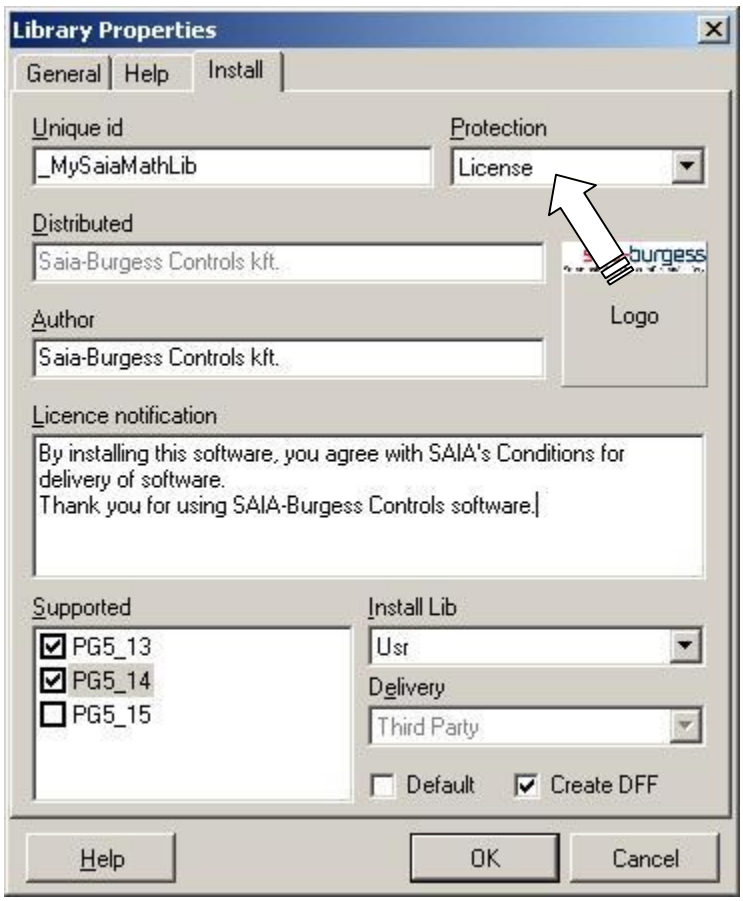
5.2.1 Licensed FBOX Library Install Package

This feature allows you to generate a licensed install package for your own libraries. The FBOX Builder generates a license key based on your customer name and on the Unique ID of your library. The install package will then ask this key before installing the library.

If a licensed FBOX is used without a license, PG5 wont compile them. This mechanism is implemented since the PG5 1.3 version.

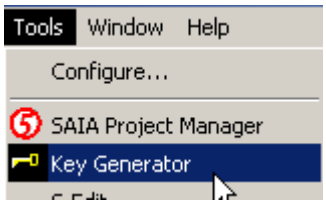
1. Set the Protection to **License** in the Library Properties dialog:

   Note: The distributed field is read-only, because it is taken from the PG5 user key and this info is part of the license code generation

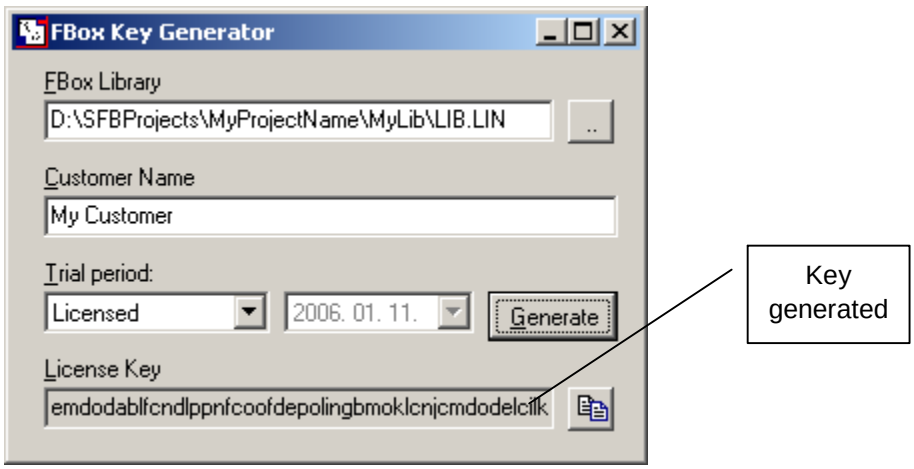


Build the library and generates an Install package.

2. You must now generate the key for your customer, to do so click on the “Key Generator” menu of the main tool menu.



The following dialog comes up:



In the FBOX Library field should be filled by default if Key Generator is opened from FBOX Builder and it should display the full path of the LIN file of your active library. The path of this LIN file is the following:

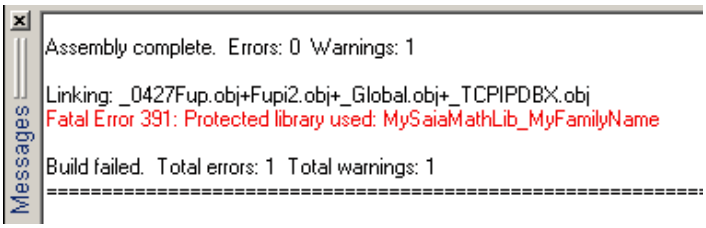
$FBOX Builder Project Dir$\$Your Project Dir$\$Your Library Dir$_libfiles

The customer name must be the same as the customer's PG5 key is.

The customer can find this name in the PG5 about box:

To install your Library, your customer will have to enter the generated key during the installation process. The installer will also enter a key in the computer registry of the customer, which will be used by PG5 to check the right to use this library.

Note: If at this dialog the user clicks on ***Cancel*** button, the application will ask to copy the files to the appropriate directory (without registering those FBOXes). If the user uses this way, the FBOXes will be available, projects that using this FBOXes can be opened, the user will be able to put those FBOXes into Fupla page.
The only restriction that the project cannot be built:

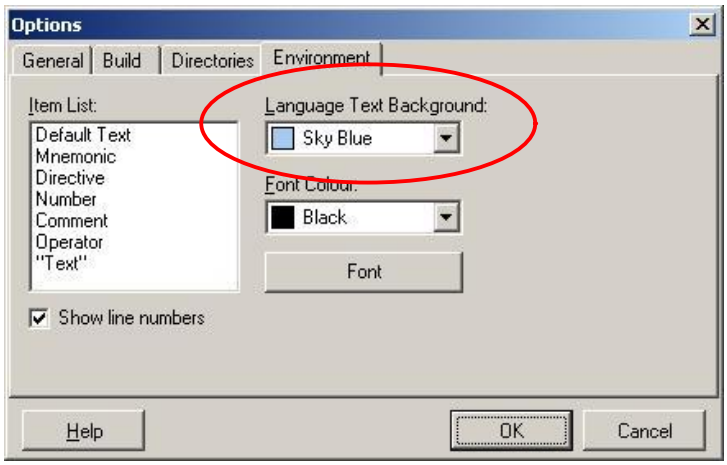


5.3 Multi-Language Handling

An FBOX Library that you distribute must - in some cases - support more than one language. The FBOX Builder has a “Language Editor” feature that helps you in the handling in the multi-language library development. You can develop your FBOX Library in one language and at the end when everything is finished, you can add one or more new languages and begin the translation

The help file and the FBOX strings themselves are language-dependent. The language-dependent texts are in edit boxes with light blue background (by default).

Note: To set the color of the language-dependent edit box background use the options dialog - Environment tab:

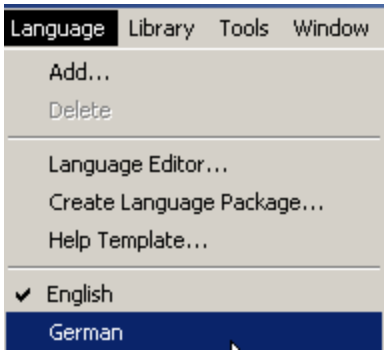


The default language is always English but you can easily add a new one as follows:

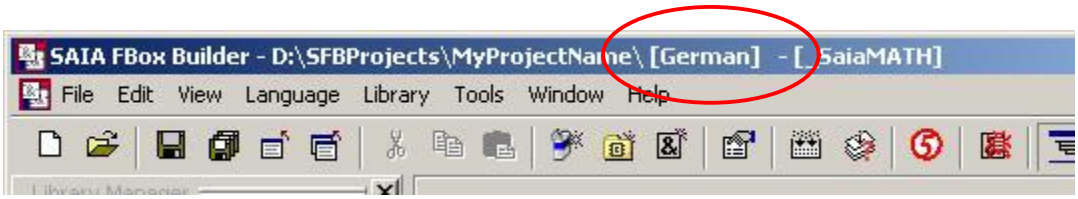
Select a language of your choice.

Now a new language is added. All the language-dependent texts will be filled with the texts in English.

To change from one language to another, you have to use the “Language” menu as follows:



The selected language is shown in the title bar too:

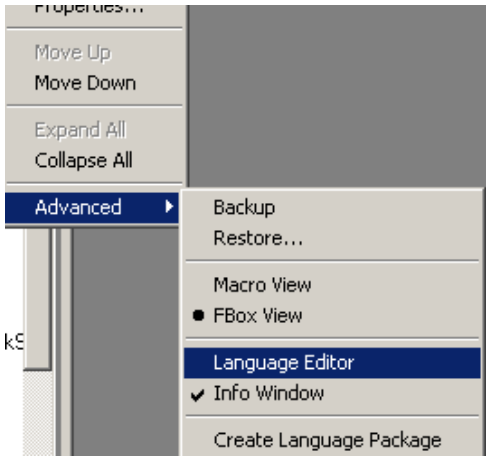


Now what you want to do is to translate all the strings to one language to the other, you have two solutions, either you select the language in which you want the translation and then go through each string of each workspace you need to modify or the quickest way is to use the “Language Editor”.

Important: All the languages must be added when creating a project and should never be deleted. The Restore mechanism is not able to restore more languages than the “Actual” project languages number.

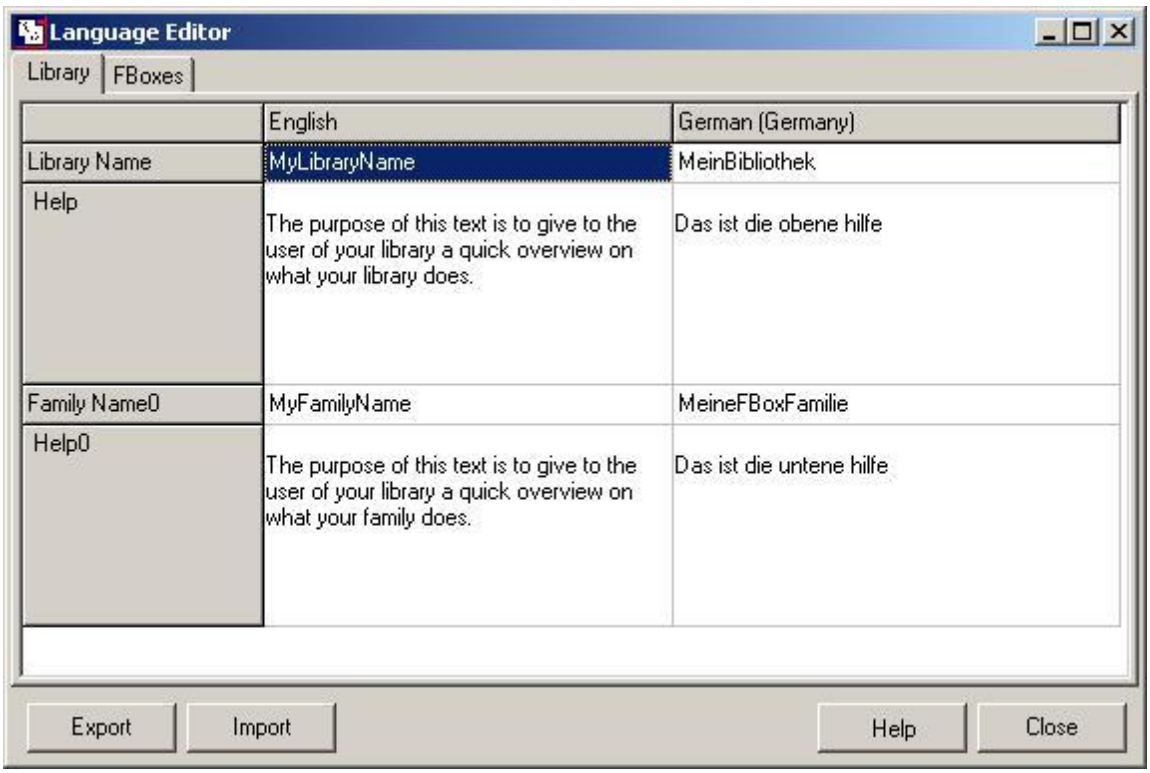
5.3.1 Language Editor

The “Language Editor” gives you a good overview of all the translation work you have to do.
To call the language editor use the context menu of Library Manager:

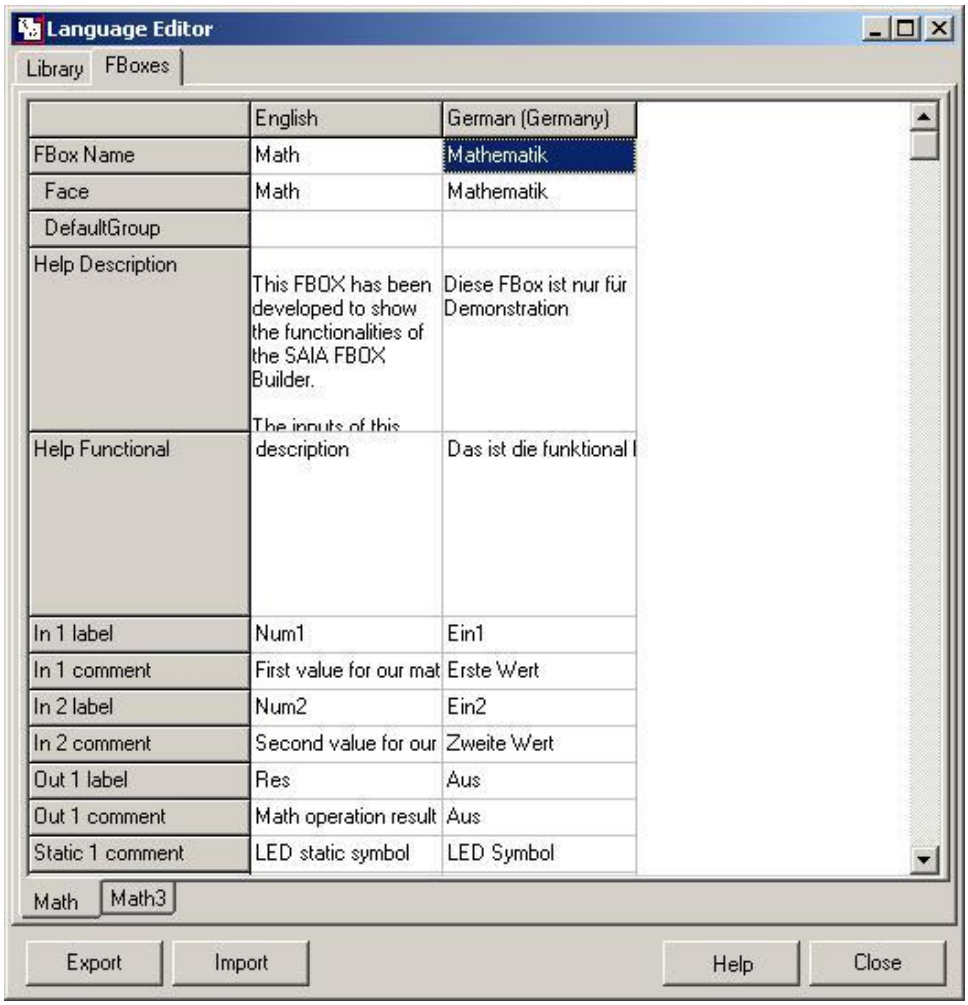


In the “Language Editor” you can edit the language-dependent texts in the Library level and in the FBox level.

Language-dependent strings in Library level:



And this is the FBoxes level.
At the bottom you can select the FBoxes in the Library



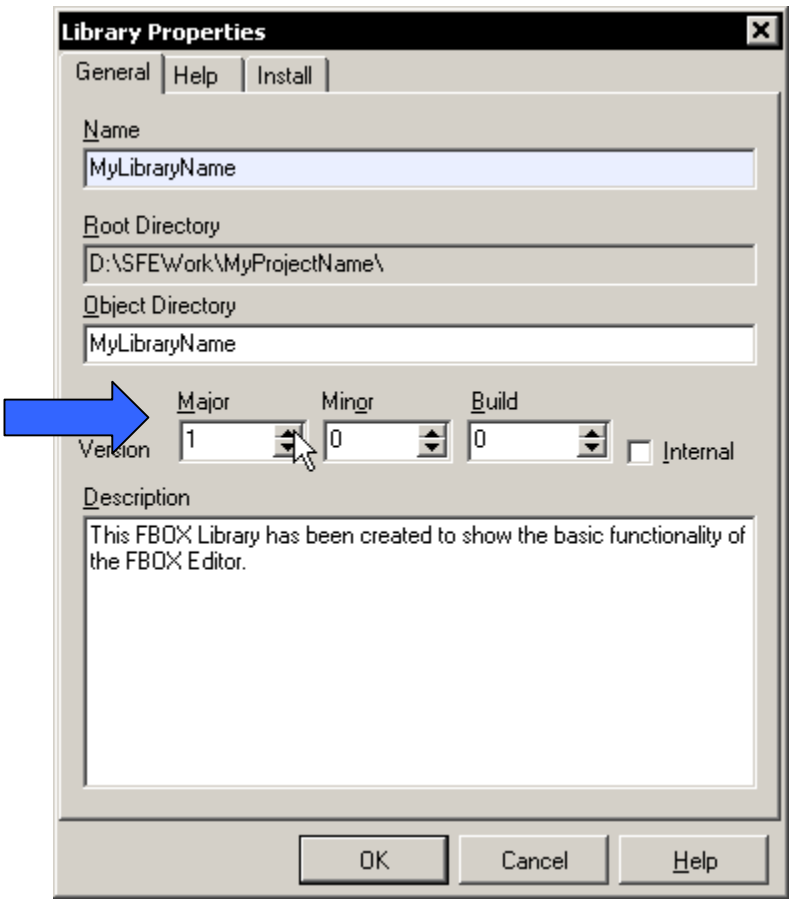
It is possible to export the texts (using Export button) into CSV format, and translate it in an editor of your choice, which handles this format (e.g. Ms Excel).
During export you have to select the reference language, and the language to translate.

After the translation file is complete, use Import button and select the translated CSV file.

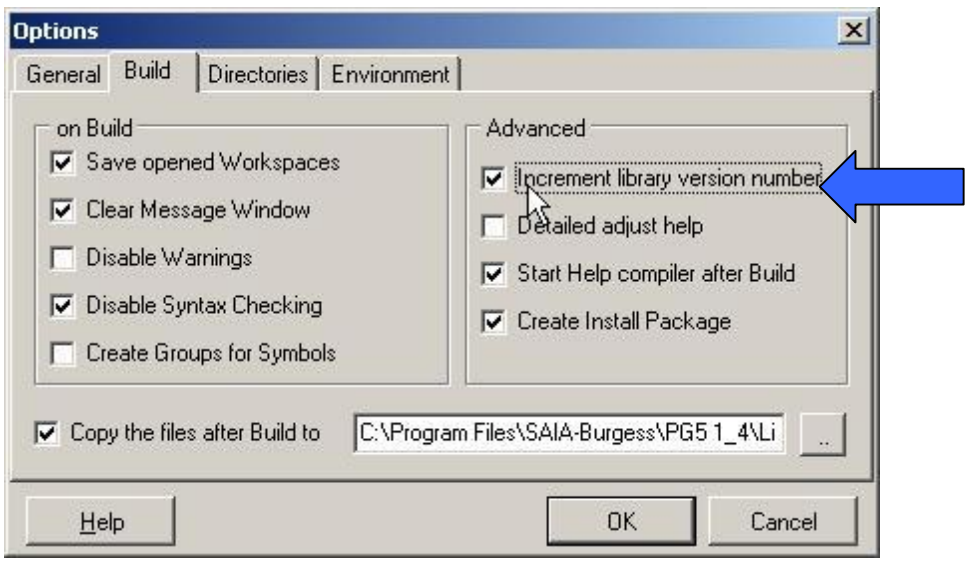
When the translation is finished, you can build your library. You will have to do a library build for each language.
But, it is possible to make a build with all possible languages using *Language* - .*Create all language package*

5.4 Backup/Restore Mechanism

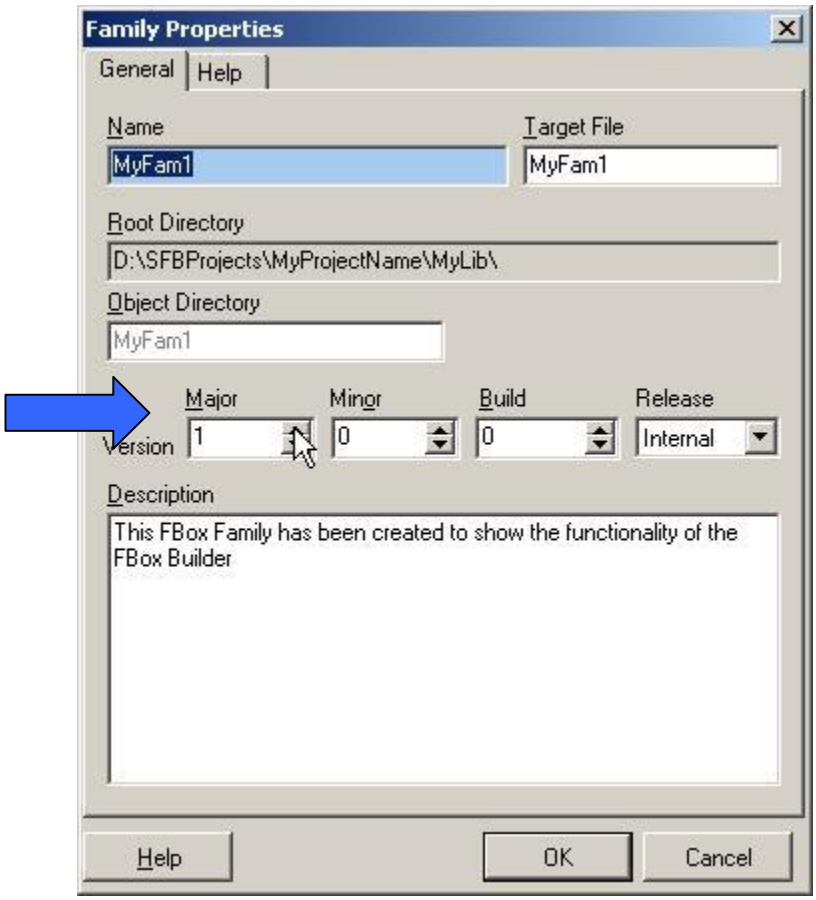
When you distribute your FBOX library, you have to manage different versions. The FBOX Builder has a Backup/Restore feature that helps you to handle the versions not only of the Library, but also Family and single FBOX. This mechanism works at every level and it is very simple to use.
The important when you do a backup, you must increment the version number. You find this version number for the Library in the “Library Property” window:



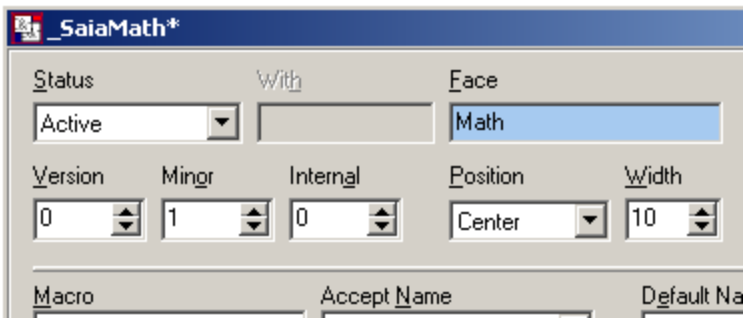
Tip: For the library level, you can make the version increment automatic if you choose the “Increment Library version after Build” option:



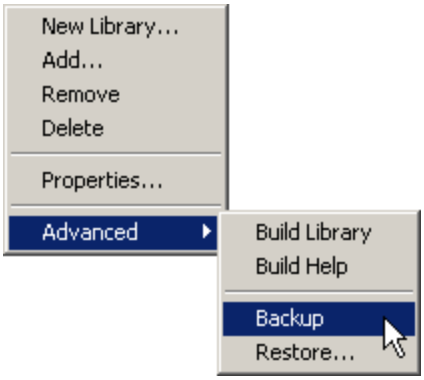
For the Family level, you find it in the “Family Properties” Dialog:



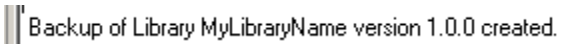
And finally for the FBOX Level, you find it on its workspace:



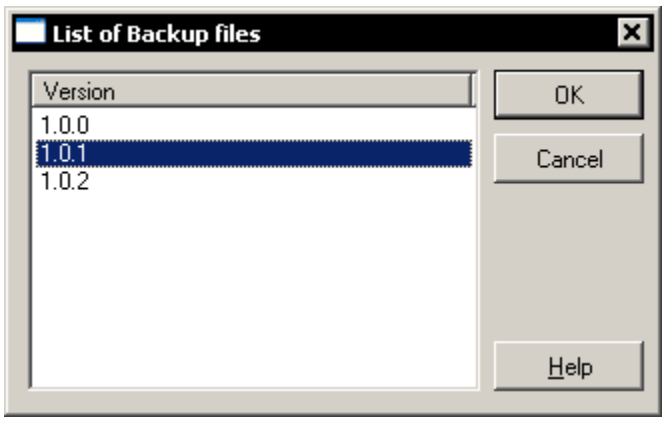
So to make a backup of any level, you have to right click on the object you want backup in the “Library Manager” and select the “Backup” menu:



You get the following message in the “Message Window”:



To restore any object that was backup, the process is exactly the same except that you have the choice of what version you want restore, the FBOX Builder ask you with the following dialog what version you want restore:



Warning: Do a backup of the current version before restoring an old one or it will be overwritten!

Important: All the languages must be added when creating a project and should never be deleted. The Restore mechanism is not able to restore more languages than the “Actual” project languages number.

5.5 Library Maintenance Handling

An FBOX Library that you distribute is always living, you might have to re-organise your library. The maintenance handling only works at the FBOX level. The FBOX Builder helps you to handle this kind of problems. If you have a look at the workspace of an FBOX, in the “General” tab you will see the following:

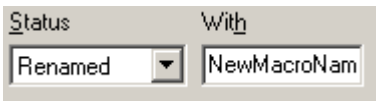
For each FBOX you can choose the following status:

5.5.1 Active

This is the default status of an FBOX.

5.5.2 Renamed

Renamed FBOX can be used if only the name of the FBOX is changed; the interface is identical. The new macro associated to the old FBOX can be located in a new family. FUPLA assumes that the new macro is identical to the old one except for its name.



5.5.3 Removed

The FBOX developer can decide to remove a FBOX from the library. This must be handled with care. FUPLA will cross out all FBOX, which were used and are not in the library. This way the user knows immediately that one FBOX is not part of the library anymore. Normally when the user will compile files with FBOX, which are not part of the library, an error will be shown “FBOX not in library”. The user can manually remove this compilation option. FUPLA remember from which family the missing FBOX come from, and if some info help is asked for these FBOX FUPLA will look in the appropriate family for this info.

5.5.4 Replaced

Replaced FBOX are used when an old implementation was changed (the interface of the new macro became incompatible with the old one). The user does not have to modify its code but FUPLA does not allow the user to add the old FBOX type. The old FBOX still work but the FBOX developer would like to force on a long term to use the new one.

5.5.5 Must be replaced

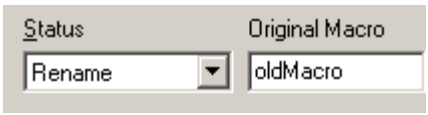
Must be replaced FBOX are used when the FBOX developer decides that no body should use the old implementation of an FBOX. The user must modify its code; replace the old FBOX with the new one. When displayed by FUPLA these FBOX are crossed-out. When compiled an error warns the user “FBOX not in library.” unless the option “error when FBOX not in library” is disabled.

5.5.6 Obsoleted

This tells FUPLA that this FBOX was removed from the library and that it was not replaced by any FBOX. The compiler gives no error, but the user cannot any more select this FBOX from the FBOX library.

5.5.7 Rename

This status is very similar to RENAMED the difference is you have to set here the original macro which will renamed to this macro.



5.5.8 Replace

This status is very similar to REPLACED the difference is you have to set here the original macro which will replaced with this macro.

5.5.9 Must Replace

This status is very similar to MUST BE REPLACED the difference is you have to set here the original macro, which must replace with this macro.

5.6 FB Import: Tags Definition

The FB parameters are used by the FB developer to bring data in the FB, to return a value at the end of the FB or both ways.
The FBOX Builder has no way to know what kind of use the FB will do with a parameter; it is the reason why we have defined some tags to ease the import of an FB in the FBOX Builder.
If those tags are not present, the parameters found during the parsing of the FB will be present in the “Parameter Editor”, but for sure not at the right place and not with the right name, in that case it is still possible to correct them by hand.

5.6.1 Input and Output Description

This kind of parameters will be respectively the “Input” and the “Output” of the FBOX based on this FB. The syntax is the following:

{In/Out= [<label>]_[E][N]<type> <Comment>}

<label> :	user defined label (max 9 chars)	
E :	edge triggered (only for inputs)	
N :	indexed (Stretch have to be enabled)	
<type> :	B	: binary
	I	: integer
	F	: float

5.6.2 Constant Description

The constant corresponds to the field, which is directly on the FBOX.

{C= [<label>]_[N]<type><media> <Comment>}

<label>:	max. 3 characters	
N:	indexed	
<type>:	B	: Binary (i,o,f)
	I	: Integer (r,t,c,k)
	F	: Float (r,k)
	D	: Data types (d,x)
	P	: Block types (s (=sb), p (=pb), u (=fb)
<media>:	[i] ¦ [o] ¦ [f] ¦ [r] ¦ [t] ¦ [c] ¦ [k] ¦ [d] ¦ [x] ¦ [s] ¦ [p] ¦ [u]	

It is possible to define several medias: C=Block_Irk (integer type, register and constant are accepted)

5.6.3 Static and Dynamic Description

They are not displayed on the FBOX face, they can be used as internal or init values (example adjust and view variables are using static symbols)

{S/D= [<label>]_[<count>]<media> <Comment>}

<label>:	max. 9 characters	
<count>:	1..255 (default 1)	
<media>:	F	: flag
	R	: register
	T	: timer
	C	: counter
	X	: text (only for static)

D : data block (only for static)
^X : Ext text (only for static)
^D : Ext data (only for static)

Adjust value specification:

If the static symbol has an assigned adjust variable, it can be predefined with the following syntax:

In5 DEF =5 ; {S=Tim_T} [a Tim 0 =1 {1..99999};Part 1 Year/Week (YYYWW)] comment

The adjust definition has to be in []. The ";" separates the definition and texts, the texts can also contain the string values.

The string description:

"**<text-name>:**	Is a string which will be placed on the left most column of the adjust list window.
\R)<text-description>:	This is the text describing in more detail the variable. It can be accessed by double clicking the left most column of the adjust list window. It must be preceded with the characters **\R)**
\n)<text-value>:	This is the text which is displayed when a value **n** is given to this variable if it has used the string format. It must be preceded with the character **\n),** where **n** is the value which is associated with this text."

5.6.4 Example

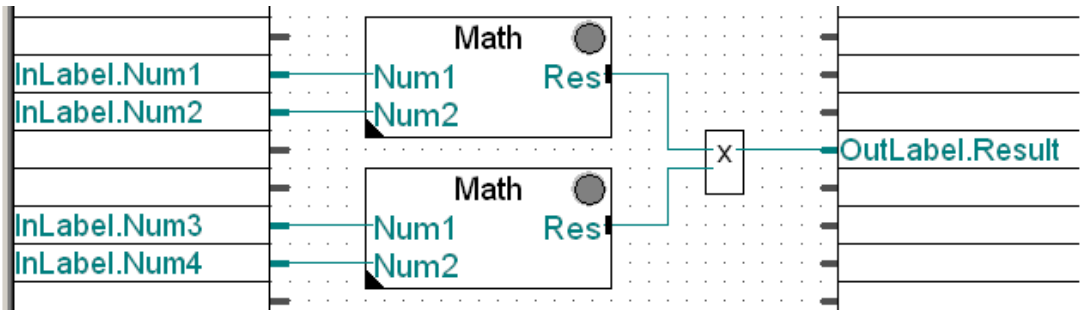
```
1  ;;Blinker FB
2  BlinkInit    EQU     FB
3  Blinker      EQU     FB
4  Timer        EQU     T
5
6  ;;Init FB
7                FB      BlinkInit
8  IV       DEF      =1      ;;{In=Tiv_I} Timer Init Value
9                GET =1
10               Timer
11               EFB
12
13
14 ;;Run FB
15               FB  Blinker
16 Ou       DEF      =1      ;;{Out=Out_B} Output
17 TV       DEF      =2      ;;{In=Tv_I} Time Value
18               STH Timer
19               JR  H EndFb
20               ACC H
21               LDL =2
22               Timer
23               COM =1
24 EndFb:
25               EFB
26
```

5.7 FUPLA Page(s) Import

The FBOX Builder gives you the chance to develop your own FBOX without writing a single line of instruction list code. The whole functionality of your FBOX can be based on one or several FUPLA page(s). The example described in this paragraph is based on the “Math” FBOX developed all along this manual.
You have to keep in mind that it is not meant to be a “full program compressor”; it is not reasonable to take an existing program of twenty pages and directly import it in the FBOX Builder. To work correctly **it should never exceed 3 pages and the maximum nested level should not exceeds 5**. By nested levels, I mean import a page, make another page with the resulting FBOX and re-import it and so on…

5.7.1 Program the functionality

Open your FUPLA and program the following page:



The Adjust dialog of each “Math” FBOX stays unchanged that means each “Math” FBOX does a simple addition between “Num1” and “Num2”. The results of both of them are finally multiplied.

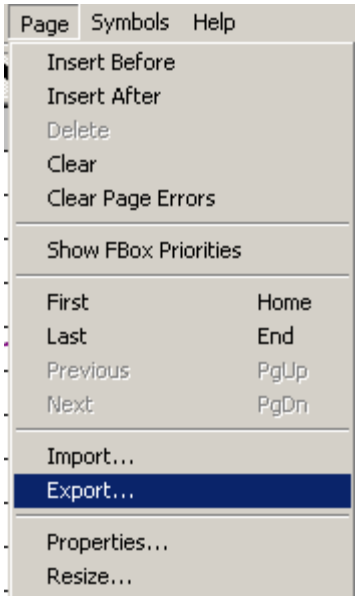
You can see on this page the groups “InLabel” and “OutLabel” which are used here as tags. The FBOX Builder will read these tags during the import process to place these parameters at the right place in the “Parameter Editor”, this works exactly the same way as the FB import mechanism.

Other tags are defined for the FUPLA page(s) import features:

- **“InLabel”** tag for FBOX inputs variables
- **“OutLabel”** tag for FBOX outputs variables
- **“StaLabel”** tag for FBOX static variables
- **“ConstLabel”** tag for FBOX constant variable
- **Not tagged means dynamic variables!**

5.7.2 Export the FUPLA Page

After the compilation and the tests of this page, we are sure that the functionality is correct we can export it. Select the “Export…” menu of FUPLA.



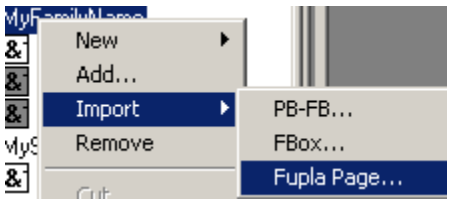
FUPLA asks you a name for the exported file and gives you the opportunity to export one or several pages. For our example, select only the current page. FUPLA now will create an “FXP” file that contains all the data needed to re-create the page. The FBOX Builder used as input data the following file:

- “FXP” file which is the result of the export page

In the “import” process FBOX Builder will start FUPLA to compile the FXP file. The result of this compilation is mainly an “FBD” file that the FBOX Builder needs to define the behaviour of the imported page.

5.7.3 Import the Page(s) in the FBOX Builder

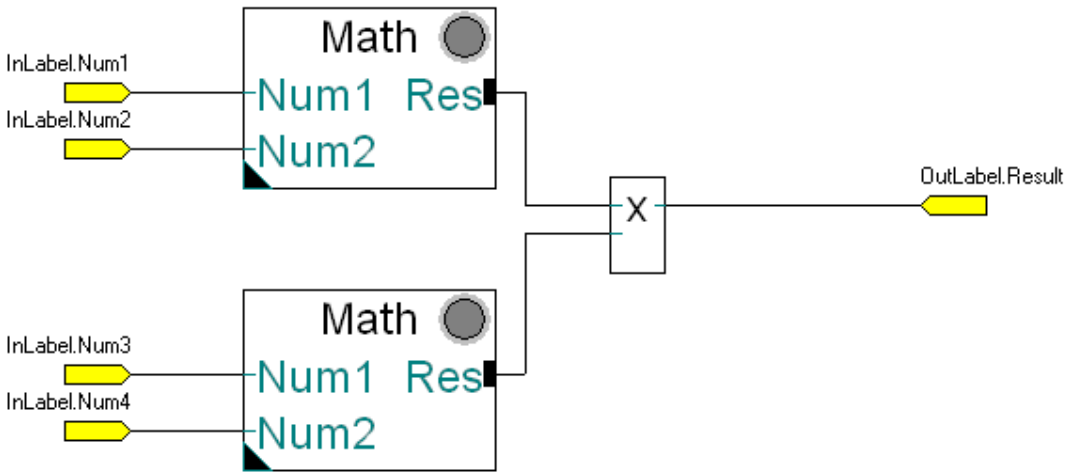
To create an FBOX based on exported FUPLA page(s) is quite straightforward. The only thing you have to do is to right click on the family where you want create this new FBOX and choose the menu “Import -> ZIP” as follow:



Then you just have to browse for your just created “FXP” file; give a name for your new FBOX for example “MathExtend”; the FBOX Builder will create a workspace for the FBOX.

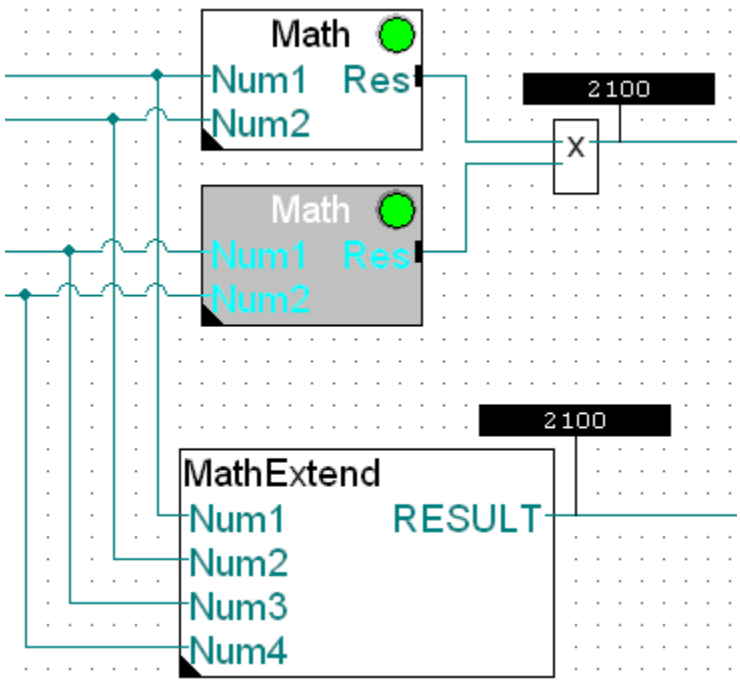
If you have a look at the “Parameter Editor”, you will see the inputs and output we have tagged are at the right place and you will see a whole bench of variables of three letters in the static and adjust tab. It is your work now to move the variables from one tab to another as needed.

You can see the schema of the imported page(s) under the Schematic tab of the workspace; our example looks like this:



5.7.4 Check the Resulting FBOX

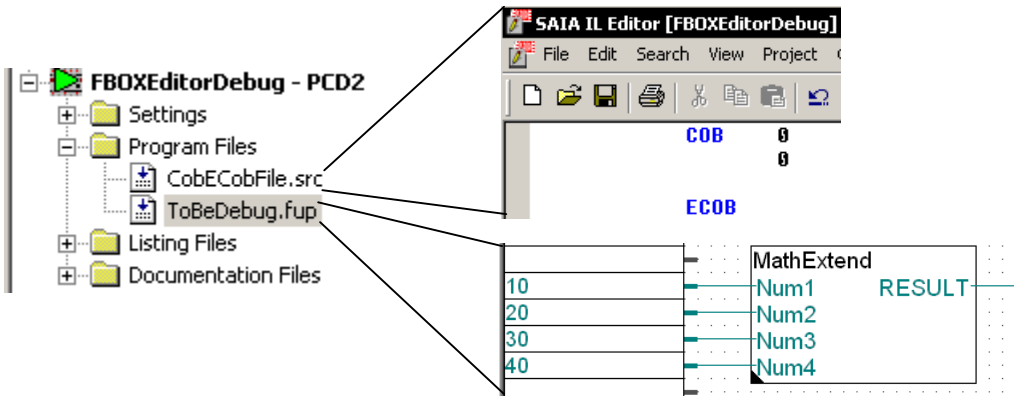
To be sure the import functionality has worked correctly, compile your library and place the IMPORTED FBOX on the same page you exported, with the same entry values you must get the same results as shown in the picture below.



5.8 FBOX Builder Debug Functionality

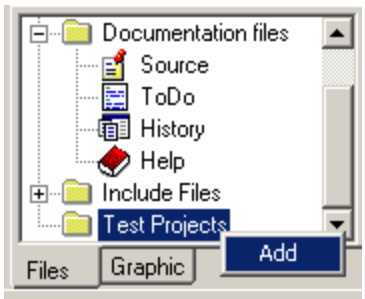
It is not easy to debug an FBOX; the code contained in a macro is not traceable like the code of an FB or of a COB. The FBOX Builder debug feature helps you to trace the code of your FBOX in putting it in a COB basically.

To use this functionality, you must create a PG5 project with an IL source file in which you just write the statement COB/ECOB and a FUPLA file where you drop the FBOX you want debug.



Those two files must be linked, compiled and then return in the FBOX Builder.

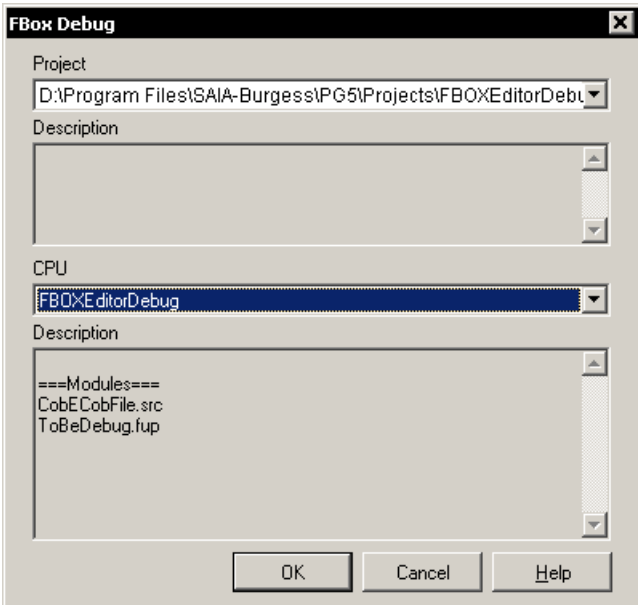
Open the workspace of the FBOX to be debugged, add to the “Test Project” node of the workspace the PG5 project you have just created as shown below:



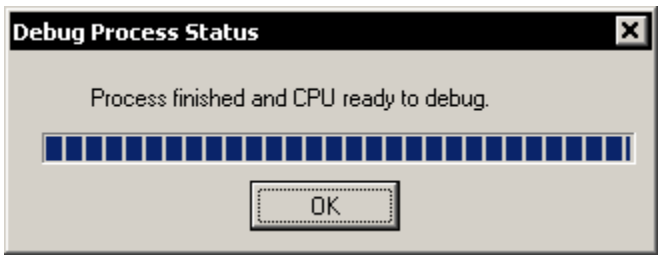
And then browse to your project.

We are ready now to start the modification process of the FUPLA generated file (FBD) to be able to debug it. Press the “Debug FBOX” button  of the main tool bar.

You get the following dialog:

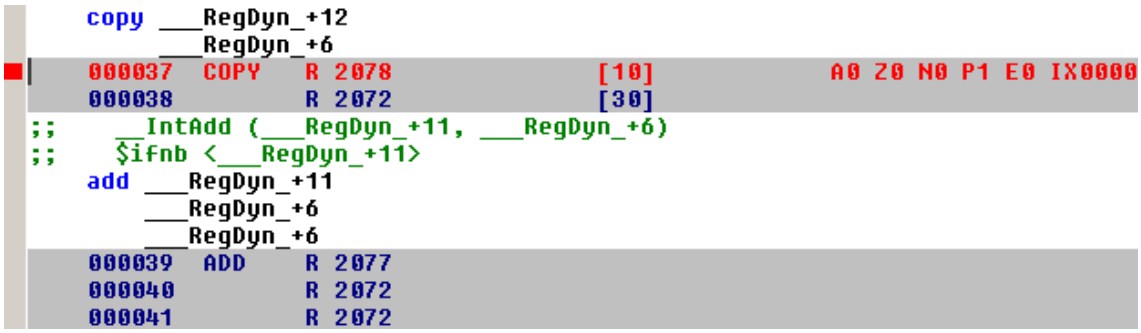


Click on the “OK” button, the FBOX Builder will start the PG5 to compile a first time the project, then the FBOX Builder will modify the “FBD” and start again the PG5 to re-compile the project with the modified files. At the end of that process, you get this dialog:



5.8.1 Debug the FBOX

The idea of this feature is basically to put the FBOX code in a COB. This was done in modifying the FUPLA generated file (FBD). You might ask yourself “What the source file with the COB/ECOB statement is made for?”; the answer is very simple this file gives you the chance to put a break points to put the PCD in trace mode. And then you can use the “Trace into” functionality of S-Edit. You should get something as follow when you are online and tracing into the FBOX:



As you can see, you have the “PCD” code and the source code like a normal IL program.

Contents

6 FBOX Language Introduction

6.1 Forewords

This chapter is made for people who want a detailed understanding of what the FBOX does behind the scene. The FBOX Builder automatically handles most of the points treated in this chapter.

6.2 FBOX Structure

An FBOX is based on a macro written in IL that defines the functionality and a graphical interface that allows an easy configuration. The “FBOX Language” which is our main concern in this chapter defines the graphical interface of an FBOX. The functionality and the graphical interface of an FBOX are tightly linked together therefore; a brief introduction must be done about the macros in IL language.

6.2.1 Introduction to Macro

A macro is a sub-program that can be called from the main program. Basically the reason why to use IL block like macro, FB or PB is to avoid writing several times the same functionality or to improve the readability of a program.

Macros, like FBs can have parameters; the syntax, the parameters types are different between these two blocks, but the main difference is that each macro call placed in the program will be replaced by the appropriate code during the compilation as for the FBs, each call will execute the same piece of code. In using macro, we spare the time to call the FB but the memory needed by the program gets bigger for each macro call.

The reason why we use macro with FBOXES instead of using FBs is because of the flexibility of the parameters type allowed by the macros.

The macro has the following syntax:

```
MacroName macro Param_1, Param_2, … Param_n
;;Macro code
….
Endm
```

The macro call has the following syntax:

```
;;IL Program
…
MacroName Param_1, Param_2, … Param_n
…
```

The macro that belongs to an FBOX is a normal IL macro but its parameters must match the definition of the graphical interface.

For more details about macros see the “SAIA Instructions List Help”.

6.3 FBOX Library Files Structure

The information needed by FUPLA and the Compiler is distributed in several files. Each of them has its own responsibility.

The file type can be divided in two different families, the one that belongs to the FBOX library:

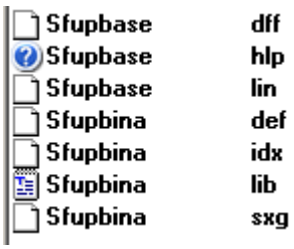
- Library information (.LIN file)
- Family list order (.DFF) file
- Library Help (.HLP)

And the one that belongs to the FBOX family:

- FBOX Definition (.DEF)
- FBOX Functionality (.LIB)
- FBOX Display Text (.IDX)
- FBOX Family Management (.SXG)

All the “Family” files have the same name with a different extension and the “Library” files have also the same name with a different extension.

Example:



6.3.1 Library Information (.LIN)

This file is not mandatory; it is used as information source for the Install Package tool. The FBOX Builder generates this file based on the library information found in the "Library Properties", tab called "Install".

The information contained in the file is separated in section as follow:

[ID]	Identification
ID=_SAIA_Modem_SMS	Unique identification for the library.

[About]	
Name=Modem SMS Library	Explicit name of the library.
Version=$2.1.306,21306	Version indication. String format and numeric format.
Distributed=SAIA-Burgess Controls	Responsible for the distribution. SAIA or third party.
Author=Engiby, N.Bovigny	Author of the library.

[Licence]	Section for license note displayed before the package is installed.
By installing the software...	License not. Up to 4 text lines can be displayed by the Engiby-Installer.

[Library]	Kind of library.
Type=Fbox Type=FB Type=FB and Fbox	For FBOX library For FB library If both are supported.
Delivery=Base Delivery=Optional Delivery=Third Party	Delivered as Base library with PG5 Delivered as optional library Delivered by Third party
Hierarchy=Single Hierarchy=Main Hierarchy=Sub	Single library without sub-libraries. Main library having sub-libraries. Su-library, which needs installation of a main library.

[Main]	Not yet implemented Define the required Main library
ID=_SAIA_Modem	Unique ID of the required Main lib.
Name=Modem Base Library	Name of the required Main lib.
MinVers=21306	Minimum version of the required Main lib.

[PG]	PGx supporting the lib.
PG4=No PG4=Yes	Not supported by PG4 Supported by PG4
PG5_xx=No PG5_xx=Yes	Not supported by PG5_xx Supported by PG5_xx xx is **10** for vers 1.0.xxx, **11** for vers 1.1.xxx and so on
PG5Libs=Std PG5Libs=App PG5Libs=Usr	To be installed in Std folder To be installed in App folder To be installed in Usr folder

[DOC]	Section for documentation files to install.

`Readme=Generic.txt,Link_name`	Name of a Readme file. This files can be viewed before to install the package. If a link name is given (see File0.. below), a link will also be created like for otheer files.
`Group=Engiby Doc`	Default group name for the link installed in the Start - Program menu of Windows.
`File0=Generic.doc, GenericDoc` `File1=...`	List of files to install. A link is created with the link name for file where a link name is specified.

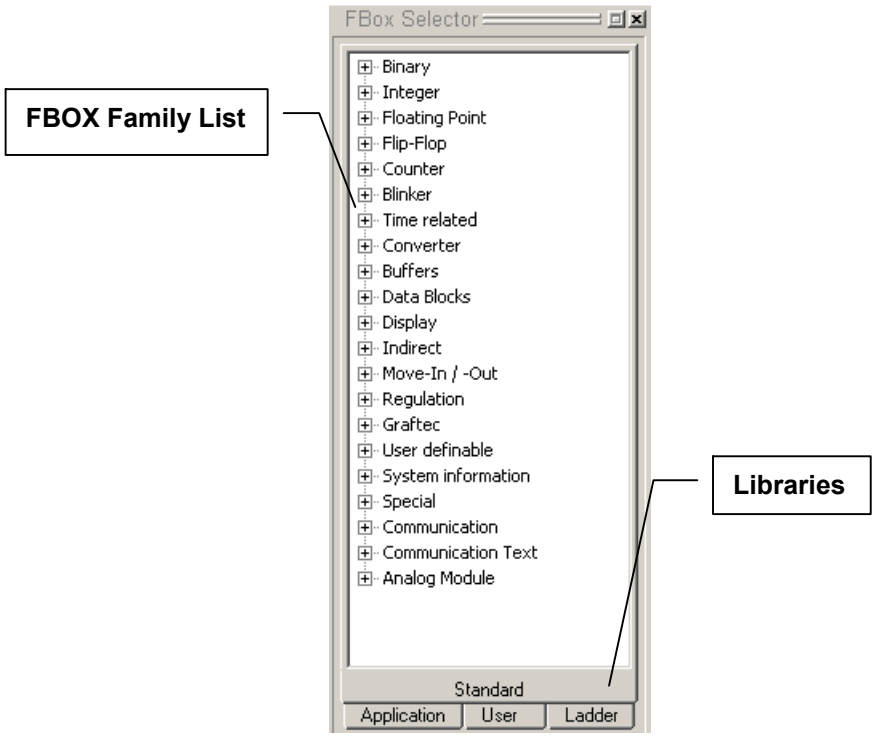
`[FileCheck]`	Check the files to be installed or check the presence of the files in the folder.

`File_mask=min_number,max_number,min_size,max_size,min_date,max_date`
`Example:`
`file*.*=4,5,10000,20000,01.01.2001,31.12.2001`
`file3.obl=1, ,328, ,14.09.2001`
	Check that :
	1) The number of files corresponding to the mask is in the given range.
	2) That the total size in bytes is in the given range.
	3) That the date of each files is in the given range.
	If a parameter is missing, the check for this parameter is skipped. If a file name is given without parameter, the file must simply exist.

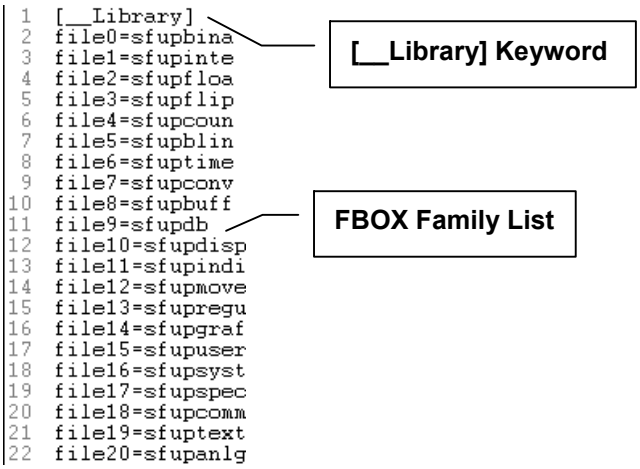
`[DeleteFiles]`	Section Files to be removed before installation.
`File0=SMRProt.mac` `File1=...`	List of file to remove.

6.3.2 Family List Order (.DFF)

This file is not mandatory; it is used to define the order of the family list of the “FUPLA FBOX Selector”.

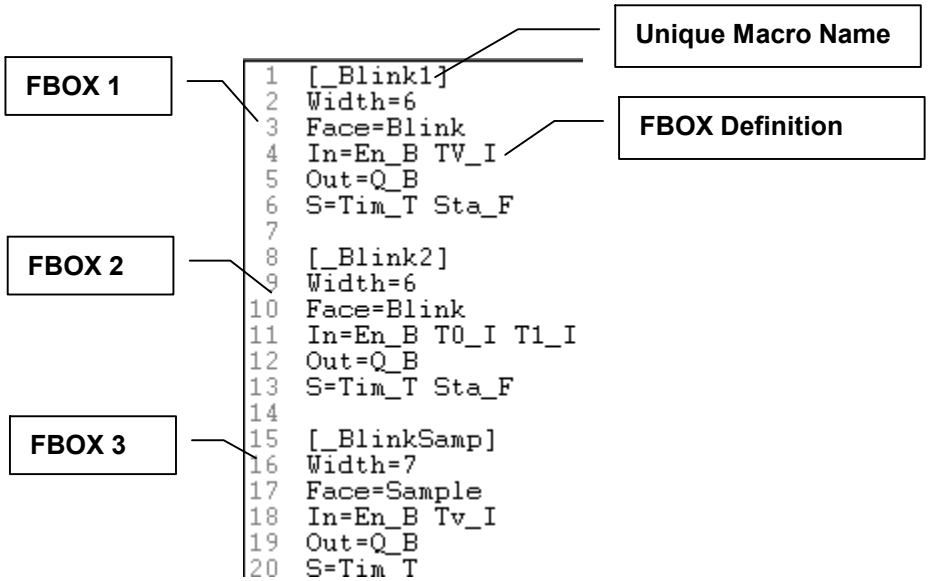


The Family List Order file simply contains a list of the families in the right order. This file has the following structure:



6.3.3 FBOX Definition (.DEF)

This file is mandatory and it's the key information for the graphical face of the FBOX in FUPLA. We can say that its structure is "FBOX Oriented"; in other words this file is separated in sections where each sections represent an FBOX. The ".DEF" file looks like this:



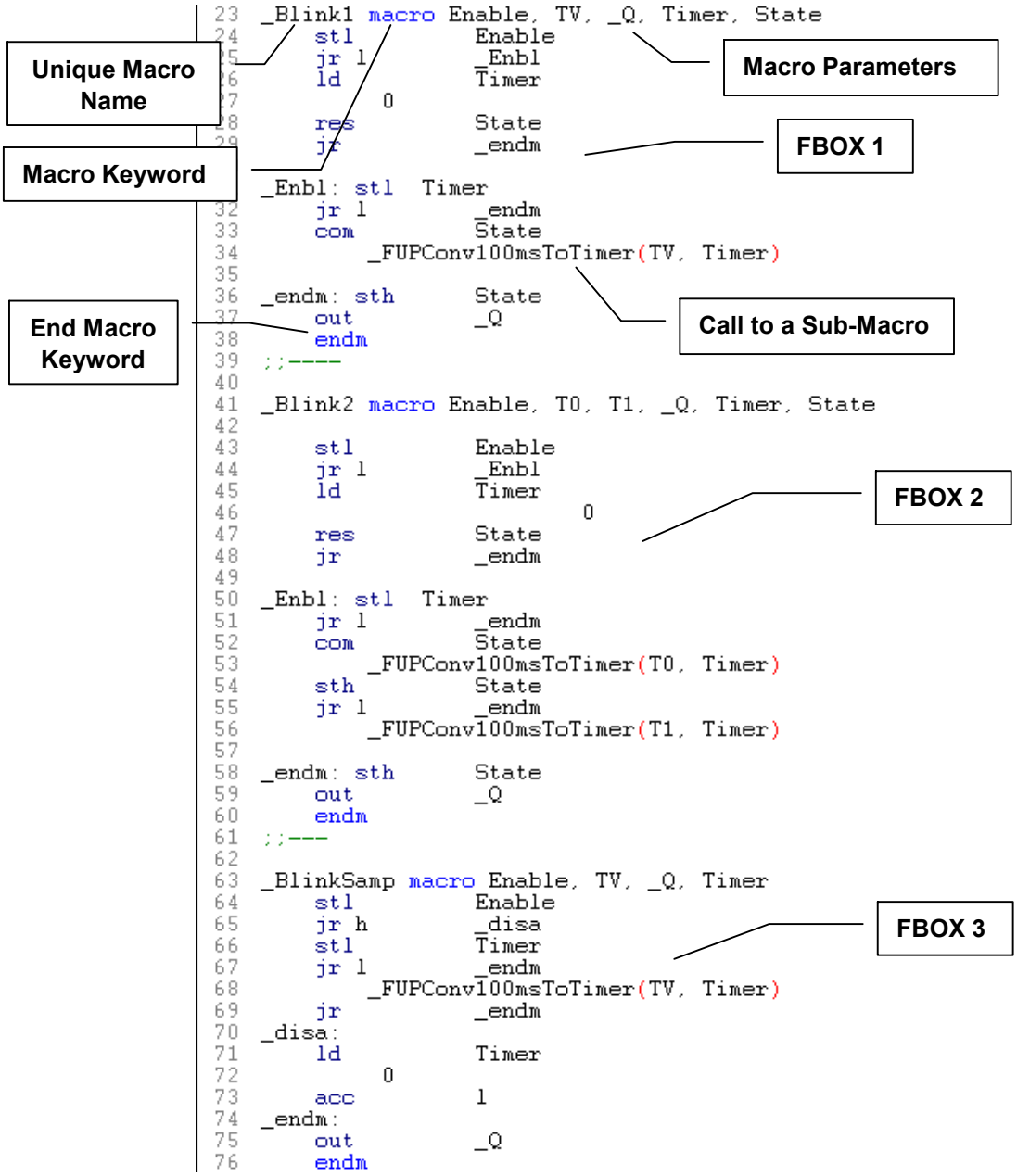
The macro name is the "Keyword" to make the link between all the FBOX of the Family. For this reason the macro name must be unique not only in the family but also among the Libraries. If not, when opening FUPLA an error message will be displayed.

The definition part of an FBOX section is the central piece of the "FBOX Language" which will be treated in details in this chapter.

This file does not need to be edited by the FBOX developer, the FBOX Builder generate it.

6.3.4 FBOX Functionality (.LIB)

This file is mandatory and defines the behaviour of an FBOX. The structure of that file is also “FBOX Oriented”. This file looks like this:

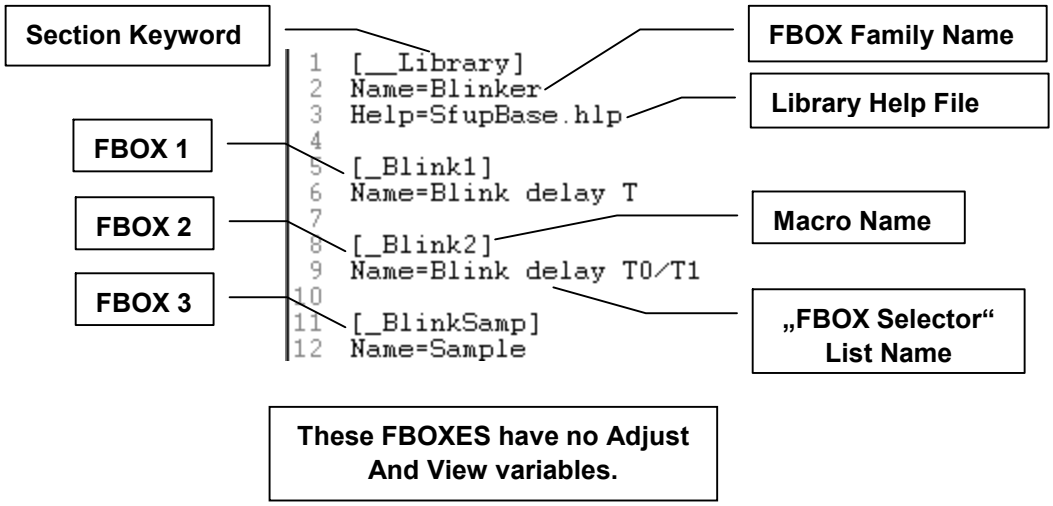



```
23  _Blink1 macro Enable, TV, _Q, Timer, State
24      stl         Enable
25      jr l        _Enbl
26      ld          Timer
27              0
28      res         State
29      jr          _endm
30
31  _Enbl: stl  Timer
32      jr l        _endm
33      com         State
34          _FUPConv100msToTimer(TV, Timer)
35
36  _endm: sth      State
37      out         _Q
38      endm
39  ;;----
40
41  _Blink2 macro Enable, T0, T1, _Q, Timer, State
42
43      stl         Enable
44      jr l        _Enbl
45      ld          Timer
46                      0
47      res         State
48      jr          _endm
49
50  _Enbl: stl  Timer
51      jr l        _endm
52      com         State
53          _FUPConv100msToTimer(T0, Timer)
54      sth         State
55      jr l        _endm
56          _FUPConv100msToTimer(T1, Timer)
57
58  _endm: sth      State
59      out         _Q
60      endm
61  ;;---
62
63  _BlinkSamp macro Enable, TV, _Q, Timer
64      stl         Enable
65      jr h        _disa
66      stl         Timer
67      jr l        _endm
68          _FUPConv100msToTimer(TV, Timer)
69      jr          _endm
70  _disa:
71      ld          Timer
72              0
73      acc         1
74  _endm:
75      out         _Q
76      endm
```

The macro name of each FBOX must be exactly the same as the one of the definition file. As you can see on the above picture, each macro has parameters. The definition of each FBOX defines the parameters of the macro. The FBOX Builder generates the macros only based on the definition information. The order of the parameters is very important; we’ll see in details in this chapter what these parameters are linked to.

6.3.5 FBOX Display Text (.IDX)

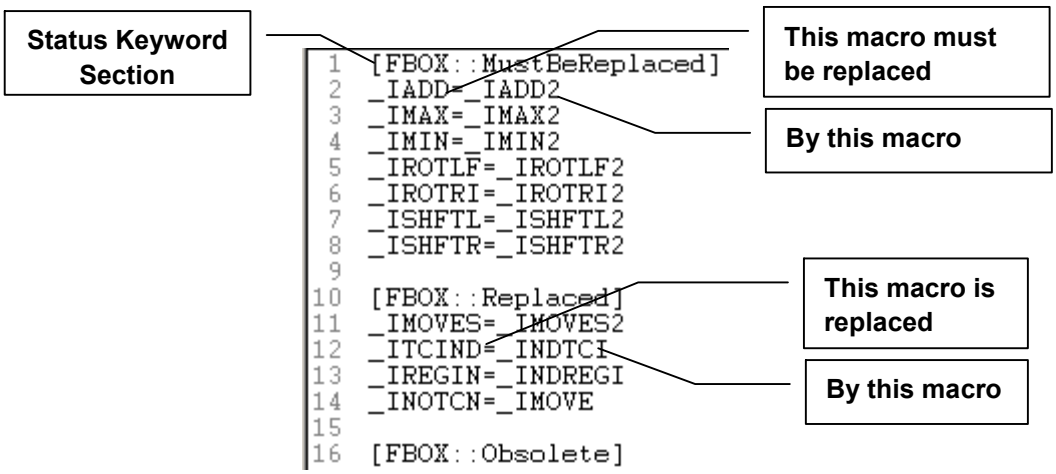
This file is mandatory and defines all the texts displayed on an FBOXES. The Name of the Family, the name of the Library help file and the name displayed in the FUPLA "FBOX Selector" are defined in the IDX file.
Some FBOXES have an Adjust And View window; this dialog window appears when you double click on an FBOX that has the down left corner black. This type of FBOX allows parameters settings online or offline. The ".IDX" file contains all the texts definition for this kind of configuration.
This file looks like this:



Like with the ".LIB" file described in the previous paragraph, the macro name of each FBOX of the ".IDX" file must strictly correspond to the one defined in the definition file. The FBOX Builder can generate this file.

6.3.6 FBOX Family Management (.SXG)

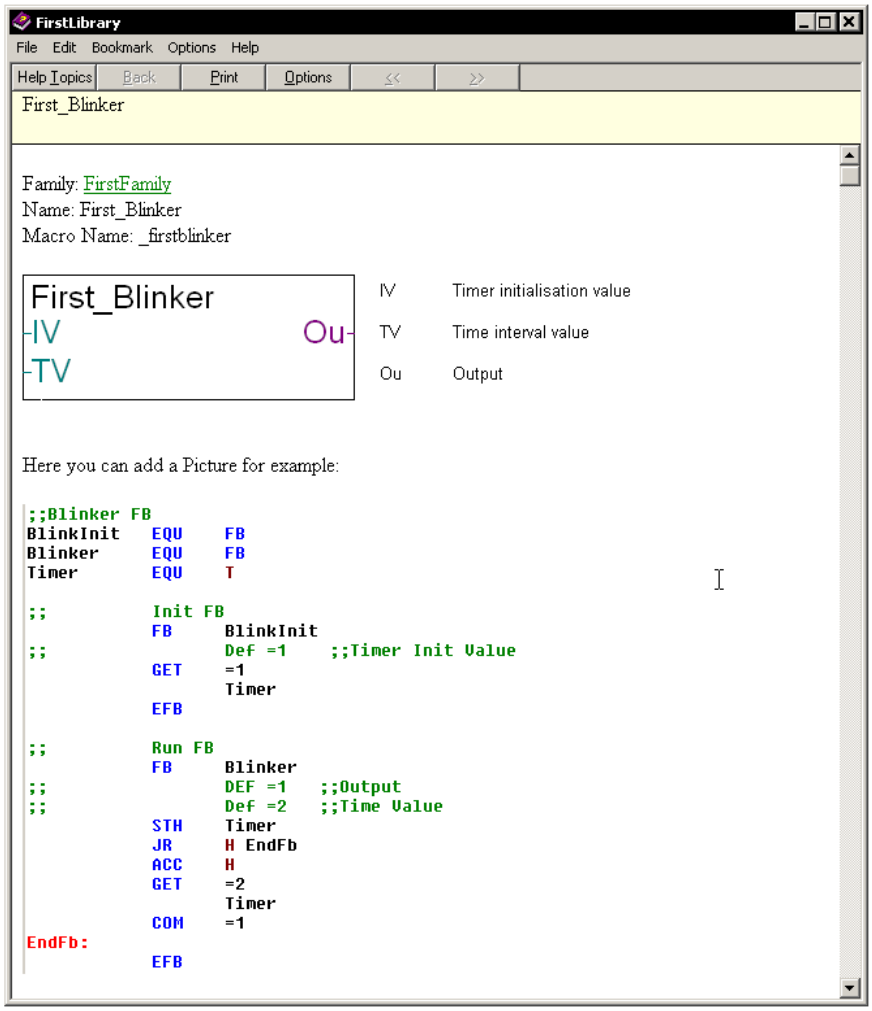
This file is not mandatory; it is used for the maintenance of an FBOX Library, especially for the Libraries that are distributed to manage the compatibility problems between versions. If an FBOX has been renamed, has been replaced, has been removed, has been moved to an other library or is obsolete, this information can be found in the ".SXG" file. The ".SXG" file of the integer family looks like:



6.3.7 FBOX Help (.HLP)

The FBOX Library help file is standard Windows help file. Basically the name of this help file must be mentioned in the ".IDX" files of the library and the keyword of each help topic is the FBOX name found in the ".IDX", if no name are defined the macro name is used.

The FBOX Builder has a Help File Editor, based on the comment of each FBOX parameter and the graphical interface, the FBOX Builder can generate the corresponding Library help file. This automatically generated Help file looks like this:



6.4 FBOX Definition Language

The FBOX definition language is a kind of configuration language. Here is the list of all the possible parameter needed to define an FBOX:

[*macro name*]
Width= <*width*>
Vers= <*version*>
Face= <*face name*>
FacePosi= <*left*¦*right*¦*center*>
Stretch= <*stretch format*>
GiveStretch= <YES¦NO>
In= <*in-format*>
Out= <*out-format*>
C= <*constant format*>
D= <*dynamic format*>
S= <*static format*>
avn= <*adjust-view format*>
IsKopla= <*YES*¦*NO*>
AcceptName=[<1|2>]
uName=[<*Name*>]
uComment=[<*Comment*>]
aRef==[<1|2>]
Ref=[<*Fbox Reference*>]
Exist=[<*Symbol*>]

The order of this list must be strictly respected.

From now on, we will take as example the FBOX *"PCD2.W34"* which is an FBOX for an analogue module. You can find this FBOX in the standard library in the "Analog Module" family.

The definition of this FBOX looks like this:

```
[_anad2w34]
Width=15
Face=PCD2.W34
Vers=2
Stretch=0 7 1 0
GiveStretch=YES
Out=i_NI
C=Add_Bio
D=Rd_2R
S=Fs_F LED_F
av0=b Fs 0 =1 %s
av1=c c11
av2=a Ch0 =10 %s {0,10,11,20,21,22}
av3=a Ch1 =10 %s {0,10,11,20,21,22}
av4=a Ch2 =10 %s {0,10,11,20,21,22}
av5=a Ch3 =10 %s {0,10,11,20,21,22}
av6=a Ch4 =10 %s {0,10,11,20,21,22}
av7=a Ch5 =10 %s {0,10,11,20,21,22}
av8=a Ch6 =10 %s {0,10,11,20,21,22}
av9=a Ch7 =10 %s {0,10,11,20,21,22}
```

A section begins with a line containing **[*name*]** and ends just before the beginning of the following section**.** The name of the section is the name located in the bracket.

Each lines contained in a section is called an entry. Each entry has a name followed by **=** and a string. **Between the „=" and the string absolutely no space are allowed.**

6.4.1 Macro Name (Section Name)

The name of the section is the name of the FBOX and also the name of its macro associated to it. Each entry in the section represents the different characteristics associated to the FBOX.

The macro name must be unique among FBOX Libraries and must not exceed 80 characters. A good tip to avoid naming clash with the macro name is to add the name of the company as for example: *"_SaiaMacroName"*.

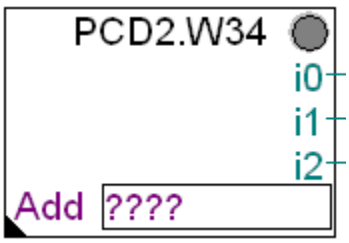
6.4.2 Width Entry

This field defines the width of the FBOX. The width should be calculated to take in consideration the title of the FBOX, the width of the input and output names and the availability of constants. When designing a new FBOX this parameter should be adjusted until you find the result expected. The range of the width is 1 to 17. This field is mandatory.

6.4.3 Face Entry

The face parameter indicates a string that should be displayed on the FBOX. This field is optional. If it is not used FUPLA will not display a face string on the FBOX. The max characters number is defined by the width of the FBOX in question.

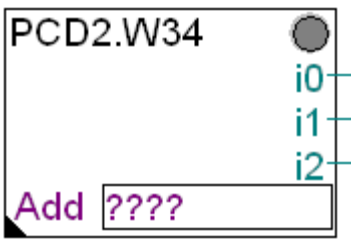
E.g. The face of the following FBOX is defined like this **Face=PCD2.W34:**



6.4.4 FacePosi Entry

This parameter indicates the position where FUPLA should display the face within the FBOX. The face can be justified on the left, right or centre of the FBOX if **FacePosi** is initialised with left right or centre respectively. By default the face is in the middle.

E.g.: The following FBOX is using a face justified on the left: **FacePosi=left**



6.4.5 Vers Entry

This field is optional. It associates a version number to the FBOX. This indication, when provided is passed as the first parameter to the macro. The range is 0 to 65534. 65535 and -1 are used internally to indicate that there is no version number specified.

E.g. Vers=2

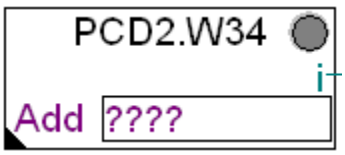
6.4.6 Stretch Entry

This entry is optional; if this entry is not defined the FBOX will not be stretchable. If present it indicates to FUPLA that the FBOX is stretchable. When a FBOX is stretchable the notion of index comes into consideration, and the inputs or outputs marked as stretchable becomes indexed, i.e. they will be duplicated a certain amount of time according to the stretch index chosen by the user. A stretch index of 0 indicates that an input or output marked stretchable is displayed only once. A stretch index of 1 indicates that an input or output marked stretchable is displayed twice...
When an input or output marked stretchable contains a label, it will automatically be appended with an index number.

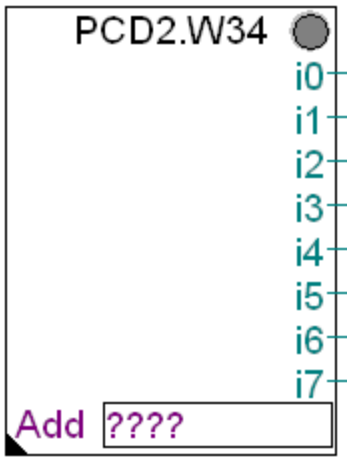
This line contains four fields with numbers. The first two fields is the minimum and maximum stretch index at which the user can stretch the FBOX. The third field is the increment at which an FBOX can be stretched. And the last field is the default stretch index.

E.g. The FBOX “PCD2.W34” is stretchable with index between 0 and 7 by index increment of 1 and is by default displayed with an index of 0. Its **Stretch** entry is: **Stretch=0 7 1 0**.

The following shows the FBOX “PCD2.W34” stretched with a factor of 0. This is the default stretch index for “PCD2.W34”.



The following shows the FBOX “PCD2.W34” stretched with a factor of 7. This is the maximum value available for this FBOX.



6.4.7 GiveStretch Entry

This entry tells the FUPLA compiler to place in the parameter list of the macro an indication of the stretch index. Sometimes this indication eases the macro construction of a stretchable FBOX. This entry should only be used for a stretchable FBOX. It is not commonly used. It should be set to **YES** to enable it.

E.g.: GiveStretch=YES

6.4.8 In Entry

This entry defines all the input parameters of the FBOX. If this entry is not present, the FBOX has no input. Each field in this entry describes an input.

Here is the general form of the **In** entry:

In=<input_descr> [<input_descr> ...]

<input_descr> =	**[<label>]_[E][N]<type>**
<label> =	**max. 3 characters**
<type> =	**[B]^[I]^[F]**

Each field is separated in two by an underscore. The first part is the optional label name that is associated to the input. It has a maximum of three characters. It will be shown when FUPLA displays the FBOX. If it is indexable an index number will be attached to it. The second part is the type description of the input. It can be binary, integer, or floating-point when marked with the letter **B, I** or **F** respectively. When the input is indexable the type letter should be preceded by the character **N**. Any binary inputs can also be edge triggered, i.e. they become active only during a rising edge is detected. When a binary input is edge triggered the letter **E** precedes the type letter.

E.g. The FBOX **BinInt** descriptor has the following **In** entry: **In=I_NB**. It defines only one input labelled **I** of type binary and it is marked indexable. The input will be marked with the letter **I** and will be followed by the index number.

This is the **In** entry of the **SRClk**: **In=S_B Clk_EB R_B**. Which defines three inputs. None of them is indexable. The first input is labelled **S** and is of type binary. The second input is labelled **Clk**, it is a binary edge triggered input. The last input is labelled **R** and is binary.

6.4.9 Out Entry

The **O** line includes the list of all the outputs coming out of a FBOX. This line is optional. When it is not present it indicates that there are no outputs. Each field on the **O** line describes an output.

Here is the general form of the **O** line:

Out= <output_descr> [<output_descr> ...]

<output_descr> =	**[<label>]_[N]<type>**
<label> =	**max. 3 characters**
<type> =	**[B]^[I]^[F]**

Each field is separated in two by an underscore. The first part is the optional label name that is associated to the output. It will be shown when FUPLA displays the FBOX. If it is indexable an index number will be attached to it. The second part is the type description of the output. It can be binary, integer, or floating-point when marked with the letter **B,** I or **F** respectively. When the output is indexable the character N should precede the type letter. An output connector cannot be edge triggered.

E.g. The FBOX **BinInt** descriptor has the following **O** output line: **O Out_I**. It defines only one output labelled **Out** of type binary. The output will be marked with **Out**.

6.4.10 C Entry

The **C** line includes the list of all the constant windows available on the FBOX. Constant windows are user-defined parameters that are passed to the FBOX. These parameters are absolute addresses to PCD medias or they are constants. They are also called in-out parameters because the macro can use them in read or write mode. This line is optional. When it is not present it indicates that there are no constant windows. Each field on the **C** line describes a constant window.

Here is the general form of the **C** line:

C= <constant_descr> [<constant_descr> ...]

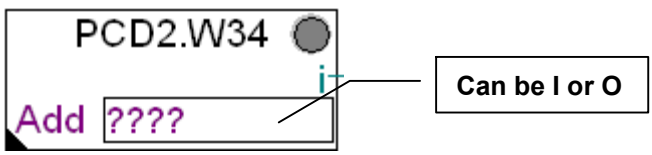
<constant_descr> =	**[<label>]_[N]<type><media>**
<label> =	**max. 3 characters**
<type> =	**[B]^[I]^[F]^[D]^[P]**
<media> =	**[i] ¦ [o] ¦ [f] ¦ [r] ¦ [t] ¦ [c] ¦ [k] ¦ [d] ¦ [x] | [p] | [u] | [s]**

Each field is separated in two by an underscore. The first part is the optional label name that is associated to the constant window. It will be shown when FUPLA displays the FBOX. If it is indexable an index number will be attached to it. The second part is the type description of the constant. It can be binary, integer, floating-point, Data or Program block when marked with the letter **B, I, F, D, or P** respectively. When a constant window is indexable the character N should precede the type letter. A constant window cannot be edge triggered. The media part of the constant description is an or-mask of all the medias that the user is allowed to use in the constant window. **i** for input, **o** for output, **f** for flag, **r** for register, **t** for timer, **c** for counter, **k** for constants, **d** for datablock, **x** for texts, **p** for PB, **u** for FB or **s** for SB. When the medias **d** or **x** are used the type **D** must be specified; when the media **p, u or s** are used, the type P must be specified.

E.g. The following descriptor:

...
C=Add_Bio
...

Represents the following FBOX:



6.4.11 D Entry

The **D** line includes the list of all the dynamic variables that are used within the FBOX macro. Dynamic variables are used to store intermediate results in FBOX that must make many calculation before they provide their results. This information is only important to the compiler so that it knows that it has to give temporary variables to the FBOX. These variables are not initialised and a FBOX must not expect them to contain any valid data. The same dynamic variable can be used by many FBOX, this is why

even if an FBOX initialise a dynamic variable, it must not expect that it will contain the same value next time it is called. Dynamic variables are owned temporarily by an FBOX, it is own from the instance it is called until it is finish executing. Only one FBOX can be executed at a time. This line is optional. When it is not present it indicates that there are no dynamic variables within the FBOX.

Here is the general form of the **D** line:

D= <dynamicvar_descr> [<dynamicvar_descr> ...]

<dynamicvar_descr> =	**[<label>]_[<count>]<media>**
<label> =	**max. 3 characters**
<count> =	**1..255 (default 1)**
<media> =	**[F]^[R]^[T]^[C]**

Each field on the **D** line describes a dynamic variable. It is separated in two sections by an underscore. The first section is the optional label name (maximum 3 chars.), which is associated to the dynamic variable. It is only for means of documentation in the source file generated at compilation.
The second part is the type description of the dynamic variable. It can be a flag a register, a timer or a counter when marked **F, R, C or T** respectively. These elements are chosen among the dynamic variables given in the resources used by FUPLA. Dynamic variables are not indexable, but many consecutive dynamic variables can be reserved for the FBOX simply by preceding the type description by the amount of variables necessary.

E.g. The FBOX “PCD2.W34” has the following **D** entry line:

D=Rd_2R

It defines one dynamic variable called “Rd” and it is a sequence of two consecutive registers. When the compiler will translate the FBOX “PCD2.W34” used in FUPLA in a source file it will reserve automatically two registers. These elements will also be shared with other FBOXES. So the “PCD2.W34” cannot initialise them and expect to find the same result the next time it is called.

6.4.12 S Entry

The **S** entry includes the list of all the static variables that are used within the FBOX macro. Static variables are used to keep a previous result. E.g. flip-flop or counter box. This information is important to the compiler so that it knows that it has to reserve some variables for a specific FBOX. This line is optional. When it is not present it indicates that there are no static variables within the FBOX.

Here is the general form of the **S** entry line:

S= <staticvar_descr> [<staticvar_descr> ...]

<staticvar_descr> =	**<label>_[<count>]<media>**
<label> =	**max. 3 characters**
<count> =	**1..255 (default 1)**
<media> =	**[F] | [R] | [T] | [C] | [X] | [D] | [^X] | [^D]**

Each field on the **S** line describes a static variable. It is separated in two sections by an underscore. The first section is the optional label name (maximum 3 chars.), which is

associated to the static variable. It is only for means of documentation in the source file generated at compilation.

The second part is the type description of the static variable. It can be a flag a register, a timer, a counter, a text in RAM/EPROM memory, a datablock in RAM/EPROM memory, a text in RAM memory or db in RAM memory when marked with **F, R, T, C, X, D, ^X, ^D** respectively. These elements are chosen among the dynamic variables given in the resources used by FUPLA. Static variables are not indexable, but many consecutive static variables can be reserved for the FBOX simply by preceding the type description by the amount of variables necessary.

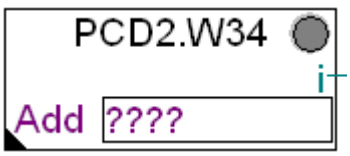
E.g. The FBOX “PCD2.W34” has the following **S** static variable line:

S=Fs_F LED_F

It defines two static variables. The first one is called **Fs** and it is a flag the second is called **LED** and it is also a flag. When the compiler will translate the FBOX “PCD2.W34” used in FUPLA to a source file it will reserve automatically two flags. These flags will not be available for any other FBOX calls.

6.4.13 The Static Variable LED

A static variable LED has a special meaning. It is a normal static variable except that a LED indicates automatically displayed on the top right corner of the FBOX the state of the variable.
When a FUPLA is Online with a PCD the LED is red if the value associated to the static variable LED is not 0, and it is green when its content is 0. When FUPLA is Offline this LED is shown grey.



This mechanism is useful to monitor FBOX error status.

6.4.14 Adjust And View Variables Entry

This entry is called adjust and view entry. It defines the internal variables of an FBOX which can be visualize or modified while the user program is being executed in a PCD. For each variable defined an **av** entry is defined. They should be numbered from **0** to **n**. FUPLA doesn't limit the number of **av** entries, but this resource consume much memory and should not be used too extensively. Many adjust variables can be joined together on the same line separated with the semi-colon character ';'. This saves FUPLA some access time.

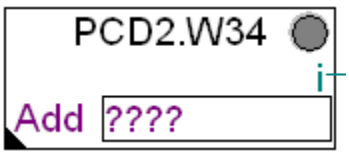
E.g.

av0=<*adjust-view format*>;<*adjust-view format*>;<*adjust-view format*>;
av1=<*adjust-view format*>...
av2=<*adjust-view format*>
...

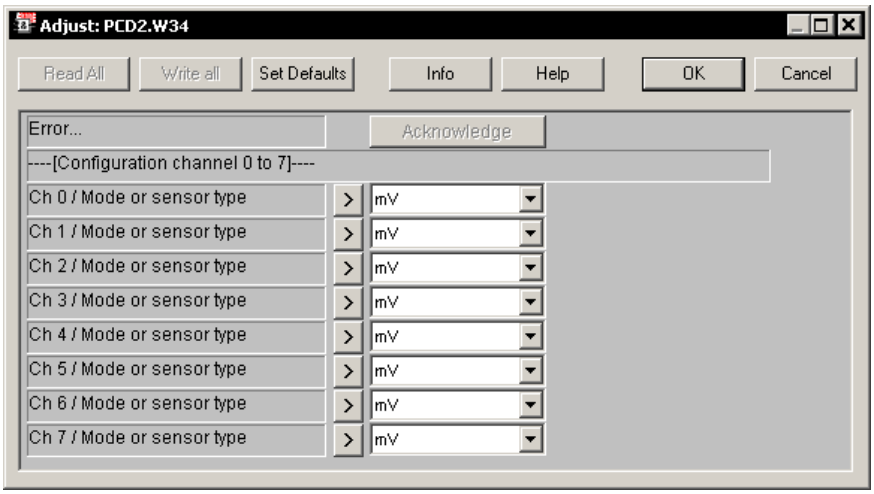
In our example, the Adjust and View entries are the following:

```
av0=b Fs 0 =1 %s
av1=c c11
av2=a Ch0 =10 %s {0,10,11,20,21,22}
av3=a Ch1 =10 %s {0,10,11,20,21,22}
av4=a Ch2 =10 %s {0,10,11,20,21,22}
av5=a Ch3 =10 %s {0,10,11,20,21,22}
av6=a Ch4 =10 %s {0,10,11,20,21,22}
av7=a Ch5 =10 %s {0,10,11,20,21,22}
av8=a Ch6 =10 %s {0,10,11,20,21,22}
av9=a Ch7 =10 %s {0,10,11,20,21,22}
```

These entries are used by FUPLA as information to display the FBOX Dialog window; when an FBOX use the AV feature, a black triangle cover the down left corner of the FBOX face in FUPLA as shown in the picture bellow:



In double clicking on this FBOX, the following dialog box will appear:



The structure of this dialog is defined by the AV entries of the definition file; but the text displayed are defined in the “FBOX Display Texts” (.IDX file) which we will see the details in the next paragraphs.

The AV of the FBOX in our concern are the following:

“av0” is a button that needs a static flag (Fs 0), a default value (=1) and a format with which the value should be displayed on the button (%s).

“av1” is a simple comment line in the dialog window which only needs a label (c11).

“av1..9” are offline Adjust variables that need a label (CHn), a default value (=10), a format with which the value should be displayed/enter (%s) and a range of value that can be chosen before compilation ({0,10,11,20,21,22}).

As you can see, each AV is a line of the dialog window. The button and the comment are followed by the 8-offline variables.

The following paragraphs detail all the possibilities of the Adjust And View Variables.

6.4.15 Description of a view variable using the avn field

Only static variables (the one marked with the letter S) can be viewed. They can be Flags, Timers, Counters or Registers. For each static variable that we want to view the user must enter the following entry at the end of the FBOX definition:

avn=v <label>[<index>][<%format>][\nabs]

avn =	Is the name of the entry. A number between 0 and n should replace the character n. The first entry **av** should be numbered 0 the following ones should be numbered successively.
<label> =	max. 3 characters that correspond to a static variable previously defined.
<index> =	an optional index can be appended to the label. It identifies one element within an array of static variables.
<%format> =	the format with which the value should be displayed.
<\nabs> =	specifies that the variable cannot be locked by the user.

The format can be one of the following:

- %s string format.
- %d decimal format. This is the default format.
- %x hexadecimal format.
- %o octal format.
- %f floating point format.
- %[0..9]p decimal floating point format. The digit indicates the position of the decimal point. The default is 0.
- %[0]D this format converts an integer value ....YYMMDD into DD/MM/YY.
- %1D this format converts an integer value ....MMDD into DD/MM.
- %2D this format converts an integer value …YYYYMMDD into DD/MM/YYYY.
- %[0]H this format converts an integer value ...HHMMSS into HH:MM:SS.
- %1H this format converts an integer value ...HHMM into HH:MM.
- %2H this format converts an integer value which represents seconds and converts it in the format HH:MM:SS. The Hours can be up to 99. This format can be used to represent delays before or delta of time between states.
- %t,%[0..3]t this format is used for entering special modem strings.
  %0t - actual format of modem string
  %1t - needed for GSM modems. Like %0t except ! replaced by +
  %2t,%3t – length is 2*8, 3*8 (not yet implemented)
- %0i IP address in dot notation
- %[1..5]e EIB formats for ETS2
  %1e – Full EIB address <main group 0..15><middle 0..7><sub 0..255>
  %2e – main group only 12/-/-
  %3e – middle and sub group -/4/123
  %4e – main and middle 12/4/-
  %5e – sub only -/-/123
- The modem format %t
  This format is used to enter some Modem specific strings and embed them into register values. The format is very simple and is the following:

CODE	DISPLAY	Comment
0-9	0-9	Number to dial
0AH	*	
0BH	#	
0CH	W	Commande ‘Wait for dial Tone’

0DH	,	
0EH	!	
0FH	Blank	Space, authorized before numbers and between numbers.

The modem library will contain two fields of 8 characters for the telephon number. Example:

Tf number prefix “ 0W 037” => value in register: FF0CF037
Tf number “75 39 88” ===> value in register: 75F39F88

6.4.16 Description of an Online adjust variable using the avn field

Online adjust variables are internal static variables of a macro which can be modified and visualised by the user. Online variables are associated with an internal static variable; they can be registers, flags, counters or timers.

When FUPLA is Online with a PCD it will be able to modify their value on user request. The value is then transferred in its corresponding static variable.

Only Online variables (the one associated with an internal static variable in the S entry) can be adjusted in real time. They can be Flags, Timers, Counters or Registers.

With the adjustable Online variable we give the user a chance to view and modify the content of a given FBOX internal variable. For these variables an adjust edit window and an actual set value window are displayed in the adjust list window. The adjust edit window contains the value which the user wants to pass to the internal variable. The > button on the left order FUPLA to transfer the default value in the adjust edit window. The > button on the right of the adjust edit window order FUPLA to transfer the value from the adjust edit window in the corresponding variable in the PLC. The Actual value window shows the actual content of the PLC variable when possible (when in Online).

If we are Online: the button > on the left of the adjust edit window will be enable (to show that we can modify the effective value) and the right window will contain the effective value.

If we are Offline: the button > on the left of the adjust edit window will be disable (to show that we can't modify the effective value) and the right window will contain --------.

The following describes the syntax used for defining an Online adjust variable:

avn=a <label>[<index>][<%format>] [<=init>] [{<range>}][\nabs][\wax=*wacces*]

avn =	Is the name of the entry. A number between 0 and n should replace the character n. The first entry **av** should be numbered 0 the following ones should be numbered successively.
<label> =	max. 3 characters that correspond to a static variable previously defined.
<index> =	an optional index can be appended to the label. It identifies one element within an array of static variables.
<%format> =	the format with which the value should be displayed and how they should be entered by the user. It is the same as the format described in section **Description of a view variable using the avn field.**

<=init> = initial value. It must be definable for each parameter independently specially for arrays like x_10R. Default is 0 if no range is given else init is at the middle of the range. The init value can be entered with decimal or with hexadecimal format using 0X or 0x preceding the number. Example the hexadecimal value FFF can be entered like this =0xfff.
The init segment has the following syntax: DefaultValue[:SetValue]. When the SetValue is not specified it is assigned with the value of the default value. The set value is the value which is placed in the adjust field the first time. The default value is the value which is placed in the adjust field when the user presses the set default button.

<range> = The range is taken without the format attribute. This parameter is optional.

\nabs = specifies that the variable cannot be locked by the user.

\wax=*waccess* This defines the write access level needed to modify any value in the PCD. For more details see section **Limiting Write Access.**

The format of the range is the following:

<range> = <subrange>[,<range>]
<subrange> = value[..value]

E.g.	{10..20}	range: 10 to 20.
	{10, 15..20}	range: 10 or 15 to 20.
	{10..12,15..20,0..5}	range: 10 to 12 or 15 to 20 or 0 to 5.

6.4.17 Description of an Offline adjust variable using the avn field

These are constants that are passed to a macro (they are passed directly, they are not passed via a register or a flag like the Online variables). They are used for initialising initial parameters in the FBOX. They use up less dynamic variables and they are not fragile to a bad initialisation like Online parameters are. They can only be defined Offline, and the user must recompile and re download his program.

For these variables only an adjust edit window is available. The adjust edit window is displayed in the adjust list window. The adjust edit window contains the value which will be pass to the macro the next time the program is recompiled. The  button on the left order FUPLA to transfer the default value in the adjust edit window.

Adjustable Offline variables, gives the user a chance to view and modify the constant which is passed to a macro. If the user modifies an Offline variable the editor will suggest to him to re-compile, since its modification will only be effective if re-compiled.

The syntax of an Offline adjust variable is the same as the Online variable, except that the label must not correspond to any static variables previously defined in the **S** entry.

6.4.18 Description of a FBOX user button using the avn field

FBOX user buttons are associated to internal static variables of an FBOX; they can be registers, flags, counters or timers. When a user presses a FBOX button, FUPLA transfer its default value to its corresponding internal static variable. This can be done only when FUPLA is Online with a PCD, otherwise the button is disabled. This is a very convenient way to set or reset parameters or algorithm in an FBOX.

The following describes the syntax used for defining an FBOX user button:

avn=b <label>[<index>][<%format>] [<=init>][\nabs] [\wax=*waccess*]

avn =	Is the name of the entry. A number between 0 and n should replace the character n. The first entry **av** should be numbered 0 the following ones should be numbered successively.
<label> =	max. 3 characters that correspond to a static variable previously defined.
<index> =	an optional index can be appended to the label. It identifies one element within an array of static variables.
<%format> =	the format with which the value should be displayed on the button. It is the same as the format described in section **6.4.15 Description of a view variable using the avn field.**
<=init> =	initial value. Default is 0. This is the value that is shown on the button. The init value can be entered with decimal or with hexadecimal format using 0X or 0x preceding the number. Example the hexadecimal value FFF can be entered like this =0xfff.
\nabs =	specifies that the variable cannot be locked by the user.
\wax=*waccess*	This defines the write access level needed to modify any value in the PCD. For more details see section **Limiting Write Access**.

6.4.19 Description of a FBOX user button with a view window

This is a button like defined in section **Description of an FBOX user button using the avn field.** The only difference is that a view window can also be associated to it.

The following describes the syntax used for defining a FBOX user button with a view window:

avn=bv <label>[<index>][<%format>] [<=init>][\nabs][\wax=*waccess*]

The view window will show the content of the variable associated to the button. The user can also view an alternate view of another variable see section **Defining an alternate view variable**.

6.4.20 Defining an alternate view variable

This feature is used to show in the view window an alternate variable as the one that was previously defined in the current adjust variable. This feature is exclusively used today for button with view; see section **Description of an FBOX user button with view window.**
The following describes the syntax used for defining a FBOX user button with an alternate view window:

avn=bv <label>[<index>][<%format>] [<=init>][\nabs][\wax=*waccess*] [\v…]

Where **\v…** is the same syntax as described in section **Description of a view variable using avn field,** preceded by a backslash '\'.

When the user locks the absolute address it will lock the alternate view variable not the button variable.

The text column is always the one associated with the button not the one associated to the alternate view.

6.4.21 Description of a comment field using the avn field

Comments are used in an Adjust Dialog window to comment or create sections in a list of adjust and view variables. Double clicking on these fields gives the description associated with the variable.

The following describes the syntax used for defining comment in an adjust dialog window:

avn=c <label>

The syntax of an Offline adjust variable is the same as the Online variable, except that the label must not correspond to any static variables previously defined in the **S** entry.

avn = Is the name of the entry. A number between 0 and n should replace the character n. The first entry **av** should be numbered 0 the following ones should be numbered successively.

<label> = max. 3 characters that must not correspond to any static variables previously defined in the **S** entry.

The comment and the description of this line is located in the **.IDX** file.

6.4.22 Limiting Write Access

It is maybe required to limit write access to some memory in the PCD while the PCD program is in run. Previous implementation of the adjust variable allowed anyone to modify the content of the PCD memory by using the adjust window of a FBOX. Now with the new **\wax=** keyword the FBOX designer can set a level of protection.

We store in the file {appsdir}\pg4limit.ini the current write access level. This is stored in the section **[Security]** entry **WriteAccessLevel=**. The value allowed for the current write access is between 1 and 99. If this entry is not defined a default of 50 is taken. For mean of conveniences we can also set this variable with a lower value by setting the variable WriteAccessLevel found in the registry in the section [HKEY_CURRENT_USER\Software\SAIA-Burgess\PG4\2.0\SFUP32\Security].
The maximum WriteAccessLevel is the one defined in the pg4limit.ini file - this file is protected therefore it does not become a back door access.

The keyword **\wax=** can be used on any adjust variable line. It only make sense to use it for variables which can be modified by a button, i.e.: it makes no sense to use **\was=** on a comment an offline variable or a view only variable.

When the write access level defined by the **\wax=** directive is lower than the current one defined in the file spg4.ini than any button giving the possibility to write in the PCD variable is removed from the adjust window. The range for the **\wax=** directive is between 1 and 99. When it is not used the write access level of the adjust variable is set to 0. When an adjust variable has a write access equal to 0 it means that there are no restrictions.

6.4.23 The format string

Adjust variables which use the string format will be provided with a pick list (COMBOBOX) fill with choices that are available for the adjustment of the variable. These adjust variable must absolutely provide a range on there **avn** entry line definition.

6.4.24 AcceptName Entry

Values: AP_NOT_USED = 0, AP_OPTIONAL = 1, AP_REQUIRED = 2, AP_USERDEFINED = 3,AP_DISABLED = 4

This entry informs FUPLA that the macro was designed to take advantage of this mechanism, and it is passed in the parameter list. If this entry is set to 1 the user can enter a name if he wants, if he does not FUPLA will generate one for him. If AcceptName is set to 2 then FUPLA will notify the user if he does not set the user name. If AcceptName is set to 1 or 2 the user name will be passed to the macro. If AcceptName is not defined then FUPLA does not pass the FBOX user name to the macro.
If AcceptName is 3 the user can define a symbol name, and this name as group name will be used for internal variables. The name must be unique in the module.
AcceptName = 4 means that the user cannot specify the name.

6.4.25 uName Entry

This entry is used to memorise or to specify a default user name. This should never be bigger than 80 characters. Only valid symbols accepted by the assembler or positive numbers will be accepted (AP_OPTIONAL, AP_REQUIRED, AP_NOT_USED).

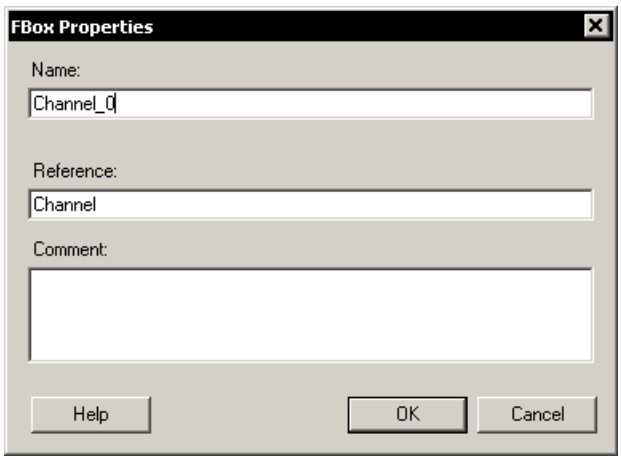
6.4.26 aRef Entry (Accept Reference)

This entry informs FUPLA that the macro was designed to take advantage of this mechanism, and it is passed in the parameter list. If the entry *aRef* is set to 1 the user can specify a FBOX Reference if he wants, if he does not FUPLA will generate one for him. By default FUPLA generates __NO_FBOX_REFERENCE__. If *aRef* is set to 2 then FUPLA will notify the user if he does not set the FBOX reference. If *aRef* is set to 1 or 2 the user name will be passed to the macro. If *aRef* is not defined then FUPLA does not pass the FBOX reference to the macro.

6.4.27 Ref Entry

This entry is used to memorise or to specify a default FBOX Reference. This should never be bigger than 12 characters. Only valid symbols accepted by the assembler or positive numbers will be accepted.

The user can specify a new FBOX reference from the dialog box shown below. This dialog box is accessible from the menu **FBOX Properties…** by pressing the right mouse button on the FBOX desired.



6.4.28 UComment Entry

This entry is used to memorise the user comment entered by the user for a specific FBOX. This is normally not used by the FBOX designer. Nevertheless it can be used to define a default comment. The maximum characters of this string are 80.

6.4.29 Exist Entry

This is a directive which tells FUPLA to define a symbol made of the macro name or the symbol name if specified in the exist directive. This symbol is then equated to the number of time the FBOX was used. This symbol can than be used by another FBOX to verify if an FBOX is in use or not.

The syntax is:

Exist=[*Symbol*]

These symbols are placed into a group called __FBOX_IN_USE__ in the file .inc to limit collision of symbols.

NOTE: The symbol should not be used to define a common name to a group of different FBOX. We use it to define a common name for different version of the same FBOX.

The compiler will fill the file .inc with a section group __FBOX_IN_USE__. For example:

```
$GROUP __FBOX_IN_USE__
_fbox1          EQU    2
_fbox2          EQU    1
$ENDGROUP
```

The compiler detected in this example 2 occurrences of FBOX _fbox1 and 1 occurrence of FBOX _fbox2. Other macro can access these symbols with the following syntax:

$ifdef __FBOX_IN_USE__. _fbox1

See also the documentation on assembler directive $GROUP.

6.5 FBOX Texts Definition

The "**.IDX"** file contains all the strings associated to the FBOX interface. It contains:

- Library name that appears in the family FBOX list.
- FBOX name that appears in the FBOX list.
- Name and description of each **avn** variables.
- Strings used for translating the **string** format.

There should be for each **.DEF** file a corresponding **.IDX** file. **.IDX** files have been design to ease the translation of an FBOX in an other language. If the **.IDX** file is corrupted the FBOX will still behave properly. The general format of the **.IDX** file is the following:

```
[__Library]
Name=Name of library
...
        [macro name 1]
        Name=Name of FBOX
        LBL= <adjust-view idx format>
        LED= <adjust-view idx format>
        REG 0= <adjust-view idx format>
        ...
        [macro name 2]
        ...
```

6.5.1 Library section

The first section of the **.IDX** file must be named __Library (two underscore preceding Library). In this section FUPLA finds the name of the library in the entry **Name**. This is the name that is displayed in the list of libraries.

6.5.2 Macro sections

Each macro has a corresponding section in the **.IDX** file of there families. The name of each section is the same as the one defined in the corresponding **.DEF** file. Each one of these sections contains a **Name** entry a **Help** entry and an entry associated to each variable that were associated to the **avn** section in its corresponding **.DEF** file.

6.5.3 Name Entry

The entry **Name=** defines the name of the FBOX. This name will appear in the FBOX list box.

6.5.4 Variables Description Entry

A variable description entry gives a name and a description to an **avn** variable. It can also contain the string used for translating the string format. The name of the entry is the name of the label used in the **.DEF** file in the **avn** entry plus optionally the index that was used.

The format of a Variable description entry is the following:

<LBL> [<index>]=[<text-name>][\R)<text-description>][\n)<text-value0>]

<LBL> =	this is the label that was used in the **.DEF** file to identify the variable associated with the **avn** variable.
<index> =	This parameter must be used if it was used in the **avn** entry of the corresponding **.DEF** file.
<**text-name> =**	Is a string which will be placed on the left most column of the adjust list window.

\R)<text-description> = This is the text describing in more detail the variable. It can be accessed by double clicking the left most column of the adjust list window. It must be preceded with the characters **\R)**

\n)<text-valuen> = This is the text which is displayed when a value **n** is given to this variable if it has used the string format. It must be preceded with the character **\n),** where **n** is the value which is associated with this text.

6.5.5 Example with the PCD2.W34 FBOX

To understand exactly what we have to do in the .IDX file, the best way is to put side by side the .DEF and the .IDX as follow:

```
[_anad2w34]
Width=15
Face=PCD2.W34
Vers=2
Stretch=0 7 1 0
GiveStretch=YES
Out=i_NI
C=Add_Bio
D=Rd_2R
S=Fs_F LED_F
av0=b Fs 0 =1 %s
av1=c c11
av2=a Ch0 =10 %s {0,10,11,20,21,22}
av3=a Ch1 =10 %s {0,10,11,20,21,22}
av4=a Ch2 =10 %s {0,10,11,20,21,22}
av5=a Ch3 =10 %s {0,10,11,20,21,22}
av6=a Ch4 =10 %s {0,10,11,20,21,22}
av7=a Ch5 =10 %s {0,10,11,20,21,22}
av8=a Ch6 =10 %s {0,10,11,20,21,22}
av9=a Ch7 =10 %s {0,10,11,20,21,22}

[_anad2w34]
Name=PCD2.W34
Fs 0= Error\1)Acknowledge\R)Reset the error LED
c11=----[Configuration channel 0 to 7]----
Ch0=Ch 0 / Mode or sensor type \0)1:1 \10)mV \11)uA \20)Pt 1000 \21)Ni 1000 \22)Ni 1000 L&S
Ch1=Ch 1 / Mode or sensor type \0)1:1 \10)mV \11)uA \20)Pt 1000 \21)Ni 1000 \22)Ni 1000 L&S
Ch2=Ch 2 / Mode or sensor type \0)1:1 \10)mV \11)uA \20)Pt 1000 \21)Ni 1000 \22)Ni 1000 L&S
Ch3=Ch 3 / Mode or sensor type \0)1:1 \10)mV \11)uA \20)Pt 1000 \21)Ni 1000 \22)Ni 1000 L&S
Ch4=Ch 4 / Mode or sensor type \0)1:1 \10)mV \11)uA \20)Pt 1000 \21)Ni 1000 \22)Ni 1000 L&S
Ch5=Ch 5 / Mode or sensor type \0)1:1 \10)mV \11)uA \20)Pt 1000 \21)Ni 1000 \22)Ni 1000 L&S
Ch6=Ch 6 / Mode or sensor type \0)1:1 \10)mV \11)uA \20)Pt 1000 \21)Ni 1000 \22)Ni 1000 L&S
Ch7=Ch 7 / Mode or sensor type \0)1:1 \10)mV \11)uA \20)Pt 1000 \21)Ni 1000 \22)Ni 1000 L&S
```

av0:
This entry is a button (b); its responsibility is to send its default value to its attached flag when it is pushed down.

- The definition line “S=Fs_F …” ask FUPLA to reserve a flag especially for that FBOX (static).

- The definition line “av0=b Fs 0 =1 %s” define a button and link its action to the Fs flag (“0” is the media index in the case Fs is the base address of an array of flags. In this case the “0” is not needed). The default value of the attached flag is 1 (=1) and finally the value of the flag can be read on the button in a string format (%s). In our case, the caption of the button could have another string when the flag is 0.

- The string definition line “Fs 0=Error \1)Acknowledge \R)Reset the error LED” is the strings that fully describe the button. “Error” is the comment situated on the left of the button, “Acknowledge” is the caption of the button and the “\R)Reset the error” is the help string that is popped up when double clicking on the comment.

av1:
This entry is a simple comment (c) that will be displayed in the second line of the FBOX Adjust and View dialog window.

- The definition line “av1=c c11” simply define the second line as comment and the comment label that is the link with the string definition is “c11”.

- The string definition line “c11=----[Configuration channel 0 to 7]----“ define the effective text that will be displayed in the FBOX dialog window.

av2…av9:
These entries are offline adjust variables (a); the “a” is also used for online variable, the only difference is that for the off line’s, the variables don‘t correspond to PCD media but are passed directly as constant. The user before compilation of the FUPLA file must define Offline variables.

- The definition line “av2=a Ch0 =10 %s {0,10,11,20,21,22}” define “Ch0” as the label of the av, the default value “=10”, the display value format to be a string (%s) and the range of possible values ({0,10,11,20,21,22}). The FBOX dialog window will have a combo box in which the user can choose the value he wants.

- The string definition line “Ch0=Ch 0 / Mode or sensor type \0)1:1 \10)mV \11)uA \20)Pt 1000 \21)Ni 1000 \22)Ni 1000 L&S” define the description text of the av (“Ch x / Mode or sensor type”) and each possible choice with their values (…“\21)Ni 1000”…).

6.6 FBOX Macro Definition

The ".LIB" file contains a macro for each FBOX defined in the ".DEF" file. To understand exactly the macro that belongs to our example FBOX we have to compare the definition and the macro header of the "PCD2.W34" FBOX.

```
[_anad2w34]
Width=15
Face=PCD2.W34
Vers=2
Stretch=0 7 1 0
GiveStretch=YES
Out=i_NI
C=Add_Bio
D=Rd_2R
S=Fs_F LED_F
av0=b Fs 0 =1 %s
av1=c c11
av2=a Ch0 =10 %s {0,10,11,20,21,22}
av3=a Ch1 =10 %s {0,10,11,20,21,22}
av4=a Ch2 =10 %s {0,10,11,20,21,22}
av5=a Ch3 =10 %s {0,10,11,20,21,22}
av6=a Ch4 =10 %s {0,10,11,20,21,22}
av7=a Ch5 =10 %s {0,10,11,20,21,22}
av8=a Ch6 =10 %s {0,10,11,20,21,22}
av9=a Ch7 =10 %s {0,10,11,20,21,22}
```

```
_anad2w34 macro     vers,
                    in0,in1,in2,in3,    ;;output FBox
                    in4,in5,in6,in7,
                    addr,               ;;constant
                    Rd,                 ;;Dynamics
                    Fs,led,             ;;static
                    Quit,               ;;Quittance error button
                    Ch_0,Ch_1,Ch_2,Ch_3,  ;;Channel type choice register
                    Ch_4,Ch_5,Ch_6,Ch_7,
                    index               ;;stretch index
        …
endm
```

You can see in the above example that the order is very important because only this order makes the link possible between the macro and the FBOX definition. The symbols used as macro parameters are not important, of course they must comply to the symbol syntax.