

# autoSIM-200

## The shortest way to automation



# USER MANUAL





## Installation

If you are installing AUTOSIM from the AUTOSIM CD-ROM, place it in your CD-ROM drive.

The installation is launched automatically.

If this does not occur, launch the “Setup.exe” executable which is in the CD-ROM root.

## Configuration required

PC compatible computer, with:

- WINDOWS 98 SE or WINDOWS ME or WINDOWS 2000 or WINDOWS XP or WINDOWS 7 or WINDOWS 2003 or WINDOWS VISTA operating system,
- 256 MB memory (depending on the operating system: the operating system itself may require more memory),
- graphics board with a resolution of at least 1024 x 768 in 65536 colors.

## Installation in a network

AUTOSIM can be installed in a network environment.

Execute the installation procedure on the “server” PC (make sure you have all of the access rights when you carry out the procedure).

To launch AUTOSIM, on the client PCs, create a shortcut to the “autom8.exe” executable of the AUTOSIM installation directory on the server PC.

Refer to the chapter “additional information on installing AUTOSIM in a network environment” for more information about installing AUTOSIM and licenses in a network environment.

## New features of AUTOSIM<sup>3</sup>

### Increased integration of the Grafcet 60848 standard

The new elements of this standard can now be accessed in the contextual program editing menus.

### Compatibility of files

The files generated by all of the AUTOSIM<sup>3</sup> versions can be re-read by all of the AUTOSIM<sup>3</sup> versions.

### Physical engine integrated to IRIS3D

The TOKAMAK motor is integrated to IRIS3D. This enables an extremely realistic simulation of the 3D operational units to be obtained.

### Enhanced 3D object handling in IRIS3D

The saving and re-reading of objects and behaviors allows you to manage libraries of easily reusable objects. Predefined objects (cylinders, conveyor belts, etc) are proposed as standard. A 3D operational unit simulation application can now be created in just a couple of mouse clicks.

### Improved links between AUTOSIM and IRIS3D objects

Enhanced modes allow you to easily handle displacements of complex objects between AUTOSIM and IRIS3D. An AUTOSIM variable can, for example, give the speed of an object directly. Position reporting can also be simulated in the manner of an absolute encoder.

### Textured IRIS3D objects

Textured objects now provide IRIS3D with extraordinarily realistic rendering.

## Drag and drop from IRIS3D to AUTOSIM sheets

A right click on the IRIS3D objects allows you to access the list of variables and “drag” a reference over to a programming sheet.

## SIMULA user-definable object

SIMULA users will appreciate the new user-definable object, which will allow you to create your own objects.

(See the section of this manual devoted to SIMULA)

## Drag and drop from SIMULA to AUTOSIM sheets

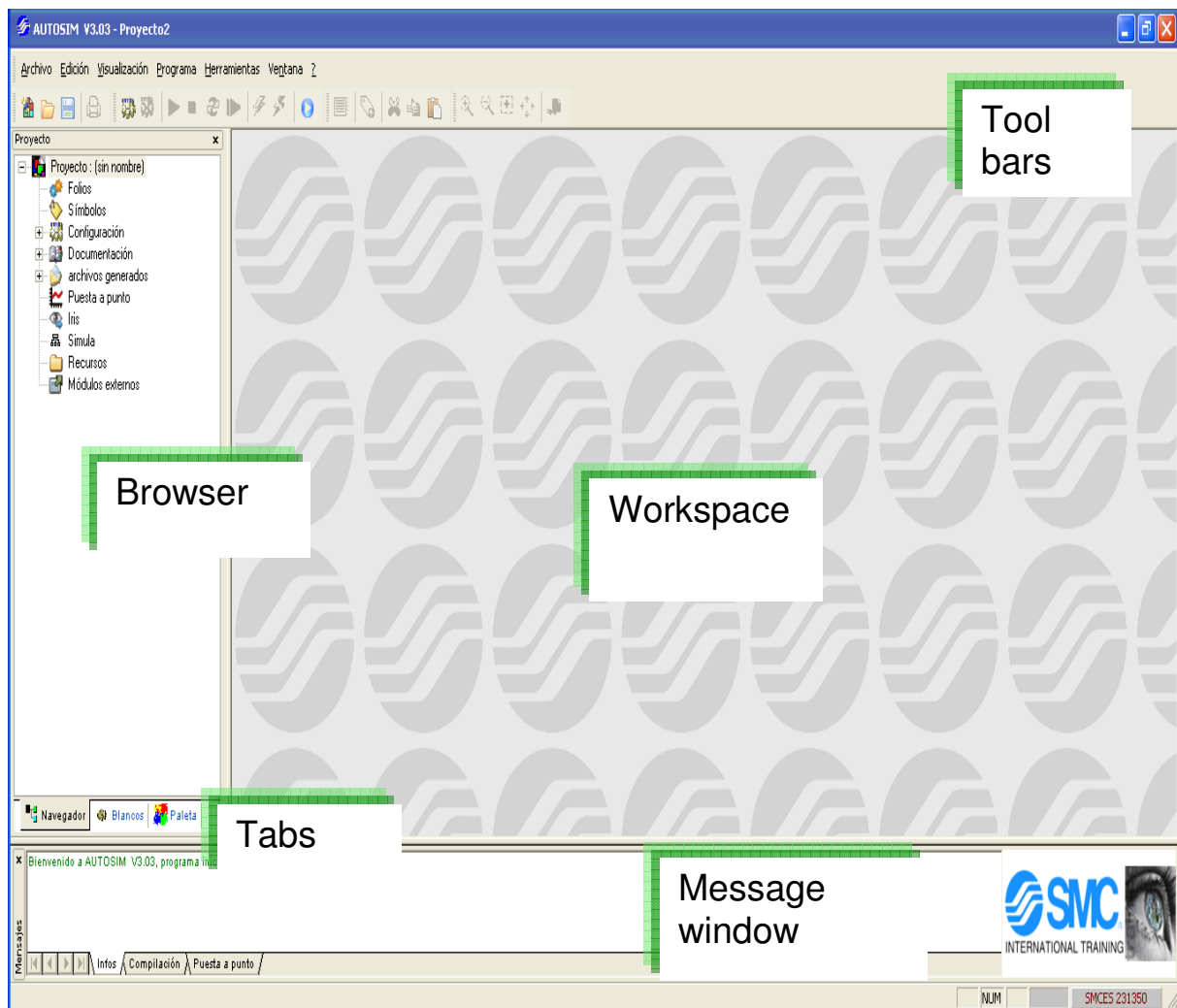
A click on the SIMULA objects allows you to “drag” a reference over to a programming sheet.

## Improvements to the environment


Finally, numerous improvements to the environment, such as the magnifying glass in the design palette, the simplified palettes in “beginner” mode, or personalizing menus make AUTOSIM even more user-friendly.

# Environment

## General views



*AUTOSIM's main window in "Expert" mode*

The environment is fully customizable. The tool bars can be moved (by dragging their moving handle ) and parameterized (menu "Tools/Customize the environment").

The state of the environment is saved when you quit it. This state can also be saved in a project file (see the project options).

## Selecting targets in expert mode

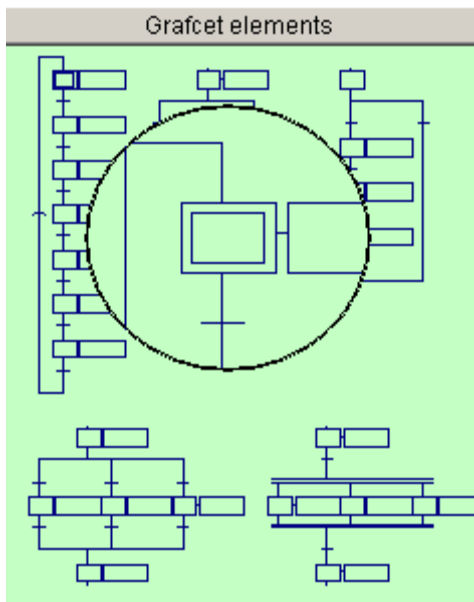
At the bottom of the browser window there is a “Targets” tab, allowing access to the list of post-processors installed.

Current	Name	Version
	PC	8.000
	PI7 (Tsx 37 & Ts...	8.000
	PL72	8.000
	STEP7 (S7200)	8.000
	ABB	8.000
	GE-FANUC	8.000

The active target is indicated with a red tick. Access to targets displayed in grey is not authorized for the license installed (see the “Licenses” chapter for more details). To change the current target, double-click on the corresponding line. The targets shown in this list are the ones selected at installation time. If the target you want to use is not shown in this list, re-launch the AUTOSIM installation and install it.

## Palettes

At the bottom of the browser window there is a “Palette” tab, allowing access to program design elements.



The palette gives a set of elements that can be selected and placed on the sheets. To select an element, left-click with the mouse in the palette, expand the selection, release the mouse button, click in the area selected and move the area towards the sheet.


The palette also contains the list of symbols for the project. You can grab them and drag them onto a test or an action on a sheet.

A magnifying glass is automatically shown when the elements displayed are small.

## Displaying or hiding the project window or message window

Select the « Project » or « Messages » option from the « Window » menu.

## Displaying the work space in full screen mode

Select the « Full screen » option from the « Display » menu. Click on  to exit full screen mode.

## Keyboard shortcuts

Keyboard shortcuts are written in the menus. « Masked » shortcuts can also be used:

CTRL + ALT + F8	Save the project in executable format
CTRL + ALT + F9	Save the project
CTRL + ALT + F10	Access project properties
CTRL + ALT + F11	Display or hide AUTOSIM window

Parameters can be set for the entire environment; its state is saved when you close AUTOSIM. Environment windows can be hidden. The « Windows » menu is used to display them again. The work space can be displayed in full screen mode. The tabs at the bottom of the browser window are used to access selection for the current post-processor and the graphics palette.



## Licenses

A license establishes AUTOSIM user rights. The following elements are established by license:

- the number of all or none inputs/outputs that can be used,
- post-processors that can be used,
- the number of users (only for network licenses).

### Registering a license

When you install AUTOSIM, you can use it for free for a period of 40 days.

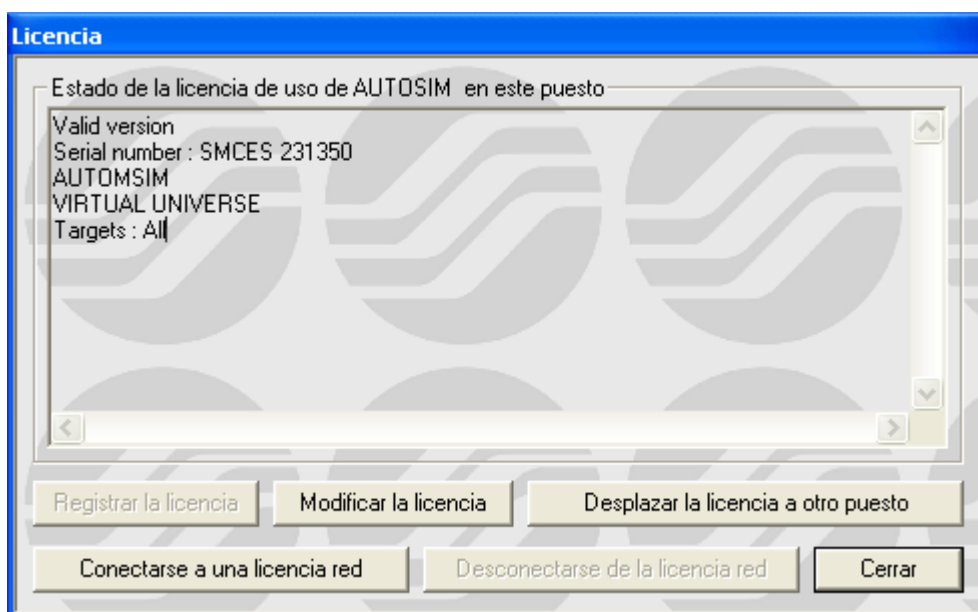
You must register your license within 40 days.

To register your license, send SMC:

- the serial number printed on the label glued to the software box, or the reference of your delivery note or order form,
- the user code provided with the software indicating the PC where you have installed the product.

You will then receive an enable code (also called validation code).

The « License » option in the AUTOSIM « File » menu can be used to display the status of your license and obtain a user code (click on « Registering the license »).



*License status.*

A user code is valid for a period of 10 days.

So a maximum period of 10 days can pass from when you send a user code to SMC and when you receive an enable code provided by SMC.

## Sending a user code to SMC

There are various methods you can use. Exchanging codes by e-mail is highly recommended as it limits the risk of error.

A single error in the code will prevent the license from being registered.

## Sending a file by e-mail (the best solution)

Guardar o modificar una protección

Ahora podrá registrar su licencia. Para ello siga estos pasos:

- Haga click en "Registrar vía Web".
- Introduzca el número de serie (Serial number) y la contraseña (Password) proporcionadas. El código de usuario (User code) aparecerá automáticamente; de lo contrario copielo en esta pantalla y péguelo.
- Haga click en "Validate". Copie en código que aparezca, y péguelo en esta ventana haciendo click en "Pegar un código de validación desde el portapapeles".

Ante cualquier problema, contacte con nosotros en [training@smctraining.com](mailto:training@smctraining.com)

**CODIGO USUARIO, ATENCION : '0' ES CERO Y 'O' ES LA LETRA**

HQH2K	75AS3	UK918	12G2Q	UQNUL	VE2BR	81202	AJQLS	4NGKK	R8C3R	C0
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	----

Copiar el  
código de  
usuario en el  
portapapeles

Pegar un código  
de validación  
desde el  
portapapeles

Obtener un  
nuevo código

Código de validación

--	--	--	--	--	--	--	--	--	--	--

Anular

Validar

*License registration dialogue box*

To generate a file containing your user code, click on « Save user code in a file ». You can then transmit the file with « .a8u » extension as an attachment and send it to the address [training@smctraining.com](mailto:training@smctraining.com).

### Copying the user code in an e-mail message

By clicking on « Copy user code to clipboard », you can then paste the code in the body of the message and transmit it to the e-mail address [training@smctraining.com](mailto:training@smctraining.com).

### By telephone (highly undesirable)

By telephoning +34 945 00 10 33. Be sure to differentiate between the letter « O » and number zero. Be careful of consonants which are difficult to tell apart on the telephone (for example « S » and « F »).

## Entering the validation/enable code

### Validating by a e-mail received file

If you have received an « .a8v » file by e-mail, save the file on your hard disk, click on « Read a validation code from a file » and select the file.

### Validating for a code received in the text of an e-mail

Select the code in the message text (make sure you only select the code and do not add any spaces to the end). Click on « Paste a validation code from the clipboard ».

### Validating for a code received by fax or telephone

Enter the code in the spaces under the title « Validation code ».

## Modifying a license

Modification of a license Involves changing the elements authorized by the license (for example adding a post-processor).

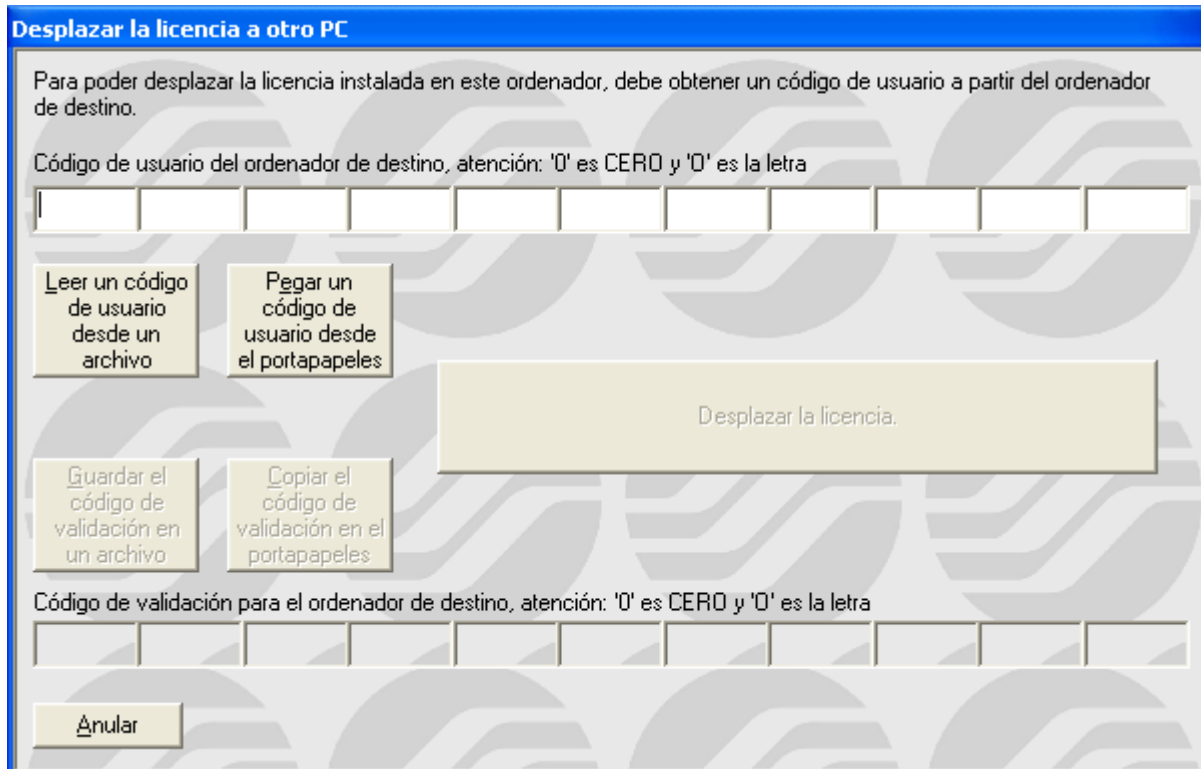
The license modification procedure is identical to registration.

## Moving a license from one computer to another

This procedure is more complex. The instructions below must be scrupulously followed to obtain good results. In the instructions below, « source » PC indicates the computer with the license and the « target » PC is the PC where the license needs to be moved.

- 1- if it has not already been done, install AUTOSIM on the target PC,



- 2- generate an « .a8u » user code file on the target PC and move this file to the source PC (for example on a floppy disk),
- 3- on the source PC, select the « Move the license to another place » option,

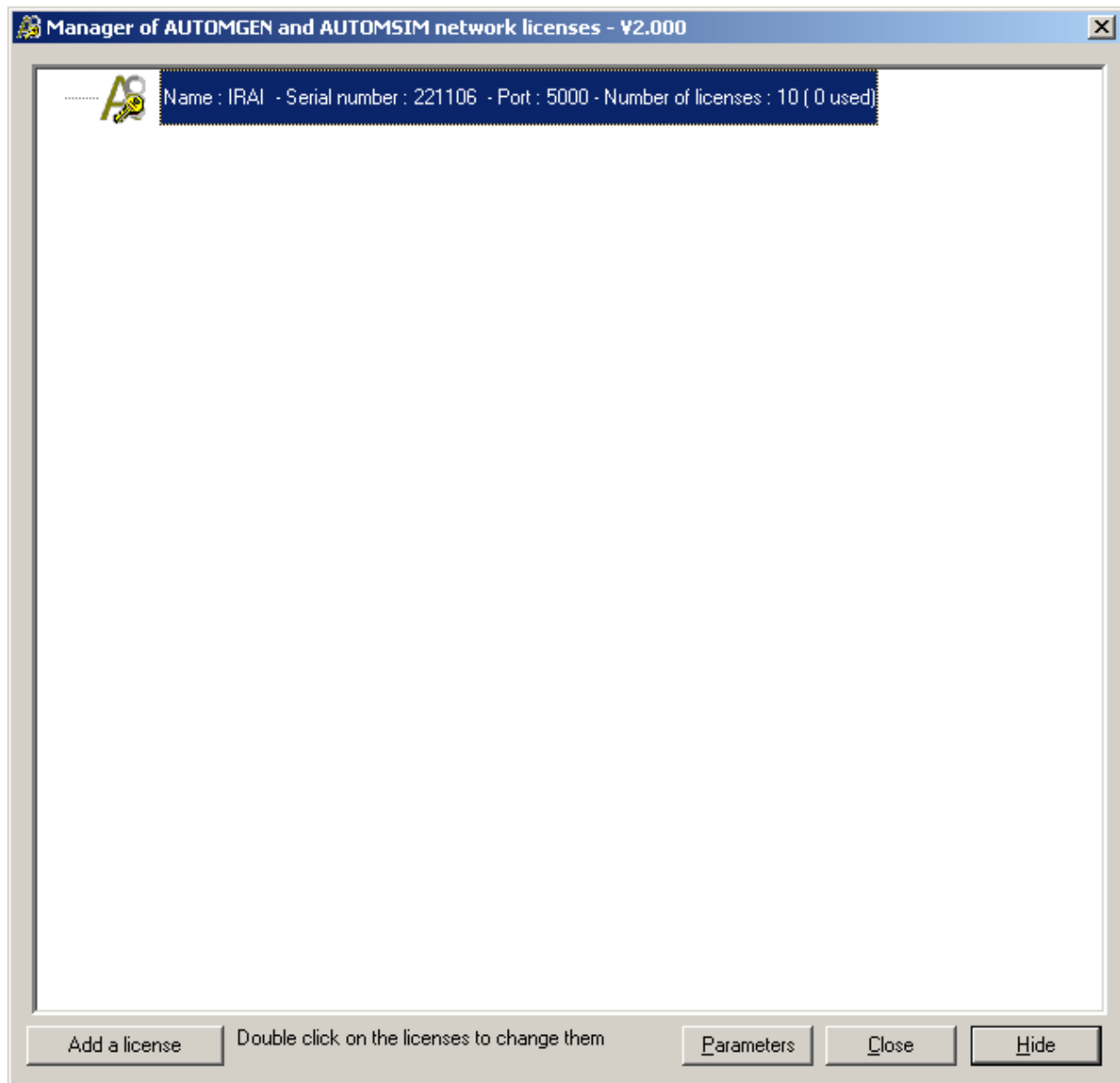


*Dialogue box for moving a license*

- 4- on the source PC, click on « Read a user code from a file » and select the « .a8u » file that came from the target PC,
- 5- on the source PC, click on « Move the license »,
- 6- on the source PC, click on « Save the validation code in a file », recopy the generated « .a8v » file to the target PC,
- 7- on the target PC, click on « Read a validation code from a file » and select the « .a8v » file that came from the source PC.

## Network licenses

The « akey8.exe » executable manages the network license. This executable must be launched from one of the network computers. The network must be able to be used with TCP IP protocol. When launched, the network license manager is hidden and only a  icon appears in the WINDOWS keybar. To display the network license manager window, double click on the  icon in the keybar.



*The network license manager*

Up to 16 different licenses can be managed by the network license manager. A network license is characterized by a number of users and a type of copyright (number of all or none inputs/outputs and useable post-processors). For each license the number of possible user/s, number of connected user/s and list of connected users (using AUTOSIM) is displayed in a tree format attached to each license. Each license is associated to a port number (a numeric value starting from 5000 by default). The first port number used can be configured by clicking on « Parameters ».

### Adding a network license

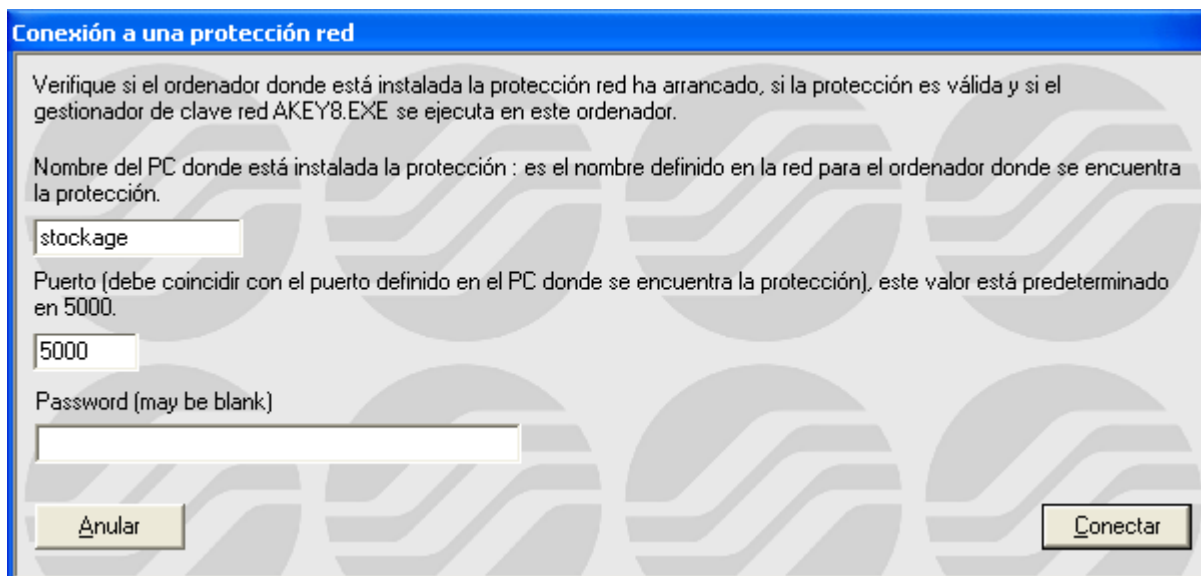
You can add a license by clicking on « Add a license ». The license registration principle is the same as for single license versions.

## Modifying a license

Double click on the licenses to modify them. The license modification procedure is the identical to that used for single license versions.

## Connecting to client stations

Click on « Connect to a network license » to connect a client station to a network license.



**Conexión a una protección red**

Verifique si el ordenador donde está instalada la protección red ha arrancado, si la protección es válida y si el gestor de clave red AKEY8.EXE se ejecuta en este ordenador.

Nombre del PC donde está instalada la protección : es el nombre definido en la red para el ordenador donde se encuentra la protección.

stockage

Puerto (debe coincidir con el puerto definido en el PC donde se encuentra la protección), este valor está predeterminado en 5000.

5000

Password (may be blank)

Anular Conectar

*Connecting to a network license*

The PC name (the one from the network) where the « akey7.exe » was launched must be provided as well as the port number corresponding to the desired license.

You must register your license with SMC ([training@smctraining.com](mailto:training@smctraining.com)) by sending your user code by e-mail (« File/License » menu. The network license manager is used to manage multiple licenses on TCP IP network PC's.

## Additional information on installing AUTOSIM in a network environment

### General information

Two aspects of the AUTOSIM<sup>8</sup> installation have to be considered: installing files on the one hand and managing licenses on the other. These two aspects are completely separate: you can choose to install the files either on the hard disk of the client PCs or else on a file server and, completely independently of this, choose to install either a license locally on a PC or else a network license on a network license manager.

### Installing AUTOSIM<sup>3</sup> on a file server

Benefit: the AUTOSIM<sup>3</sup> files are installed just once on a file server, and updates are simplified.

Procedure on the file server: install AUTOSIM<sup>3</sup>. Rights needed: read-access is sufficient.

Procedure on the client workstations: create a shortcut to the “autom8.exe” executable, which is in the AUTOSIM<sup>3</sup> installation directory on the file server.

### Installing one or more AUTOSIM<sup>3</sup> licenses on a network license manager

Benefit: the licenses are no longer restricted to one PC but can be used by all of the PCs connected to the network (floating licenses).

Principle: one or more licenses are installed on one of the network's PCs. A license authorizes from 1 to n users. AUTOSIM<sup>3</sup> may be launched on client PCs upto the maximum number of users. A license has the same features for all users in terms of the number of inputs/outputs that can be used and the types of post-processors that can be used. If several configurations (several types of licenses) are needed, then as many licenses will be created as there are different

types of configurations. When AUTOSIM<sup>3</sup> is launched on the client PCs, a connection will be created to one or other of the licenses depending on the features that are wanted.

Actual example: setting up a network of 4 16 I+16 O PL72 licenses, 4 16 I+16 O PL7 licenses + 2 unlimited I/O PL7 licenses. For this: 3 licenses will be created on the network license manager: 1 license for 4 16 I+16 O PL72 users, 1 license for 4 16 I+16 O PL7 users, 1 license for 2 unlimited I/O PL7 users.

Where to install the network license manager: on one of the network's PCs (it does not have to be the server) which must be running all the time (whenever a user would like to use AUTOSIM<sup>3</sup>).

Technical constraints: the network must support TCP/IP, the PC where the network license manager is located must be able to run a WINDOWS program (application or service).

Installation on the network license manager: on the PC where the network licenses are going to be managed, install the main AUTOSIM<sup>8</sup> module + the network license manager.

Registering one or more licenses on the network license manager: launch the network license manager: (AKEY8.EXE executable, located in the AUTOSIM<sup>8</sup> installation directory). When launched, the license manager sets up an icon in the bottom right of the WINDOWS task bar. Left-click once with the mouse to open the window.

Click on "Add a license" to add a license.

Click on "Save the user code in a file" to generate an .n8u file that you will e-mail to us at the address "[training@smctraining.com](mailto:training@smctraining.com)": we will send back an .n8v file that you will connect up by clicking on the "Read a validation code from a file" button.

The licenses installed in this way will then be shown in the network license manager with the serial number and characteristics of the license and the associated port number. It is this port number that will allow clients to connect to a specific license.

Installation on the client workstations: launch AUTOSIM<sup>3</sup>, and in the "File / License" menu select "Connect to a network license".



Enter the name of the PC where the network license manager is running (or its IP address) and the port number (this number makes it possible to identify the license you want to connect to, if there is more than one).

It is also possible to add an argument in the AUTOSIM<sup>3</sup> launch shortcut in order to force connection to one network license.

The argument is:

/NETLICENSE=<name of the PC where the network license manager is located>,<port>

Make sure that “NETLICENSE” is correctly spelled: S not C at the end.

For example:

/NETLICENSE=MYSERVER,5001

Several launch shortcuts can be created in order to connect to different licenses.

Possible problems: if you use a firewall, make sure access is authorized to the ports used by the network license manager (those displayed in the network license manager).

Installing the network license manager as a service under WINDOWS NT, 2000, XP, 2003 and VISTA.

Displaying the status of the licenses remotely: to display the status of the network license manager on a different PC from the one on which the network license manager has been launched (or if the “service” version of the network license manager is being used), use the “spya8protnet.exe” utility, which is located in the AUTOSIM<sup>3</sup> installation directory.

### Installing the network license server as a service

The “NT Service” key server allows the AUTOSIM<sup>3</sup> network licenses to be managed on a WINDOWS NT4, 2000, 2003, XP or VISTA workstation without opening a session. Unlike the AKEY8.EXE “executable” version, AKEY8NT.EXE does not allow either the protections or the connected users to be displayed.

Before installing the key server as an “NT service”, you are recommended to make sure that the key server works properly with the “executable” version: AKEY8.EXE.

Launch the “akey8nt -i” command line to install the NT key server service. The AKEY8NT.EXE executable is installed in the AUTOSIM installation directory.

So that the service starts automatically:

- under WINDOWS NT4: in the “Start/Parameters/Configuration Panel” menu, select the “Services” icon the “AKEY8” line, click on the start button and select the “Automatic” button.

Reboot your PC so that the key server is activated.

- under WINDOWS 2000, 2003, XP or VISTA: in the “Start/Parameters/Configuration Panel” menu, select the “Administrative Tools” icon then the “Services” icon. Right-click with the mouse on the “AKEY8” line and select “properties”. In the “Startup Type” option, select “Automatic”. In the “Recovery” tab, select “Restart the service” in the “First Failure” area.

## Uninstallation

Launch the “akey8nt -u” command to uninstall the NT key server service.

## Errors

After having uninstalled the AKEY8NT.EXE service, use AKEY8.EXE to determine the cause of any malfunctions.

## The project

AUTOSIM<sup>3</sup> is strongly based on the idea of a project. A project groups together the elements that compose an application. The browser displays all the project elements (sheets, symbols, configuration, IRIS objects etc.) in a tree format.

The new file format of AUTOSIM<sup>3</sup> (files with « .AGN » extension) includes all project elements.

When you save an « .AGN » file you are assured of saving all the elements of an application. You can easily and effectively interchange applications created with AUTOSIM.

« .AGN » files are compacted with « ZIP » technology, they do not need to be compressed to be interchanged, their size is already optimized.

All the files generated by AUTOSIM<sup>3</sup> can be re-read with all of the versions of AUTOSIM<sup>3</sup>: upward and downward compatibility.

## Files generated with AUTOSIM<sup>2</sup>

The files created with AUTOSIM<sup>7</sup> can be opened directly in AUTOSIM<sup>3</sup>.

## Importing an application from an earlier version of AUTOSIM (version 2 or earlier)

You need to import all of the sheets (".GR7" files) and any symbol file (".SYM" file). To do this, use the import procedures described in the following chapters.

## Generating a free distribution executable file

The « Generate an executable » command from the « File » menu is used to generate an executable starting from a project in progress (an « .EXE » file executable on a PC with WINDOWS). The AUTOSIM « viewer » is automatically integrated with the generated executable (the executable user does not need AUTOSIM). This viewer makes it possible to use the application without modifying it. You can easily distribute your applications. The generated executable is not covered by copyright. This

technique is normally used for producing a supervising application.

### Modifying project properties

With the right side of the mouse click on the « Project » element on the browser and select « Properties » from the menu.

### Modifying security options

You can restrict reading or modification access to a project by passwords.

### Advanced options

« Save the environment aspect with the project »: if checked, the position of the windows and the aspect of the toolbars are saved in the « .AGN » file. When the project is opened, these elements are reproduced.

« Hide the main window upon launching ... »: if checked, the AUTOSIM window is hidden when the project is opened. Only IRIS objects incorporated in the project will be displayed. This option is normally used to create « package » applications which only leave IRIS objects displayed. Use the [CTRL] + [F11] keys to redisplay the AUTOSIM window.

The other options are used to change the display of the AUTOSIM window when a project is opened.

### User interface

« Block IRIS object configuration »: if checked, a user cannot modify IRIS object configuration.

The other options are used to modify the behavior of the user interface.

### Model

« This project is a document model »: if checked, when opened all the options and the documents it contains act as a model for the creation of a new project. This functionality is used to create standard configuration which can be uploaded when AUTOSIM is launched (for example a default symbol file or a default processor configuration).

### Defining a mode

To define a mode that can be used when launching AUTOSIM (like the “Expert” and “Beginner” modes), save a project model in the “models” sub-directory of the AUTOSIM installation directory. An image can be

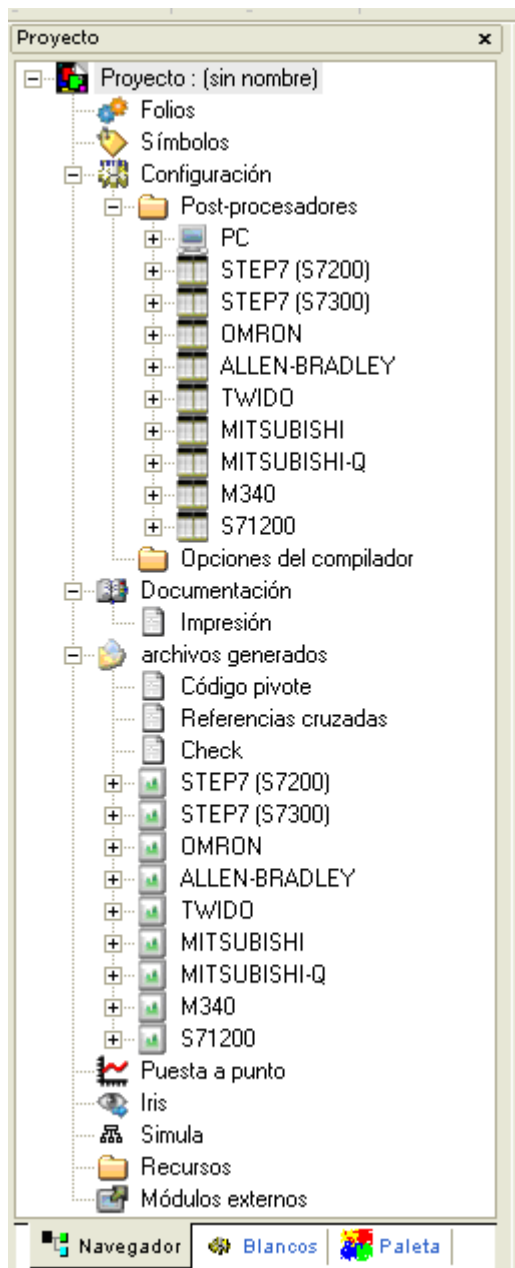
linked to a model. To do this, create a “jpg” format file with the same name as the “.agn” file. This file must have the following dimensions: 120 pixels wide by 90 pixels high.

### Automatic GO

«Automatic go at project launch »: if checked, the application is automatically run when a project is opened.

The project is used to group together the elements of an AUTOSIM application. Once regrouped, the elements form a compact file with « .AGN » extension. The project models are used to be able to easily manage different software configurations. Generation of executables makes it easy to distribute applications.

## The Browser



*Browser tree*

A central element for application management, the browser is used for fast access to different application elements: sheets, symbols, configuration, printing, IRIS objects etc.

The « + » and « - » icons are used to develop or retract project elements.

Actions on the browser are effected by double clicking on the elements (opens the element) or by clicking with the right side of the mouse (adds a new element to a project, special action on an element etc.).

Certain operations are effected by dragging and dropping the elements and moving them on the browser.

The colors (generally called up at the bottom of documents in the work space) are used to identify families of elements.

## Sheets

A sheet is a page where a program or part of a program is designed.

Using sheets is extremely simplified in AUTOSIM<sup>3</sup>. The sheet chaining orders needed in the previous versions are no longer used. For multiple sheets to be compiled together, they only need to be in the project.

The icons associated to the sheets are shown below:

- normal sheet,
- normal sheet (excluding compilation),
- sheet containing a macro-step expansion,
- sheet containing a function block program,
- sheet containing a key,
- sheet containing a key (excluding compilation),
- sheet containing an encapsulation,

Icons are marked with a cross indicating a closed sheet (not displayed in the work space). Double clicking on this type of icon opens (displays) the associated sheet.

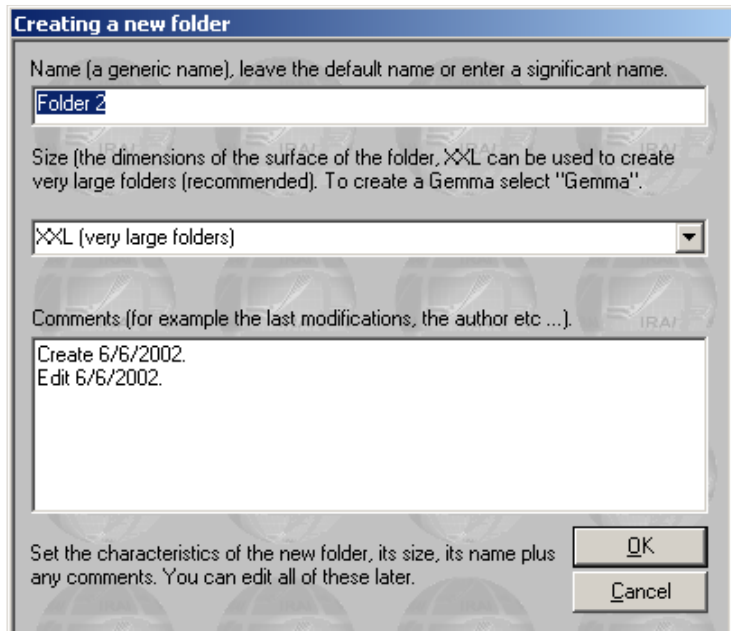
## Adding a new sheet

With the right side of the mouse click on the « Sheets » element on the browser then select « Add a new sheet ».

*Select the sheet size (XXL is the recommended format, the other formats are for older versions of AUTOSIM, GEMMA is only used for creating GEMMA models).*

*The sheet can be given any name, but each project sheet must have its own name.*

*The comment area is up to your discretion for modifications or other information relative to each sheet.*



**Creating a new folder**

Name (a generic name), leave the default name or enter a significant name.

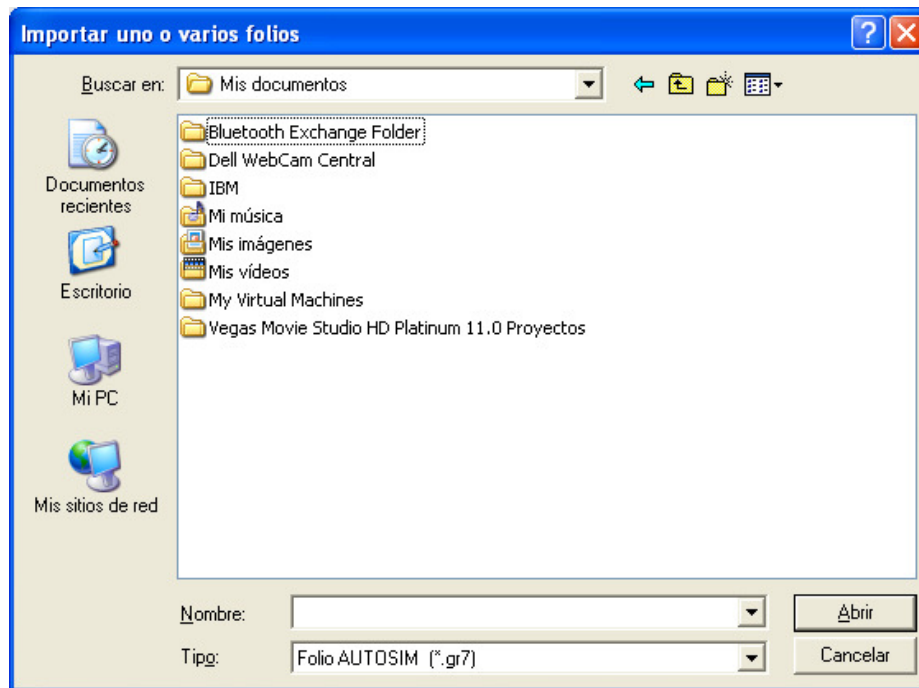
Size (the dimensions of the surface of the folder, XXL can be used to create very large folders (recommended). To create a Gemma select "Gemma".

Comments (for example the last modifications, the author etc ...).

Set the characteristics of the new folder, its size, its name plus any comments. You can edit all of these later.

## Importing old AUTOSIM version sheets, importing CADEPA sheets

With the right side of the mouse click on the « Sheets » element on the browser then select « Add one or more existing sheets ».



*Selecting one or more sheets to import.*

From the « Type » list select « AUTOSIM » or « CADEPA » for the sheet type to import then click on OK.

There are some restrictions for importing CADEPA sheets:

- the step numbers must be individual (the same step number cannot be used on multiple sheets),
- references must be converted with links to CADEPA before being able to import them.

By keeping the [CTRL] key pressed down, you can select multiple sheets.

## Modifying the sheet compilation order

The sheets are compiled in the order they are listed in for the project. To modify this order, click on the sheet with the left side of the mouse on the browser and move it in the list.



### Deleting a sheet from the list

With the right side of the mouse click the sheet to be deleted on the browser and select « Delete » from the menu.

### Exporting a sheet to a « .GR7 » file

With the right side of the mouse click the sheet to be deleted on the browser and select « Export » from the menu.

### Copying, Cutting, Pasting a sheet

With the right side of the mouse click the sheet on the browser and select « Copy/cut » from the menu. To paste, with the right side of the mouse click on the « Sheet » element on the browser and select « Paste ».

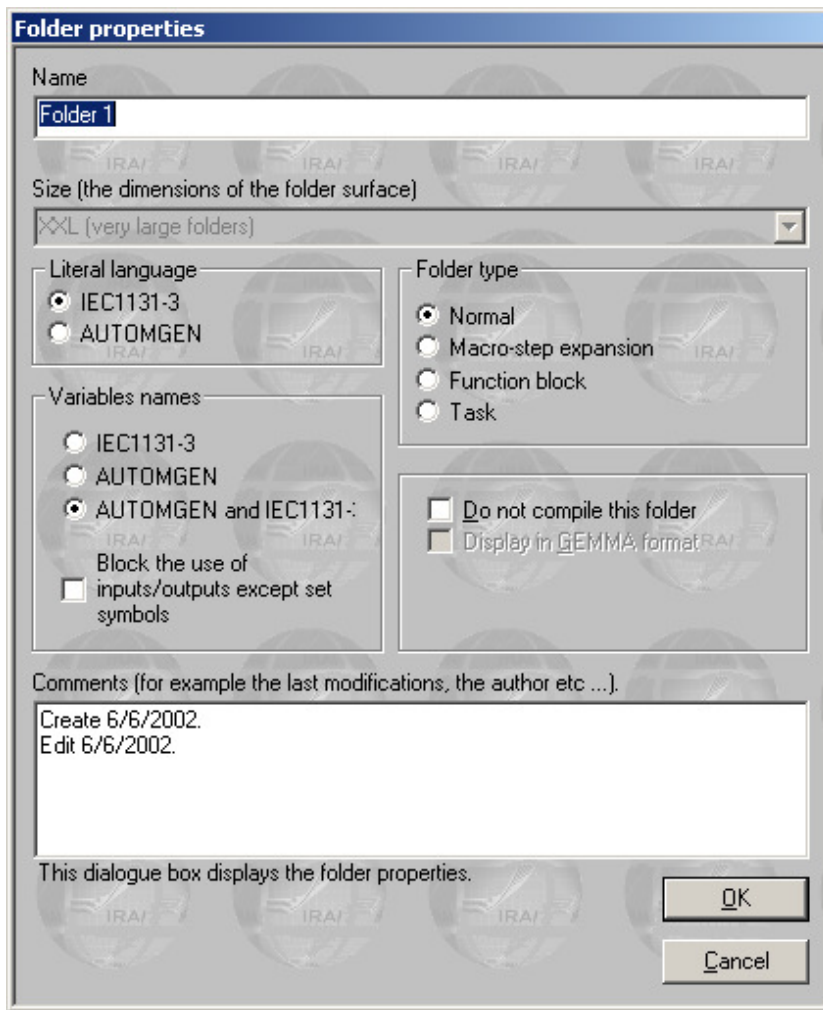
This option makes it possible to copy or transfer sheets from one project to another.

### Renaming a sheet

See « Modifying properties » below.

### Modifying sheet properties.

With the right side of the mouse click the sheet on the browser and select « Properties » from the menu.



You can modify the sheet name, the syntax used for literal language and variable names. The « Do not compile this sheet » option is used to exclude the sheet from the compilation. The « Display in GEMMA format » option is only available if the sheet format is GEMMA and is used to display and modify a sheet in GEMMA format. The « Block the use of inputs/outputs other than set symbols » option blocks the use of  $i$ ,  $\%i$ ,  $o$ ,  $\%q$  variables not attributed to symbols. Access to the sheet can be protected by a password. -The « comments » area is left to your discretion.

## Symbols

The list of symbols provides the correspondence between « symbol » names and variable names. A project may only have one symbol table.

### Creating a symbol table

With the right side of the mouse click on the « Symbols » element on the browser and select « Create a symbol table » from the menu.

### Importing a symbol table

With the right side of the mouse click on the « Symbols » element on the browser and select « Import a symbol table » from the menu.

## Configuration

### Post-processors

This section contains all the post-processor configuration elements (see the post-processor manual for more information).

### Compiler options

Double click on this element to modify the settings of compiler options.

## Documentation

This is used to access the file printing function (double click on the « Print » element. You can print a complete file composed of an end paper, cross reference table, symbol list and sheets. The print setup function is used to display all these elements.

## Generated files

### Generating the instruction list in pivot code

By double clicking on « Pivot code » you generate a list in low level literal language (AUTOSIM pivot code). Viewing of the generated code is normally reserved for specialists involved in understanding the translation methods used by the compiler.

### Generating the cross reference list

Double clicking on the « Cross reference » element generates and displays the list of variables used in an application with any associated processor variables and the name of or sheet(s) where they are used.

### Post-processors

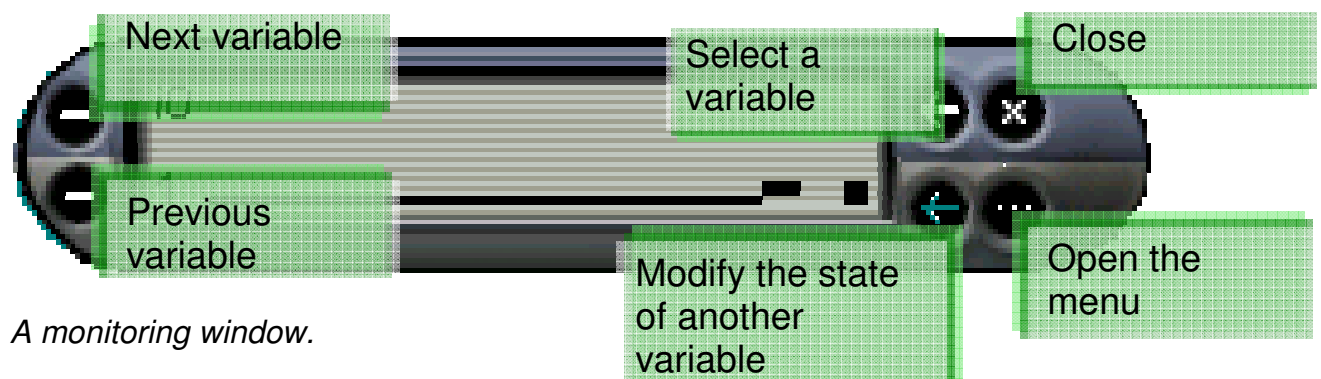
The other elements concern the files generated by the post-processors: instruction lists are in processor language.

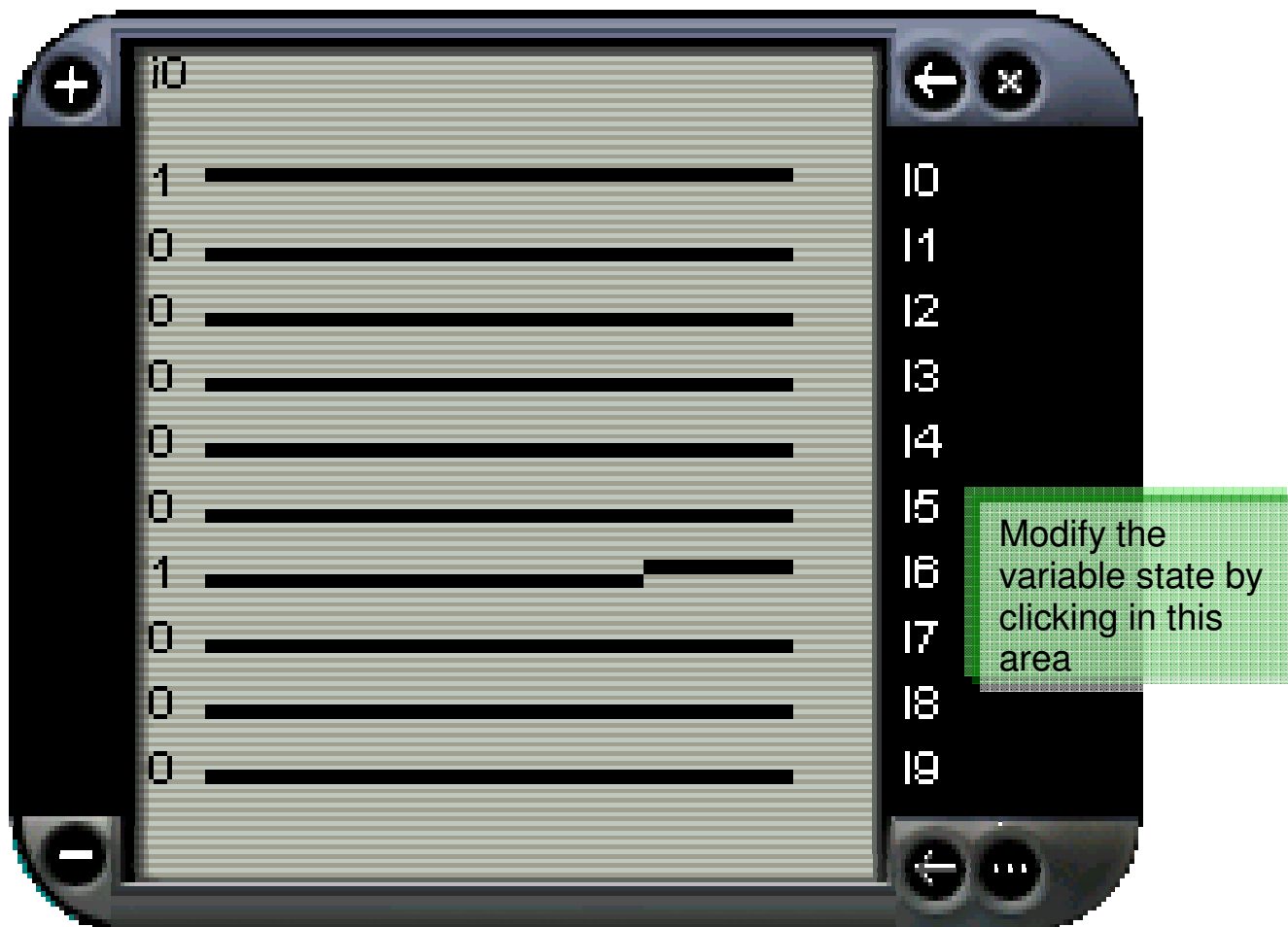
## Settings

Contains the tools to display and modify the state of the variables.

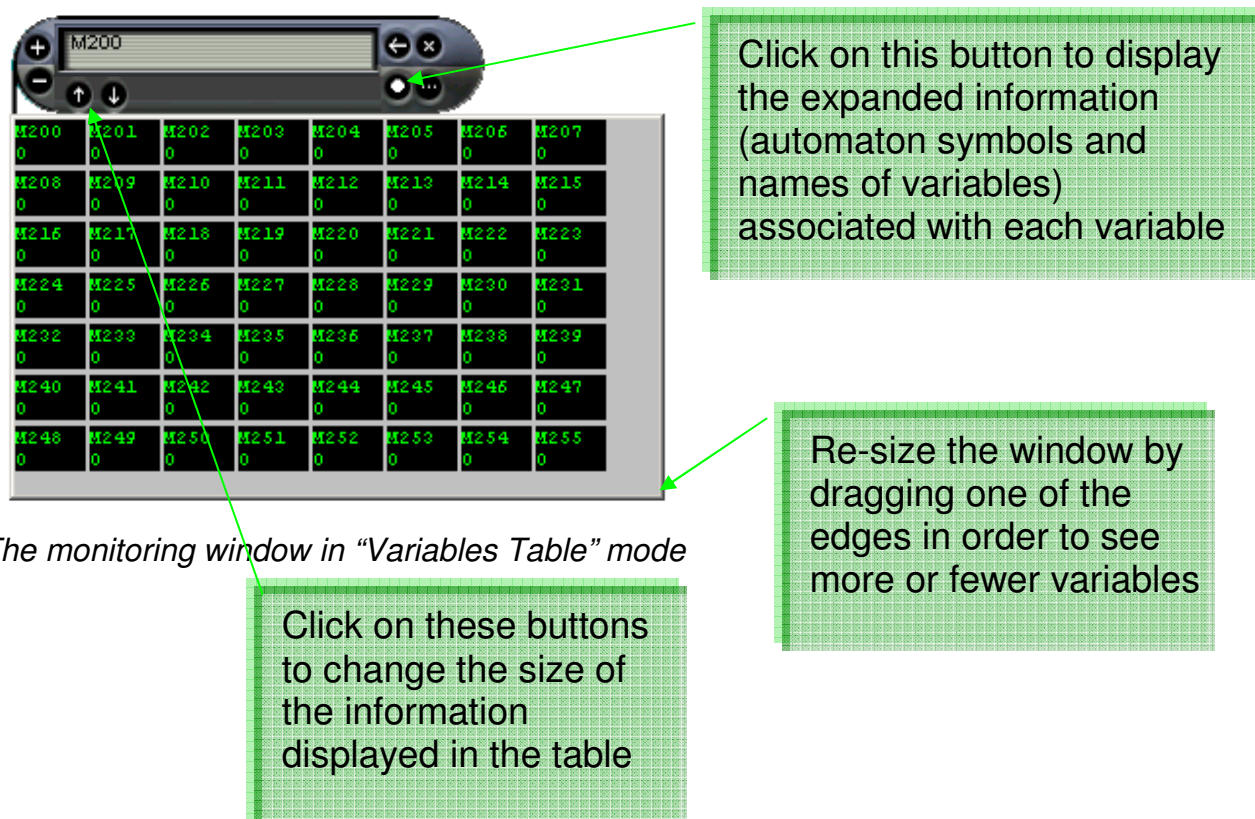
### Viewing and modifying a variable or variable table

With the right side of the mouse click on « Settings » and select « Monitoring » to open an element where you can see the state of a variable or variable table.





The monitoring window in « 10 variables table » mode.



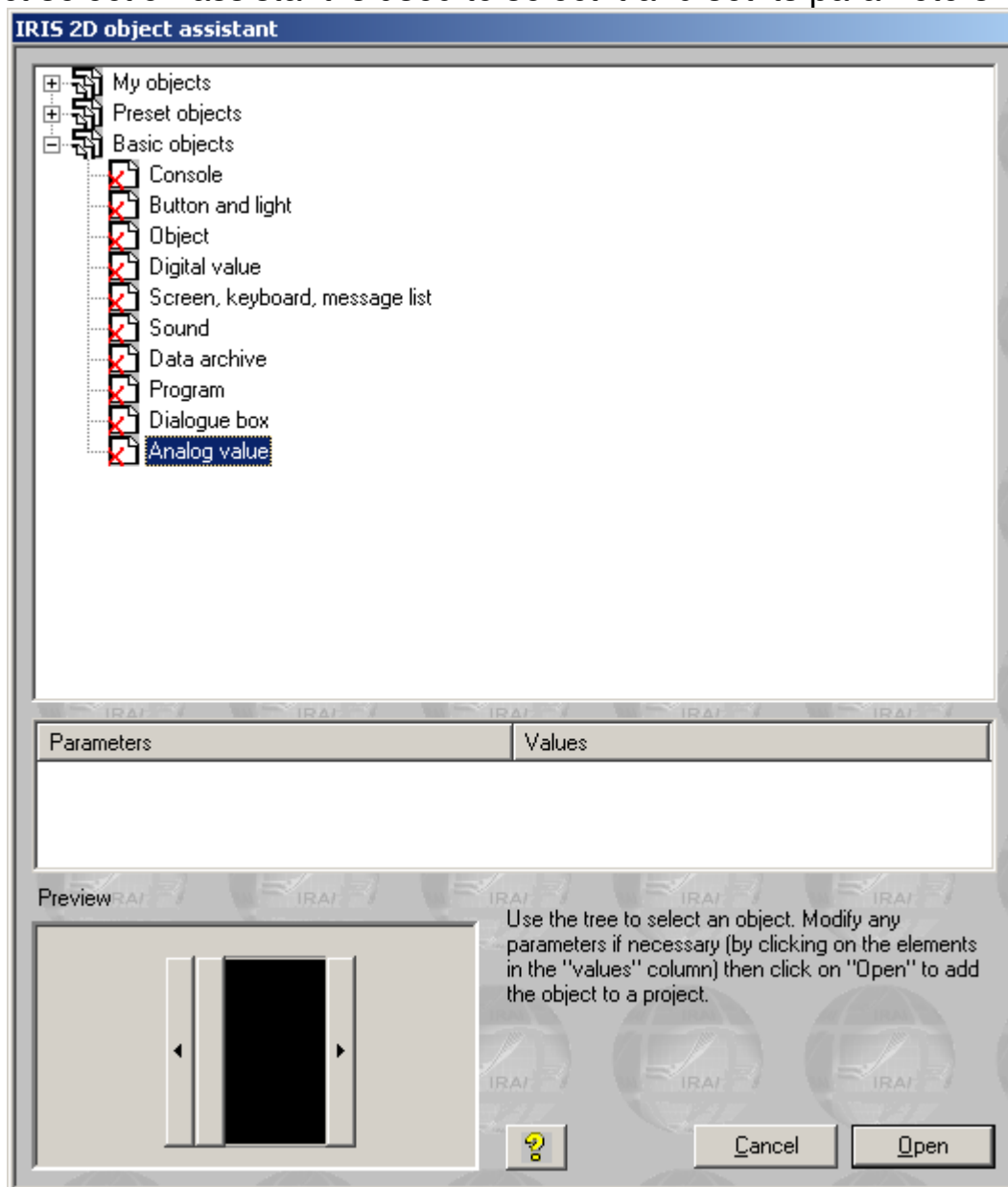
The monitoring window in "Variables Table" mode

## IRIS objects

IRIS 2D objects are used to create consoles, supervision applications and simulation applications of 2D operating parts. IRIS 3D is used to create simulation applications of 3D operating parts. Each IRIS 2D object appears in the project tree (see the chapters IRIS 2D references and IRIS 3D references for additional information).

### Adding an IRIS 2D object

Click with the right side of the mouse on « Add an IRIS 2D object ». The object selection assistant is used to select it and set its parameters.



Selection assistant for an IRIS 2D object

### Deleting an IRIS 2D object

With the right side of the mouse click on the IRIS object on the browser and select « Delete » from the menu.

### Displaying or hiding an IRIS 2D object

With the right side of the mouse click on the IRIS object on the browser and select « Display/hide » from the menu.

### Cutting, copying, pasting an IRIS 2D object

With the right side of the mouse click on the IRIS object on the browser and select « Copy » or « Cut » from the menu.

To paste, with the right side of the mouse click on the « Sheet » element on the browser and select « Paste ».

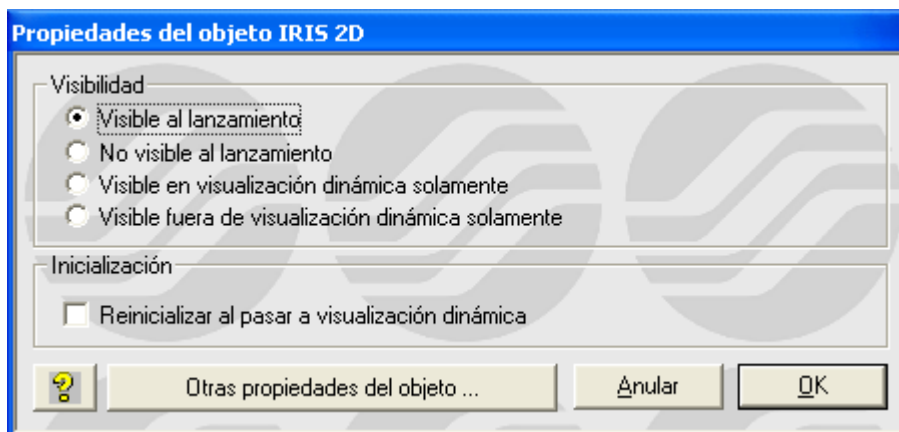
To paste an IRIS object on a console, select « Paste» from the console menu or click with the right side of the mouse on the console on the browser and select « Paste».

### Adding a new IRIS 2D object on a console

Select « Add an object » from the console menu or click with the right side of the mouse on the console on the browser and select « Add an object on the console » from the menu (for more information on the console see the chapter « Console » object)

### Modifying the properties of an IRIS 2D object

With the right side of the mouse click on the IRIS object on the browser and select « Properties ». For higher level objects (parent objects), special properties can be accessed:

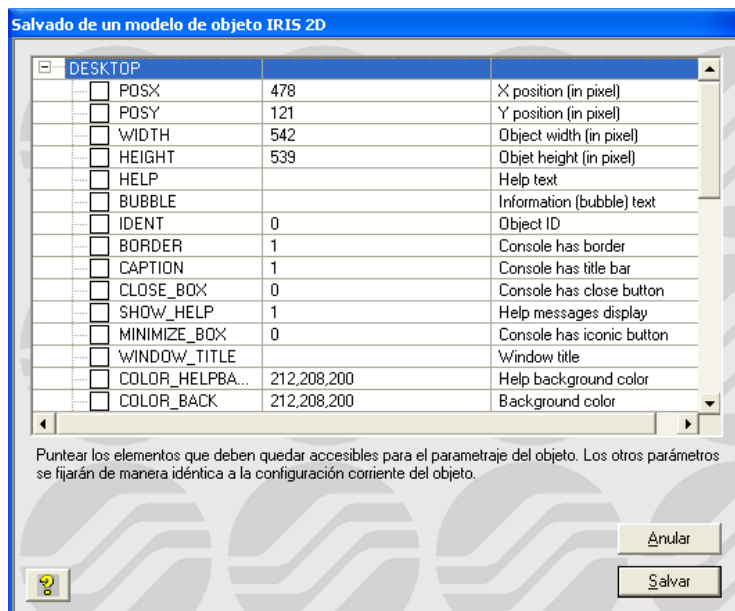


*Properties of high level objects*

Display establishes under which conditions the object is displayed or hidden. The reinstallation option is used to return an object to its initial state when dynamic display is launched (normally used for OP simulation applications).

### Setting an object model accessible on the assistant

With the right side of the mouse click on the IRIS object on the browser and select « Save as model » from the menu.



*Selection of modifiable parameters for users of your models*

You can select the list of parameters which remain accessible to the user on the assistant. By clicking on « Save », you save your object model. The storage directory for object models is « <AUTOSIM installation directory>\i2d\lib ». You can use a sub-directory called « my objects » for saving your models.



### Importing an IRIS 2D object in an earlier version of AUTOSIM

With the right side of the mouse click on the « IRIS » element on the browser and select « Import IRIS 2D objects ». Select one or more « .AOF » files.

### Creating an IRIS 3D console

With the right side of the mouse click on the « IRIS » element on the browser and select « Add an IRIS 3D console » (see the chapter on IRIS 3D for more information).

## Resources

This project element is used for adding all types of files to a project. Files which are added will become an integral part of the project and will be saved along with the other elements. To refer to a pseudo directory where the resources are, the key word « <RESDIR> » can be used in the specific directory name in AUTOSIM. For example IRIS objects can refer to bitmaps if they are included in the resources.

### Adding a file to the resources

With the right side of the mouse click on the « Resources » element on the browser and select « Add » from the menu.

### Deleting a file from the resources

With the right side of the mouse click the resource file on the browser and select « Delete ».

### Renaming a file in the resources

With the right side of the mouse click the resource file on the browser and select « Rename ».

### Modifying a file in the resources

With the right side of the mouse click the resource file on the browser and select « Modify ».

### Adding and converting 3D STUDIO files in the resources

3D STUDIO files can be converted into .x files and added to the resources by clicking with the right side of the mouse on the « Resources » element on the browser and selecting « Import 3D files » (see the chapter IRIS 2D references and IRIS 3D references for more information).

## External modules

These elements are reserved for executable modules developed by third parties and interfaced with AUTOSIM.

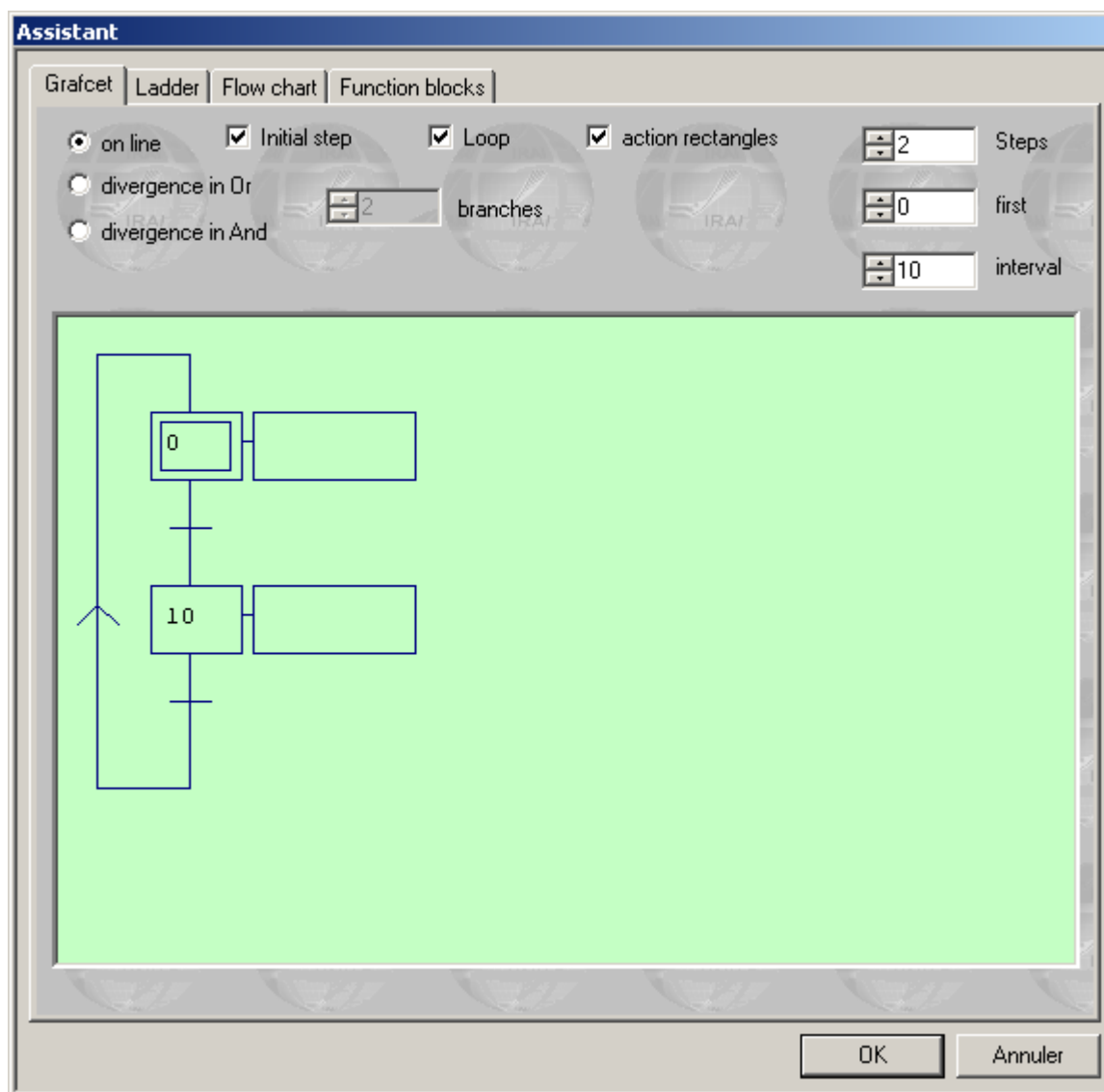
The browser is used to display and manage all the project elements. By double clicking on the elements or by clicking with the right side of the mouse, you access the different functions applicable to each element.

## Designing programs

Various tools are available for designing programs.

### Designing with the assistant

This is without doubt the simplest when starting with AUTOSIM. With the right side of the mouse click on an open sheet in the work space and select « Assistant » from the menu. You will then be guided for making selections. When you have finished click on « OK » and put the design on the sheet by clicking with the left side of the mouse.



*The assistant*

## Designing with the shortcut menu

Click with the right side of the mouse on an open sheet in the work space, the menu will propose a series of elements that you can put on the sheet. This is an instinctive and fast creation method.

## Designing with the pallet

By selecting elements on the pallet you can quickly create programs starting from previously created elements.

## Enhancing and customizing the pallet

« .GR7 » files are used to set the pallet, they are located in the directory « <AUTOSIM installation directory>\pal ». You can delete, modify, rename or add files. To generate « .GR7 », files use the « Export » command by clicking with the right side of the mouse on a sheet on the browser. The names displayed on the pallet are « .GR7 » files. Relaunch AUTOSIM for a new element to be displayed on the pallet.

## Designing with the keyboard keys






Each key is associated to design blocks. The « Blocks » element also provides access to the blocks. The table below lists the blocks and their use.

### Delete block











Aspect	Associated key	Generic name	Comments	Languages
	[A]	Delete	Used to make a cell blank again	All













### Link blocks



Aspect	Associated key	Generic name	Comments	Languages
	[E]	Vertical link	Link from top to bottom or bottom to top	All
—	[F]	Horizontal link	Link from right to left or left to right	All

	[G]	Upper left corner	Link towards the bottom right or bottom left	All
	[H]	Upper right corner	Link towards the bottom left or bottom right	All
	[I]	Lower left corner	Link from top to right or left to top	All
	[J]	Lower right corner	Link from top to left or right to top	All
	[Z]	Cross	Crosses two links	All







### Grafcet blocks

Aspect	Associated key	Generic name	Comments	Languages
	[B]	Step	Normal step	Grafcet
	[C]	Initial step without activation	Initial step without activation	Grafcet
	[D]	Initial step	Initial step	Grafcet
		Macro-step	Only available in the shortcut menu	Grafcet
	[+]	Encapsulating step	An encapsulation must be linked	Grafcet
	[-]	Initial encapsulating step	An encapsulation must be linked	Grafcet
		Initial state mark	Définie initial state for an encapsulation	Grafcet
	[T]	Transition	Transition	Grafcet
	[\$]	Source transition	Can replace the transition symbol	Grafcet
	[£]	Exit transition	Can replace the transition symbol	Grafcet







		<i>Link for action on transition crossing</i>	<i>Use the following element to design the action rectangle</i>	Grafcet
		<i>Start of an action rectangle on transition crossing</i>	<i>Use the [X] and [Y] elements to end the rectangle</i>	Grafcet
	[K]	Left limit of an « And » divergence	Compulsory to the left of an « And » divergences	Grafcet
	[L]	Supplementary branch of an « And » divergence or an « And » convergence	Do not use as a left or right limit of an « And » divergence	Grafcet
	[M]	Right limit of an « And » divergence	Compulsory to the right of an « And » divergence	Grafcet
	[N]	Extension of an « And » divergence	If placed in the [K], [L], [M], [P] or [O],[P],[Q], [L] blocks	Grafcet
	[O]	Left limit of an « And » convergence	Compulsory to the left of an « And » convergence	Grafcet
	[P]	Supplementary branch of an « And » convergence or an « And » divergence	Do not use as a left or right limit of an « And » convergence	Grafcet
	[Q]	Right limit of an « And » convergence	Compulsory to the right of an « And » convergence	Grafcet
	[R]	« Or » divergence	Do not use as a limit of an « Or » convergence	Grafcet
	[S]	« Or » convergence	Do not use as a limit of an « Or » divergence	Grafcet
	[U]	Skip or repeat left step	« Or » convergence or divergence	Grafcet

	[V]	Skip or repeat right step	« Or » convergence or divergence	Grafcet
	[SPACE] on an [E] block	Link towards the top	For relooping and repeating steps	Grafcet












### Flowchart blocks

Aspect	Associated key	Generic name	Comments	Languages
	[0] (zero)	Flowchart assignment	Separates the « test » from the « action » area	Flowchart
	[1]	« Not » function	Complements the block input signal	Flowchart
	[2]	« And » function	Combines the inputs in an « And » logic	Flowchart
	[3]	« Or » function	Combines the inputs in an « Or » logic	Flowchart
	[4]	Block environment	Enlarges an « And » or « Or » function block	Flowchart
	[5]	Bottom of block	Ends an « And » or « Or » function block	Flowchart

### Ladder blocks



Aspect	Associated key	Generic name	Comments	Languages
	[C]	Start left coil	Starts an action	Ladder
	[D]	Start right coil	Ends an action	Ladder
	[U]	Left limit	Ends the diagram	Ladder
	[V]	Right limit	Starts the diagram	Ladder
	[R]	Connection	« Or » function	Ladder
	[S]	Connection	« Or » function	Ladder

## Action blocks



Aspect	Associated key	Generic name	Comments	Languages
	[W]	Action rectangle left limit	Starts an action	Grafcet and Flowchart
	[X]	Action rectangle environment	Extends an action	Grafcet and Flowchart
	[Y]	Action rectangle right limit	Ends an action	Grafcet and Flowchart
	[.]	Left side of a double action rectangle	Starts a double action rectangle	Grafcet and Flowchart
	[/]	Middle of a double action rectangle	Prolongs a double action rectangle	Grafcet and Flowchart
	[%]	Right side of a double action rectangle	Ends a double action rectangle	Grafcet and Flowchart
	[S]	Divergence Action	Used to vertically juxtapose action rectangles	Grafcet and Flowchart
	[V]	Divergence Action	Used to vertically juxtapose action rectangles	Grafcet and Flowchart
	[#]	Action on activation	Defines the type of action	Grafcet
	[⌋]	Action on deactivation	Defines the type of action	Grafcet
	[@]	Event-driven action	Defines the type of action	Grafcet









## Test blocks

Aspect	Associated key	Generic name	Comments	Languages
	[7]	Left limit of a test	Starts a test	Flowchart and ladder
	[6]	Right limit of a test	Ends a test	Flowchart and ladder


## Organization chart blocks

Aspect	Associated key	Generic name	Comments	Languages
	[<]	Organization chart input	Indicates the input in a rectangle	Organization chart
	[=]	« False » output	Output if a test rectangle is false	Organization chart

## Function block blocks

Aspect	Associated key	Generic name	Comments	Languages
	[8]	Upper left corner of a function block	Starts the name of the function block	Function block
	[9]	Upper right corner of a function block	Ends the name of the function block	Function block
	[:]	Lower left corner of a function block	Adds an input to the function block	Function block
	[;]	Left limit of a function block	Adds an input to the function block	Function block
	[>]	Right limit of a function block	Adds an output to the function block	Function block
	[?]	Lower right corner of a function block	Adds an output to the function block	Function block

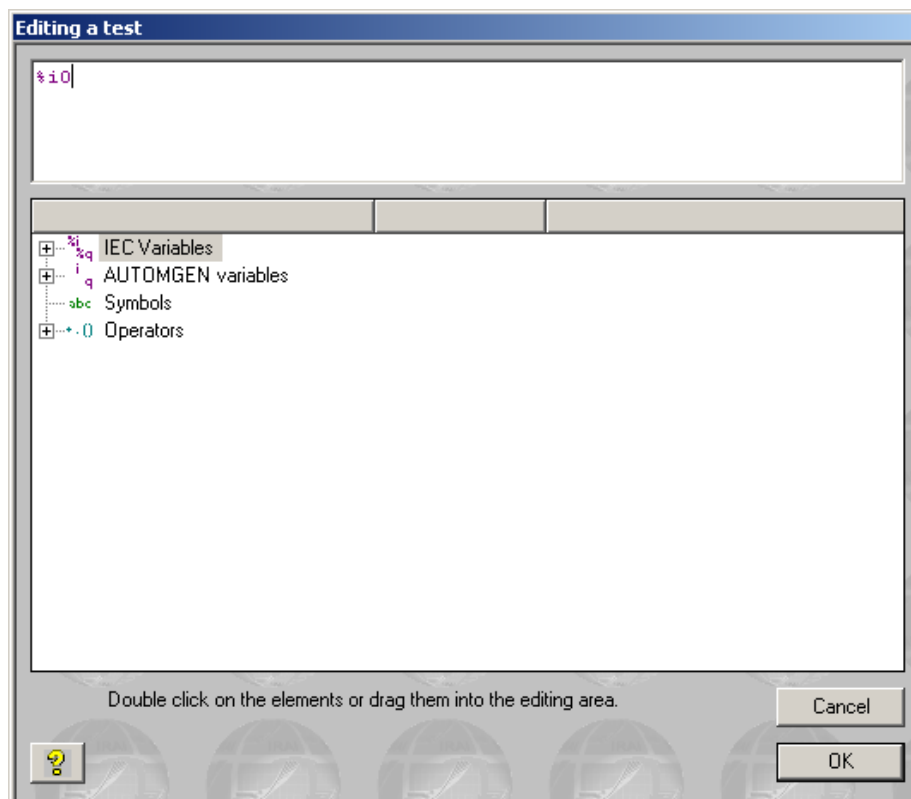
## Other blocks

Aspect	Associated key	Generic name	Comments	Languages
	[*]	Combination / transition link	This block is a link between the Logical Diagrams or Ladder languages and the Grafcet language	Grafcet / Flowchart / Ladder

## Documenting program elements


To document program elements, click below with the left side of the mouse. To create comments, click on a blank space on the sheet. To validate modifications, push the [Enter] key or click outside the editing area with the left side of the mouse. To delete modifications, push the [Esc] key or click outside the editing area with the right side of the mouse.

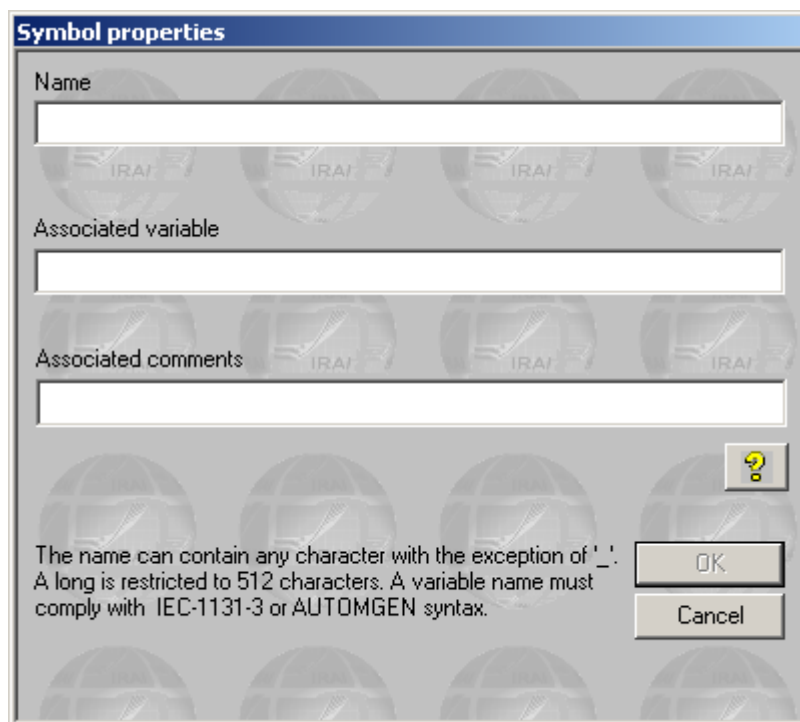
When editing tests and actions, a « ... » button appears under the editing area. If you click on it you access an assistant for creating tests or actions.



*Test creation assistant*

## Adding symbols

To create a symbol, click with the right side of the mouse on the symbol table in the work space and select « Add ». Or click the  button on the toolbar. You can also launch program compiling containing unset symbols. You will be asked for variables corresponding to the symbols during the compilation.




**Symbol properties**

Name

Associated variable

Associated comments



The name can contain any character with the exception of '\_'.  
 A long is restricted to 512 characters. A variable name must  
 comply with IEC-1131-3 or AUTOMGEN syntax.


OK  
 Cancel

*Attribution of symbols during compilation*

To easily design a program, create a new sheet, then click with the right side of the mouse on the bottom of the sheet. Select « Assistant » from the menu, you will then be guided by it.


## Running an application

### To run an application easily

The  button on the toolbar is the quickest way to see application run results. This pushbutton activates the following mechanisms:

- compilation of the application if it is not updated (not already compiled after the last modifications),
- installation of the run module (with downloading if the current target is a processor and following the connection options),
- passage of the target to RUN,
- activation of the dynamic display.

### To end the run

Click on . On the processor target, the program continues to be run on the target. On the PC, the program is stopped.

### To compile only

Click on .

### To stop the compilation

Click on .


### To connect to a processor or install a PC

Click on .

### To disconnect a processor or uninstall a PC

Click on .

### To put the target in RUN mode

Click on .

### To put the target in STOP mode

Click on .

### To initialize the target

Click on .

To run a program cycle on the target (generally not supported on processors)

Click on  .

To activate the dynamic display

Click on  .

To run an application, click on the « GO » button. To end the run, click again on the same button.

## The compiler

The compiler translates the sheets into a set of pivot language equations (these can be displayed by double clicking on the « Generated code / pivot language » element on the browser).

The pivot language is then translated into a language which can be run by a post-processor (the current post-processor can be displayed and selected by double clicking on the « Targets » panel accessible by clicking on the « Targets » tab at the lower part of the window where the browser is).

## Modifying compiler options

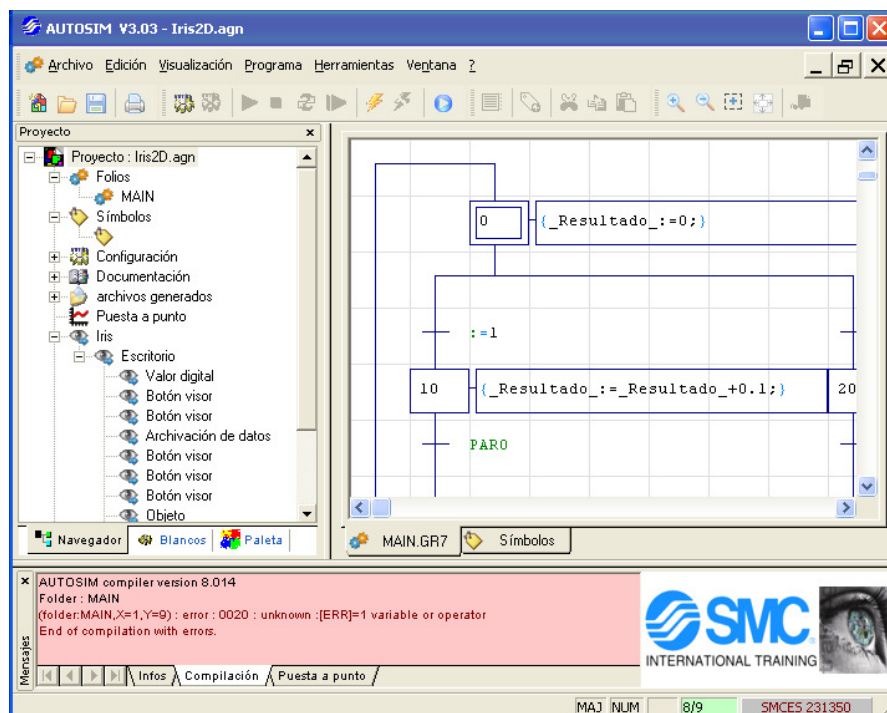
Double click on the element « Configuration / Compiler options».

## Displaying compilation messages

The « Compilation » panel on the messages window contains the counts produced by the last compilation.

## Finding an error

By double clicking on error messages, you can find the source.



*An error message and its source*

If the message windows are hidden and if one or more errors are detected by the compiler, a dialogue box indicates the first error detected (to display the message windows: use the « Messages » command from the « Windows » menu).

At the end of the compilation the « Compilation » window provides a list of any errors. By double clicking on the error messages, the site in the program that caused the error is displayed.

## Running programs on a PC

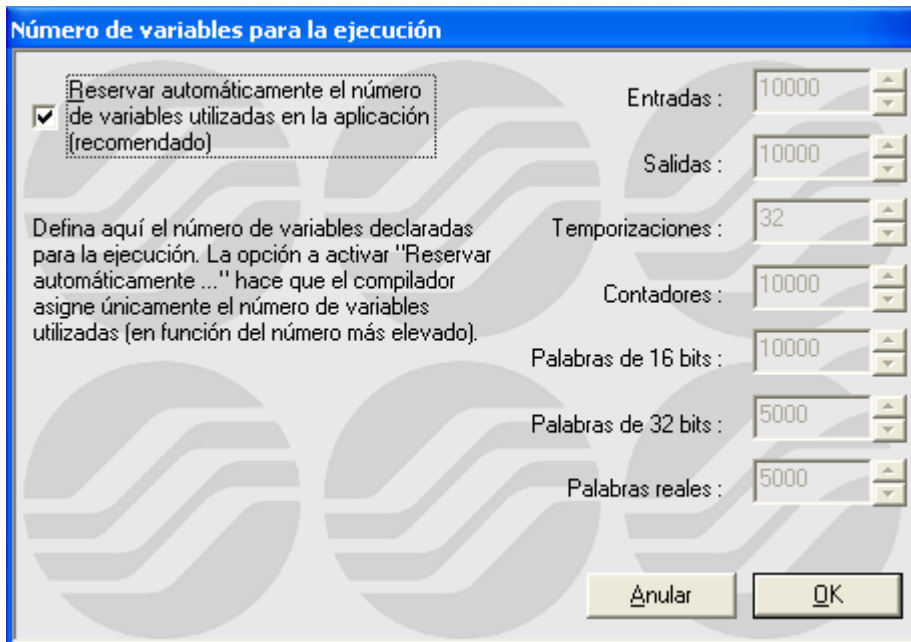
The « run PC» target is an actual processor loaded in your PC.

You can:

- test your applications,
- drive a virtual operating part created with IRIS 2D or 3D,
- drive input/output cards connected to the PC.

## Configuring the number of variables

Double click on the « Configuration / Post-processors / Executor PC / Variables » element.



*Selecting the number of variables*

The space needed for the variables used in the application is automatically reserved by default. You can manually select the amount of memory to reserve for each type of variable. This may be necessary if an indexed addressing is used to access a variable table.



## PC system variables

Bits 0 to 99 and words 0 to 199 are system variables and can not be used as user variables in your applications. The two tables below provide details on the PC system variables.

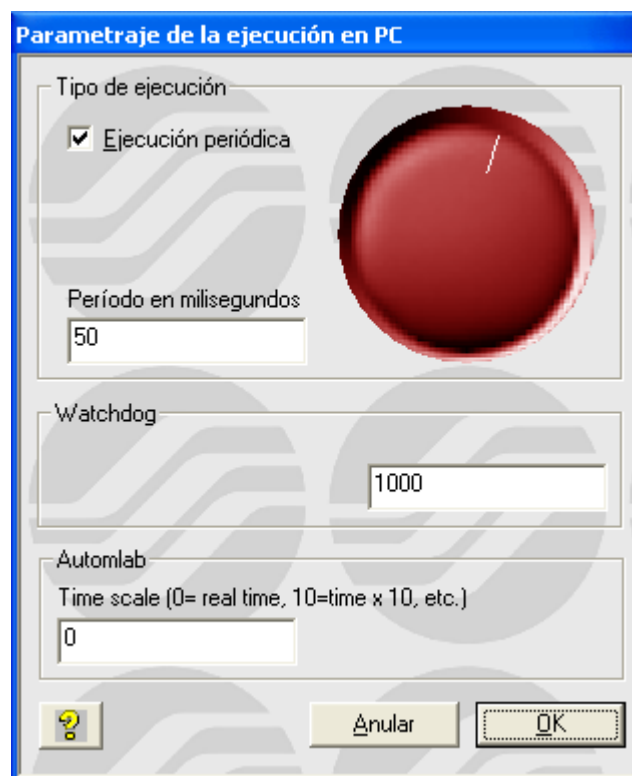
Bits	Use
0	active at first cycle, activation of initial Grafset steps
1 to 4	reserved for I/O drivers
5 to 7	reserved for I/O driver errors
8	error on watchdog overflow is equal to 1
9 and 10	error general PC fault
11	run mode 1=RUN, 0=STOP
12	emergency stop pass to 1 in the event of an error or set to 1 to stop the program
13 to 29	reserved for drivers
30	bit associated to timer 1
31	bit associated to timer 2
32	bit associated to timer 3
33	bit associated to timer 4
34	bit associated to timer 5
35	bit for repeating sector (pass to 1 on repeat sector, reset to zero is the job of the programmer)
36	setting this bit to 1 causes reading of the clock in real time and transfer to System words 4, 5, 6, 7, 8, 51 and 52.
37	setting this bit to 1 causes writing of System words 4, 5, 6, 7, 8, 51 and 52 in the real time clock.
38 to 55	reserved
56	division by zero
57 to 67	reserved for future versions
68 to 99	reserved for the stack of boolean processing

Words	Use
0	reserved for the upper part of the multiplication result or the remainder of the division
1 to 3	timers in milliseconds
4	timer in 1/10 second
5	timer in seconds
6	timer in minutes
7	timer in hours
8	timer in days
9 to 29	reserved for I/O drivers
30	timer 1 counter

31	timer 2 counter
32	timer 3 counter
33	timer 4 counter
34	timer 5 counter
35	timer 1 procedure
36	timer 2 procedure
37	timer 3 procedure
38	timer 4 procedure
39	timer 5 procedure
40	lower part of clock reference
41	upper part of clock reference
42 to	reserved for I/O drivers
50	
51	timer in months
52	timer in years

## Modifying the run period

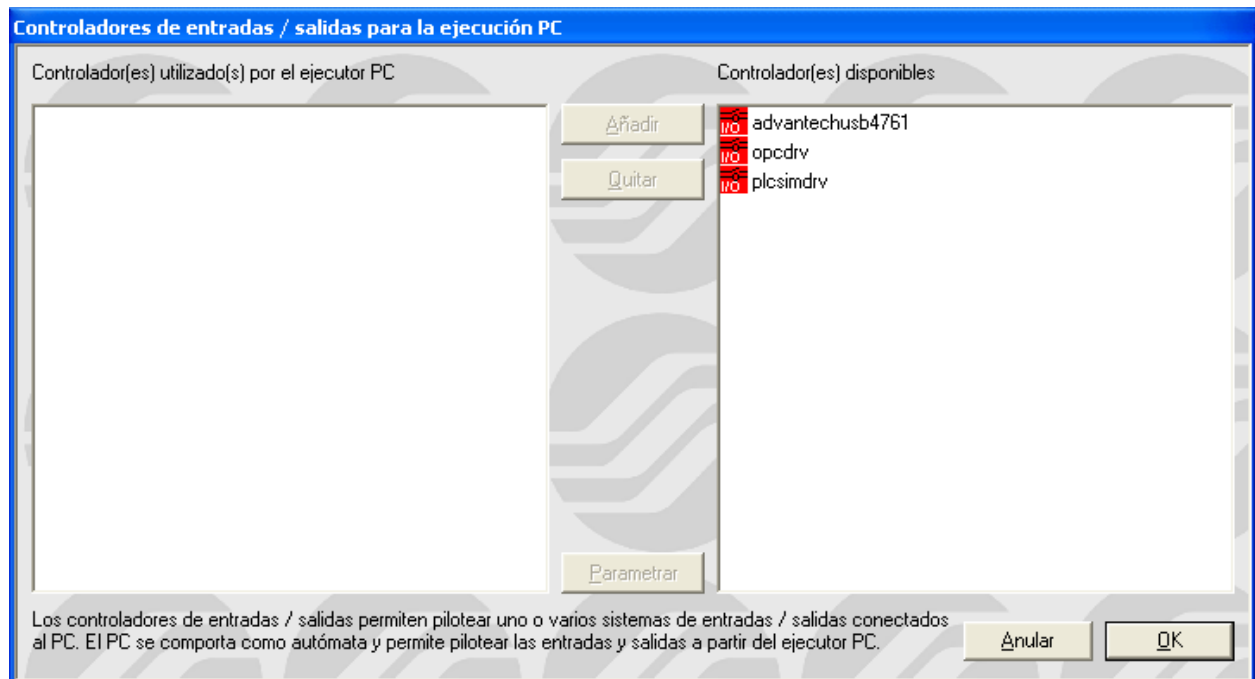
Double click on « Post-processors / Executor PC / Run ».



*Setting the run period*

## Driving inputs/outputs

Double click on « Configuration / Post-processor / Executor PC / I/O Drivers ».



*Adding an I/O driver*

Select a driver from the list on the right and then click on « Add ».

« Set parameters » is used to configure certain drivers.

The executor PC transforms your PC into a program processor, it can be used to drive inputs/outputs directly connected to your computer.

## IRIS 2D references

IRIS 2D objects are used to create supervision and simulation applications of 2D operating parts.

The link between the objects and the automatically functioning applications is always created by interchanging the variable state.

IRIS 2D objects are contained in WINDOWS windows.



*An IRIS 2D object*

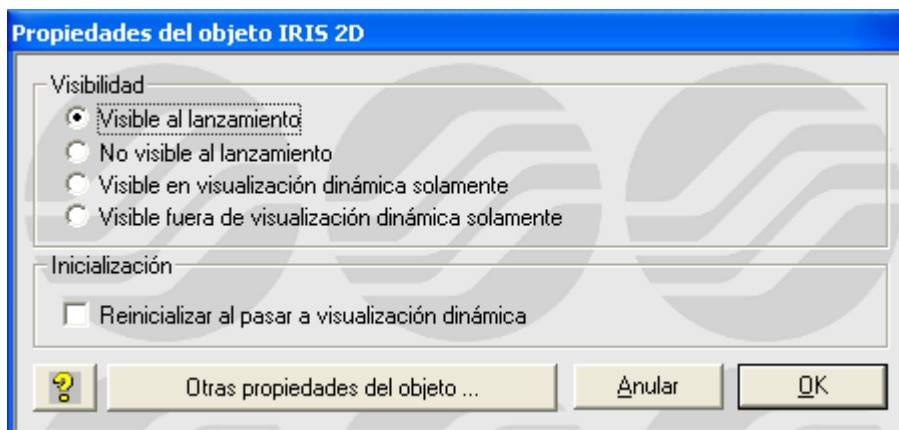
IRIS 2D objects have two possible states: the « Configuration » mode (used to modify the object characteristics) and « Use » mode (for using an object). The « User » mode is also called « Employ » mode.

### Modifying object display

The objects can be hidden or displayed. This property can be specified when opening an object or when changing the state of the dynamic display in the environment. Only higher level objects (not objects located on a console) can be displayed or hidden. Objects located on a console are displayed or hidden at the same time as the console.

To dynamically modify the visibility of objects, click with the left side of the mouse on the objects on the browser and select « Display/Hide ».


To modify the display properties, click with the left side of the mouse on the objects on the browser and select « Properties ».



*Display properties of an object.*

## Modifying object characteristics

### Removing an object

Method 1: click the  button on the surface of the object.

Method 2: with the right side of the mouse click on the object on the browser and select « Delete » from the menu.


### Dimensioning an object

By dragging the object from one of its edges you can enlarge or shrink it (you can also precisely modify the size of an object by accessing its properties, see below).

### Moving an object

Drag the object by clicking with the left side of the mouse on the minibar located on the upper part of its surface.

### Putting an object in « User » mode

Method 1: click on the button  on the object with the left side of the mouse.

Method 2: click with the right side of the mouse on the object.

### Putting an object in « Configuration » mode

Click with the right side of the mouse on the object.

### Modifying the characteristics of an object

Method 1: click on the  button.

Method 2: push down the [CTRL] key on the keyboard and click with the right side of the mouse on the object, then release the [CTRL] key.

Method 3: with the right side of the mouse click on the object on the browser and select « Properties » from the menu.

### Block access to configuration for all objects


With the right side of the mouse click on « Project » on the browser, select « Properties » and check « Block IRIS 2D object configuration » on the « Advanced » tab.

### Basic objects, preset objects

The basic objects set major functionality types. Preset objects are based on a basic type and a configuration to meet a specific need. For an example, an emergency pushbutton is an object derived from a basic object used to create pushbuttons and lights. To access preset objects, use the assistant by clicking with the right side of the mouse on the « IRIS » element on the browser and select « Adding an IRIS 2D object ».

### List of basic objects

#### « Console » object

The console object is the only object which can contain other objects on its surface. It is used to create command consoles and animation surfaces for virtual operating parts. This object has a pushbutton  used to manage objects on its surface: add, move, delete etc.

#### The « Button and light » object

This is used to create pushbuttons and lights that interact with the processing application variables.

#### The « Object » object

This is a polymorphic element primarily used to simulate operating parts.

#### The « Digital value » object

This is used to display numeric values of the processing application in a number format.

#### The « Screen, keyboard, message list » object

This is used to display information on the processing application in a text format.

### The « Sound » object

This is used to produce output sounds when the variable state of the processing application changes.

### The « Data archive » object

This is used to display processing application data in a table or chart format and save them in the computer memory or on the disk.

### The « Program » object

This is used for processing run separately from the processing application.

### The « Dialogue box » object

This is used to display messages in a pop-up window format regarding changes in the variable state of the processing application.

### The « Analog value » object

This is used to display processing application numeric variables in an analog numeric format (bars, dials etc.).

## Practical experience

In this chapter you will be able to quickly create your first IRIS 2D application. We are going to create a console, put a pushbutton on it and link the object variables to the processing application.

### Step 1

Creating a minimal application with AUTOSIM see chapter Designing programs.

This is a Grafcet with one step as shown below.



### Step 2

Launch the run of the AUTOSIM application (click on the « Go » button on the toolbar).


### Step 3

With the right side of the mouse click on the « IRIS » element on the browser and then select « Add an IRIS 2D object » from the menu. In the « Basic objects » category, double click on « Console ».

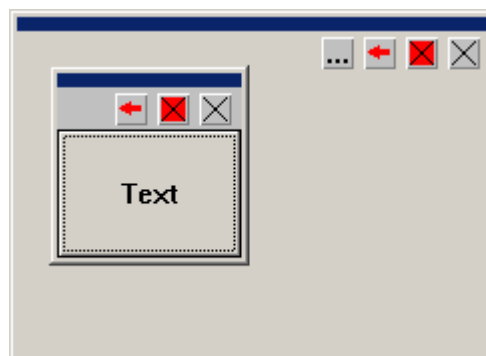
At this point the object will appear on the screen in this format:




### Step 4

To add a pushbutton to the console click on the console icon  (menu access) and select the « Add an object » option. In the « Basic objects » category, double click on « illuminated button ».

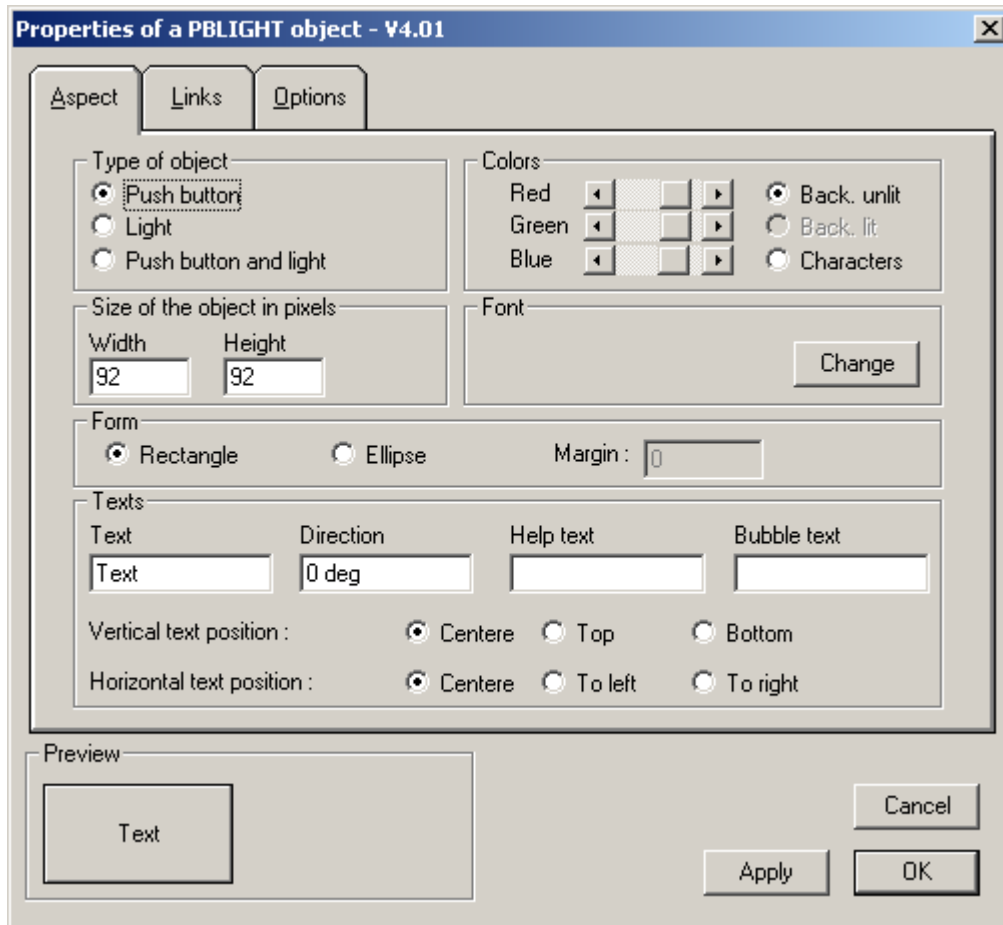
The object will then appear on the console:



### Step 5

Now we are going to associate the pushbutton to a processing application output, for example %Q4. Click the pushbutton icon  (not the console icon). The pushbutton properties dialogue box will open:







Click the « Links » tab (upper part of the dialogue window). In the « Action when button is pressed » section enter « %Q4=1 ». In the « Action when button is released » section enter « %Q4=0 ». Then click on « OK » on the pushbutton on the lower part of the dialogue window. Actions on the pushbutton will drive processing application output 4. You can open a « Monitoring » window from the « Set-up » menu by clicking with the right side of the mouse on the browser. You display the state of output 4 when you click then release the pushbutton.


## Step 6

We are going to associate a light to the « Illuminated Button » object, this light will be associated to a processing application input (for example 12).

Click the pushbutton icon  again. In the « Aspect » tab click on the « Pushbutton and light » radio button. Click on the « Links » tab and enter « %i2 » in the « Light state » section. Click on the « OK » pushbutton in the lower part of the property dialogue window. You can keep the state of variable « %i2 » modified (with a « Monitoring » window or by modifying the state of the physical input, if it exists).

## Step 7

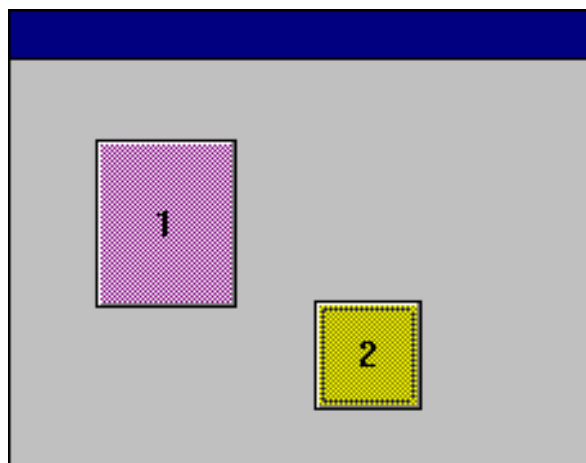
We are going to duplicate the « Illuminated Button » on the console in order to obtain a second pushbutton whose properties we will modify. Click on the pushbutton with the left side of the mouse while pressing down the [SHIFT] key. Black squares will appear around the selected object. Click on the console icon  and select the « Copy » option.

Click on the console icon  and select the « Paste » option. Now there are two overlapping « Illuminated Button » objects. Drag the upper one (it is the only accessible one) by its upper bar and move it away from the other pushbutton. The object which has been duplicated has the same properties as the first. Now you can set the parameters for the second object, for example, so it is linked to output 5 and input 3.

You can also customize the aspect of the pushbuttons by using the aspect tab for the two objects. You can modify the size of the objects by dragging their edges.


The three objects on the screen (console and two pushbuttons) are in « Configuration » mode, this means that they have a mini bar on the upper part of their surface, icons and edges for modifying their dimensions. The objects have another mode called « Employ », in this mode their aspect is permanent: the upper bar, icon and edges for modifying the dimensions no longer exist. To tilt an object, click on it with the right side of the mouse.

At this point you will have created an object that looks like this:



## Creating an autonomous supervision application

To create an autonomous supervision application (without developing a processing application with AUTOSIM) follow the procedure below:

- create correspondences for the AUTOSIM variables and the processor variables by double clicking on the « Configuration / Post-processor / <post-processor name> / Variable correspondence » element (see the post-processor manual for more information).
- compile the application by clicking on the  button on the toolbar (this validates the variable correspondence).
- configure the connection mode on « Only connect » by double clicking on « Configuration / Post-processor / <post-processor name> / Connection option ».

Comments:

- the « Automatic go » project option is used to obtain an application which automatically connects to the target to be started.
- the « Generate an executable » on the « File » menu is used to obtain an autonomous supervision application which is zipped and not covered by copyright in the format of a single executable file.

## Syntax for accessing the state of variables

You can use variable names in AUTOSIM , IEC 1131-3 or a symbol syntax. The « ... » pushbuttons located near the drag areas in the object are used to access the assistant for selecting a variable name.

### Boolean state

This syntax is used in the object « states » section.

To test the state of a boolean variable, use the variable name, for example: « i0 », « %q0 », « gate open ».

To test the complement state of a boolean variable, add a character « / » in front of the variable name, for example: « /i4 », « /%M100 », « /high level ».

To test the equality of a numeric variable with a constant, use the name of the numeric variable followed by « = », « < », « > » and a constant, for example: « %MW200=4 », « speed>2 ».

The complement state is used for creating « if different », « if less than or equal to » and « if greater than or equal to » tests, for example: « /%MW201<300 ».

The operator '&' is used to test a bit of a numeric variable, for example M200&4 tests the third bit (4 = 2 power 3) of word m200.

### Numeric state

This syntax is used in the object « states » section.

To read the state of a numeric variable, use the variable name, for example: « %MW300 », « m400 », « pressure », « \_+V\_ ».

### Modifying the state

This syntax is used in the object « order » section.

To modify the variable state, add the « = » sign followed by a constant after the variable name.

The following constants are used for boolean variables:

« 0 », « 1 », « F1 » (set to 1), « F0 » (reset), « UF » (end set), for example: « %Q0=1 », « %I10=F1 », « %I12=UF ».

For numeric variables, the constant is a number, for example: « M200=1234 », « speed=4 ».

### Special orders

The following key words can be used in the object order sections:

« RUN »: puts the target in RUN mode,

« STOP »: puts the target in stop,

« INIT »: initializes the target,

« STEP »: effects a step on the target,

« GO »: identical to the environment GO command,

« ENDGO »: stops the GO command,

« EXIT »: exits the environment,

« UCEXIT »: exits the environment without asking for confirmation,

« OPENAOF(<object>) »: displays an object. « <object> » designates an object by its title and identifier number (configured in object properties) with the « #identifier » syntax.

« CHAINAOF(<object>) »: displays an object and hides the current object. « <object> » designates an object by its title and identifier number (configured in object properties) with the « #identifier » syntax.

### Interchanging objects

« PARENTPARAM(parameter {+n} {-n}) »

This is used for a sister object to access a parent console parameter. The parameter must be set in the parent console « Links / Data for sister objects » section. See the chapter « Console » object SISTERPARAM(identifier , parameter)

When used for the OBJECT object, this syntax makes it possible to read an object's value. See the « Object » object.

SETPARAM( identifier , parameter , value)

Used to modify the object parameter.

To access the list of parameters that can be modified, click with the right side of the mouse on « Illuminated Button» while editing the action areas of an object, then select the « Parameters » command.

### Details of a « Console » object

#### « Aspect » tab

##### Window

This is used to set the aspect of the console window: presence of edges, a title bar (in this case a title can be given) presence of close and reduce icons. If you check « Display help messages » you set-up a message area at the bottom of the window, the size of this area is automatically established based on the selected font (see below). If this area is not set, messages from the sisters will be displayed on the parent console of the console and on the bottom of the AUTOSIM environment window (if the object does not have a parent).

## Console background

This establishes the console background: color (see below), transparent (accessible only if the console is the sister of another console), bitmap (the background is set by a « .BMP » file, for example created with PAINTBRUSH).

## Colors

This is used to select the color for the console background (if a colored background is selected - see above), the background and the characters of the help message display area (if this area is valid - see above).

## Fonts for the help area

This establishes the font used for displaying help messages at the bottom of the console.

## Object size

This establishes object dimensions in number of dots. These values can be modified to precisely set the size of an object.

## Texts

Help text and bubble text.

## « Bitmap » tab

### Bitmap

If the console background contains a bitmap (see « Aspect » tab) the editing area must contain a complete access name to a « .BMP » file (16 color, 256 color and 24 bits formats are supported).

The « SCAN » and « EDITOR » pushbuttons are respectively used to search for a « .BMP » file and edit a file with WINDOWS PAINTBRUSH software.

## « Links » tab

### Data for sister objects

This editing area is used to set parameters that sister objects can access with the key word « PARENTPARAM ». One setting per line must be written. Each setting must comply with the following syntax: « PARAMETER=VALUE ».

## « Options » tab

### Grid

This is used to set a grid (invisible) for positioning objects. Only the « Move » command on the console integrated menu uses the grid. Grid values are expressed in number of pixels. Values 0 and 1 cancel the grid effect. This function must be used to perfectly align objects.

### Resetting sisters

If you check « Continue to reset sisters ... » you establish that the sister must continue to be updated when the console is changed to an icon. This option is used, when it is not selected, to increase system performance when a console changed to an icon only contains visual elements.

## « Sisters » tab

### Sisters

This section contains the list of console sister objects. The « Properties » pushbutton is used to directly open the properties dialogue box for the sister selected from the list. The « Destroy » pushbutton eliminates the selected object. The « Positions » editing areas are used to set object positions.

## « External » tab

### Executable name

Name of an executable file operating on the console.

### Parameters

Parameters provided on the command line for the executable.

## Details of an « Illuminated Button » object

## « Aspect » tab

### Object type

This is used to select the object type: pushbutton, light or pushbutton integrated with a light.

## Colors

This is used to select the object color. If the object is a pushbutton, the « Background off » setting represents the color of the pushbutton. If the object is a light or a pushbutton integrated with a light the « Background on » setting establishes the color of the background when the light is on and « Background off » when the light is off. If the object aspect is established by a bitmap only the character color can be set.

## Object size

This establishes object dimensions in number of dots. These values can be modified to precisely set the size of an object. This is necessary if the object aspect is established by a bitmap.

## Font

This is used to select character font and size. The font file used must be present on the PC where the program is run.

## Text

This is used to specify the text displayed on the object, its position, its print direction as well as the help text displayed when the button is pressed and a bubble text which is displayed when the cursor is placed on the object.

## « Links » tab

### Action when

This is used to set the actions to be effected when the button is pressed and when it is released.

An action can be setting the state of a variable, for example:

```
00=1, m200=4, _depart cycle_=3
```

Or a preset key word

Configuration example where the input i10 reflects the pushbutton state (i10 to 0 if the button is released, i10 to 1 if the button is pressed):

Action when the button is pressed: i10=1

Action when the button is released: i10=0

### Light state

Establishes the light state. This section must contain the name of a variable which drives the light: 0 = light off, 1 = light on.

For example:

```
b31, o4, _light init_, m200=50, m400<8, m500&16
```



## Identifier

This is used to refer to an object in relation to the other objects.

## Deactivation condition

This is used to deactivate the light. If this section contains a variable name, then that variable deactivates the object if it is true.

## « Options » tab

### Type of pushbutton

This establishes if the pushbutton is bistable (it remains pressed) monostable or a combination of the two: monostable with a simple click and bistable with a double click.

### Keyboard

This is used to associate a key to a pushbutton. If this key or combination of keys is present on the keyboard then the pushbutton will be pressed.

Different syntaxes can be used to set the key code:

- a simple character: For example A, Z, 2,
- the \$ character followed by hexadecimal key code,
- the name of a function key, for example F5.

For combinations of keys CTRL+ or SHIFT+ must be added to the beginning.

For example: CTRL+F4 or SHIFT+Z.

### Bitmap

This is used to specify a bitmap which contains the design of an object.

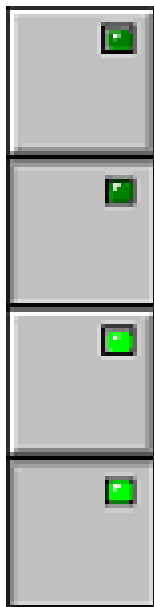
The « Resize the image » option is used to extend the bitmap over the entire surface of the object.

The bitmap file contains the four possible object aspects: button released light off, button pressed light off, button released light on, button pressed light on.

Even if the file is a pushbutton without a light or a light there are always four aspects of which only two are used.

The bitmap file is divided horizontally in four.

Example:



The « Different aspect if the cursor is on the object... » option is used to display a different image when the cursor passes over the object.

If this option is checked, the bitmap file contains 8 aspects, four supplementary aspects are added to the right of the bitmap to contain the design of the object when the cursor is on the object.

Example:



## Sounds

If WAV files are selected, the object can produce sounds if the object is pressed, released or if the cursor is on the object.

## Details of a « Digital value » object

### « Aspect » tab

#### Format

This is used to set the type of display:

- Always display the sign: display the '+' sign for positively signed values,
- Signed value: sets the signed or unsigned mode for 16 or 32 bit integers (only base 10),
- Display all digits: display the 0 at the beginning of the value if necessary.

#### Base

- Establishes the display base for 16 and 32 bit integers.

#### Colors

This is used to select the background colors of the object (if it is not transparent) and the characters.

#### Font

This is used to select character font and size. The font file used must be present on the PC where the program is run.

#### Number of digits

Sets the length of the integer and decimal parts.

#### Background

This is used to select either a colored or transparent (if the object is only placed on one console) background.

#### Object size

This establishes object dimensions in number of dots. These values can be modified to precisely set the size of an object.

### « Texts » tab

#### Bubble Text

Text displayed in a bubble when the user puts the cursor on the object.

**Text displayed before and after the value**

This is used to display information to the left and right of a numeric value.

« Links » tab

**Variable or symbol**

This designates the variable to display. To access a time delay counter or procedure the following syntax must be used:

- for the counter: COUNT (time delay), example: COUNT(t3),
- for the procedure: PRED(TIME DELAY), EXAMPLE: PRED(t7),

**The Variable state can be modified**

If this is checked then the user can modify the variable state by clicking on the object.

**Details of an « Analog value » object**

« Aspect » tab

**Objects**

This is used to set the type of display.

**Print direction**

This establishes print direction: horizontal or vertical.

**Colors**

This is used to select the background colors of the object (if it is not transparent) and the object.

**Background**

This is used to select either a colored or transparent (if the object is only placed on one console) background.

**Object size**

This establishes object dimensions in number of dots. These values can be modified to precisely set the size of an object.

**Texts**

Bubble text.

## « Links » tab

### Variable or symbol

This designates the variable linked to an object (a word or a counter).

### User action ...

This establishes if a variable can be modified by the user.

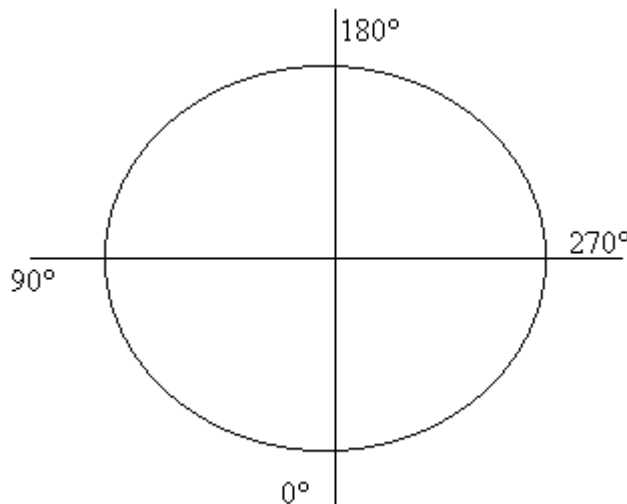
## « Limits » tab

### Minimum, maximum

Minimum and maximum values.

### Start angle, end angle

To display the type of dial which establishes the start angle and end angle. The values are expressed in degrees:



## « Graduations » tab

### Using the graduations

This validates or invalidates the use of graduations

**Start value, end value**

Values displayed for the graduations, these values can be signed and/or floating point numbers.

**No small graduations, no large graduations**

No graduations (two levels) related to start and end values. These values can be floating point numbers.

**Font**

This establishes the characters used for the graduations.

**Area N°1, area N°2 and area N°3**

This is used to establish colored areas. « Start value » and « End value » set each area. The color for each area is specified by three components of red, green and blue between 0 and 255.

**Colors**

This establishes the character and graduation color. Again here the colors are expressed by their three components: red, green and blue.

**Details of « Screen, keyboard, message list » object****Links with the application**

The link between the object and the application is made using word tables.

To send data to a type of object (with or without the keyboard) the data must be placed starting from the second word of the reception table plus the length of the data in the first word in the table (maximum length is 255). Each word contains a datum.

The data can be: an ASCII character, a number of a preset message + 8000 hexa, or a special command: 100 hexa deletes the window, 200 hexa displays the date, 300 hexa displays the time, 400 displays the message number.

When the object has reread the available data in a table it resets the first word to 0 to indicate that the operation has been effected.

The principle is the same for « with keyboard » objects: the first word of the transmission table contains the number of characters entered on the keyboard, the following words contain the characters (one per word). The application must reset the first word to 0 when it has used the data.

The interchange table for the « Message box, alarm list » object has a fixed length of 10 words. As is true for the « Screen » type the first word starts the message display. If it is different than 0 it designates a message number to be displayed. Only registered messages can be displayed. The first word can also take an ffff hexa value to clear the message box.

Description of 10 words used for interchanges with the « Message box »:

Word 0 represents the first word on the table, Word 1 the second, etc.  
 Word 0: message number to be displayed if 0 is no messages or ffff hexa to clear all messages,  
 Word 1: class number for the message (see chapter message classes for a more detailed explanation).


The following words establish the date and time and can displayed for each message. A value equal to ffff hexa asks the object to use the current computer date and time (this does not include milliseconds).


Word 2: day  
 Word 3: month  
 Word 4: year  
 Word 5: hours  
 Word 6: minutes  
 Word 7: seconds  
 Word 8: milliseconds  
 Word 9: reserved (put 0)

## Message classes

Message classes are used to classify messages into families which share the following characteristics: background color, character color and an icon.

there are two preset classes:

- the information message class: blue characters on a white background, icon , it bears the number -1,

- the alarm message class: white characters on a red background, icon , it bears the number -2.

Other classes can be set by the user.  
A bubble text can be associated with the object.

#### « Aspect » tab

### Object type

This is used to set an object type. See chapter links with the application

### Colors

This is used to select the background colors of the object and the characters.

### Font

This is used to select the character font used for displaying texts.

### Object size

This establishes object dimensions in number of dots. These values can be modified to precisely set the size of an object.

### Texts

Bubble text.

#### « Links » tab

### Reception, transmission

This sets the first variables of the reception and transmission tables. These areas can contain a variable name or symbol.

#### « List » tab

These sections do not regard « Message box » objects.

### Icons

If this is checked an icon is displayed before the messages.

### Classes

If this is checked a message class number is displayed



**Days, Months, Years, Hours, Minutes, Seconds, 1/1000 seconds**

If these are checked each one of these elements is displayed.

**Messages**

If this is checked a message is displayed.

**Numbers**

If this is checked a message display number is displayed.

**Message classes**

This editing area is used to establish new message classes. Each line sets a class. The following must appear in order and be separated by commas on each line: the background color (three components red, green and blue), the character color (three components red, green and blue), the class name, the bitmap file name for the icon associated to the class.

For example:

255,0,0,0,0,0,ALARM,alarm.bmp

Means:

Red background color, black character color, ALARM class name, file containing icon: « alarm.bmp ».

« Options » tab

**Displaying character hexadecimal codes**

This option is used to display hexadecimal code for each character in place of its ASCII representation. It is used for « Screen ... » type objects and is normally used for starting up programs.

**Horizontal, vertical scroll bar**

Displays or hides scroll bars.

**Converting OEM characters to ANSI**

If this is checked, the characters from the processing application are automatically converted from OEM characters (MS-DOS) to ANSI characters (WINDOWS). The reverse conversion is applied to characters which drive the object for the processing application.

**Duplicating messages to ...**

This section can receive a file or peripheral name (for example, « LPT1 » for the printer) It is possible to specify multiple files and/or peripherals by separating them with a comma. The displays will be automatically duplicated: Printing « edge of the water ».

**Associating a message storage file ...**

This is used for setting a file which will be associated to the object and used for storing information. If this file exists then the messages will be saved (according to the number set in the « number of memorized lines» section, when the number is reached the oldest data is deleted. When the object is open, and if a storage file exists since its last use, then the data contained in the file is transferred to the object.

**Write the old message to ...**

This is used to set a file or a peripheral which receives old messages (the messages which are eliminated from the storage file to make room).

**Number of memorized lines ...**

This establishes the message storage file capacity in number of lines. The value 0 attributes the maximum space that can be used (not a fixed limit).

**« Messages » tab****Preset messages**

This editing box is used to document preset messages (one per line).

**Details of « Data archive » object****« Aspect » tab****Objects**

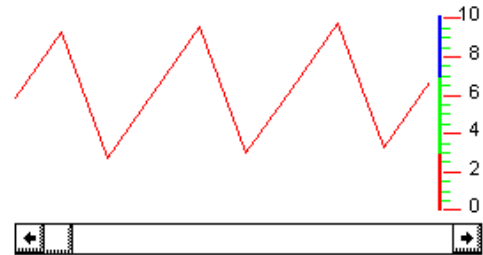
This is used to set the type of display.

The object can be represented in table format (figure 1.1) or graph format (figure 1.2).

Date	Heure d'acquisition	Valeur
23/07/96	16.52.52.443	-28043
23/07/96	16.52.53.541	-6059
23/07/96	16.52.54.640	16477
23/07/96	16.52.55.738	-26575
23/07/96	16.52.56.837	-4091
23/07/96	16.52.57.935	18441
23/07/96	16.52.59.034	-24579
23/07/96	16.53.00.132	-2067



(figure 1.1)



(figure 1.2)

## Colors

This is used to select the font color when the object is in a table format as well as color for marking values on the graph.

## Object size

This establishes object dimensions in number of dots. These values can be modified to precisely set the size of an object.

## Text

A bubble text associated with the object.

« Data » tab

## First variable to read

This is used to select the first variable to be archived.

## Number of variables to read

This indicates to the ARCHIVE object the consecutive number of variables to the « First variable to read » that it must archive.

## Number of memorized registrations

This is used to size memory database.

A registration represents an acquisition of « n » variables (« n » is the number of variables to read).

## Periodic reading

Variable acquisition will be done at fixed intervals of ARCHIVE object running.

## Start reading

Variable acquisition will be effected when the « Control word » has given the order.

## Period

This is used to establish the time between two acquisitions. The time is in Day(s)/Hour(s)/Minute(s)/Second(s)/Millisecond(s) format:

J for days

H for hours

M for minutes

S for seconds

MS for milliseconds

E.g.: 2J

E.g.: 2H10M15S

## Control

This is used to set a variable (a word) that controls the ARCHIVE object. From the value taken in the count, its contents is reset by the ARCHIVE object.

<u>Value</u>	<u>Action</u>
0	Nothing
1	Start an acquisition (Reading started)
2	Freeze the acquisitions
3	Restart archiving (after freezing)
4	Clear the memory database
5	Destroy the archive file
6	Activate « Save last acquisitions » mode
7	Cancel « Save last acquisitions » mode

« Options » tab

## Use the image file

The image file is used:

At the end of using the ARCHIVE object, to save the database present in the memory.

When the ARCHIVE object is launched, to reconstruct the database present in the memory during the last use.

## Using the archive file

Each acquisition is saved in the file in standard database format.

## Displaying

**Acquisition date:** This is used to display the acquisition date of a registration.

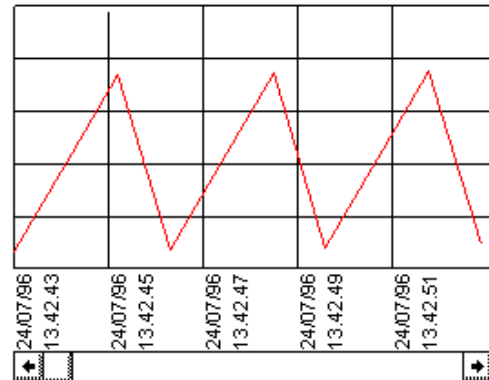
**Acquisition time:** This is used to display the acquisition time of a registration.

**Hours, minutes, seconds, milliseconds:** This is used to configure the acquisition time display.

The time display is effected downstream from the display of acquisitions for the TABLE object (figure 3.1) or under the grid when it is set for the GRAPH object (figure 3.2)

Date	Heure d'acquisition	Valeur
24/07/96	13.42.43.112	3141
24/07/96	13.42.44.211	25217
24/07/96	13.42.45.309	-18451
24/07/96	13.42.46.408	3489
24/07/96	13.42.47.506	25525
24/07/96	13.42.48.605	-17931
24/07/96	13.42.49.703	4085
24/07/96	13.42.50.802	26149
24/07/96	13.42.51.900	-17375
24/07/96	13.42.52.999	4709

(figure 3.1)



(figure 3.2)

## « Tables » tab

### Font

This is used to select a font for displaying the column name, times and acquisition value.

### Column name

This is used to set the column name for the TABLE object as well as the display format for these columns (figure 4.1)

syntax: name, format

#### **format \***

*no format specified*

h

d

ns

s

nv

v

#### **Display**

Signed, decimal, visible

Hexadecimal

Decimal

Not signed

Signed

Not visible

Visible

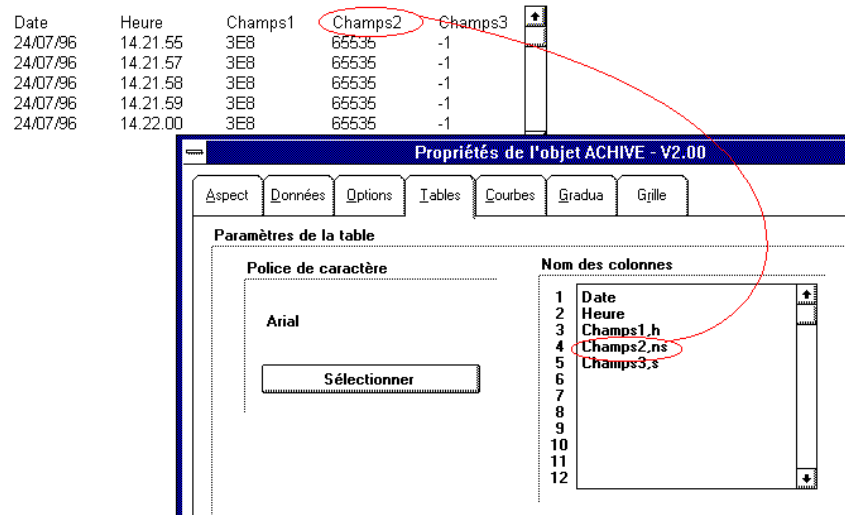
\* The different options can be combined, for example:

#### **Format**

d,ns,v

#### **Display**

Decimal without sign visible



(figure 4.1)

## « Graph » tab

### Minimum, maximum value

This is used to select the minimum and maximum values for displaying graphs.

Only values included between the minimum and maximum values will be displayed on the screen.

### Display

This is used to set the display time.

This is communicated to the ARCHIVE object in the day(s)/Hour(s)/Minute(s)/Second(s)/Millisecond(s) format:

J for days

H for hours

M for minutes

S for seconds

MS for milliseconds

E.g.: **Display\_2H30M10S**

E.g.: **Display\_100MS**

### Plotting values on the graph

This is used to make a mark on the graph for each acquisition ([figure 5.1](#))

## Displaying time

This is used to display the date and time of an acquisition of one or more variables on the grid if it is open. Colors and fonts can be set for the time display.

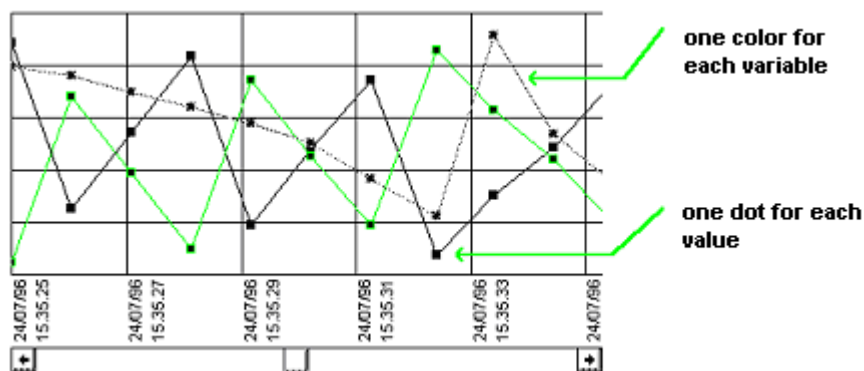
## Outline colors

This is used to set a color for each graph. The first graph has the color of the first line, the second graph has the color of the second line etc.

Colors are in Red, Green, Blue format.

E.g.: 255,0,0 red outline

If a color is not set on a line, the graph corresponding to this line will not be outlined.



(figure 5.1)

## « Graduations » tab

### Using the graduations

This validates or invalidates the use of graduations (figure 6.1).

### Start value, end value

Values displayed for the graduations, these values can be signed and/or floating point numbers.

### No small graduations, no large graduations

No graduations (two levels) related to start and end values. These values can be floating point numbers.

### Font

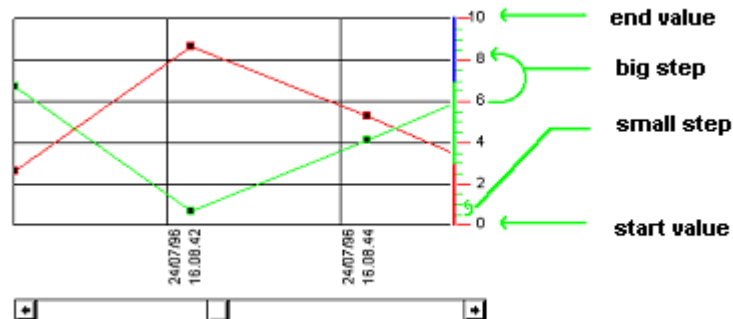
Establishes the characters used for the graduations.

### Area N°1, area N°2 and area N°3

This is used to establish colored areas. "Start value" and "End value" set each area. The color for each area is set by three components of red, green and blue between 0 and 255.

## Colors

This establishes the character and graduation color. Again here the colors are expressed by their three components: red, green and blue.



(figure 6.1)

## « Grid » tab

### Displaying the grid

This validates or invalidate grid display.

### Not for ordinates

This sets the vertical pitch of the grid.

### Not for abscissas

This sets the horizontal pitch of the grid. The pitch is in Day(s)/Hour(s)/Minute(s)/Second(s)/Millisecond(s) format:

J for days

H for hours

M for minutes

S for seconds

MS for milliseconds

E.g.: 1J

E.g.: 2H30M15S

### Color

This is used to set a color for each grid.

The color is in Red, Green, Blue format

E.g.: 255,0,0                      Red outline



## Details of « Object » object

### « Aspect » tab

#### Type

This is used to set one of the object type aspects:

- « n bitmap aspects »: the object aspect is provided by a bitmap file which can contain various aspects, see the chapter « Bitmap » tab
- « n bitmap colors »: the object aspect is provided by a bitmap file, the color is controlled by a processing application variable that replaces the blank pixels of the bitmap. The other bitmap pixels must be black. The processing application variable provides a color number, the colors are set in the « Colors » tab.
- « gauge bitmap »: the object is a gauge with a format set by a bitmap. The blank bitmap pixels set the format. The other pixels must be black. The minimum, maximum and print direction are set in the « Gauge » tab.
- « n format colors »: a rectangle, a rectangle with rounded edges or an ellipse. The color is managed in the same manner as « n bitmap colors ».
- « gauge formats »: the object is a rectangular gauge. The principle is the same as for a « gauge bitmap »

#### Colors

This is used to select the character color for the text displayed on the object.

#### Font

This establishes the font used for displaying text on the object.

#### Object size

This establishes object dimensions in number of dots. These values can be modified to precisely set the size of an object.

#### Texts

Help text and bubble text.

The text displayed on the object: the position and print direction can be modified.

## « Links » tab

### Clicked object, not clicked object

This sets the actions to be effected when the user clicks on the object and when the user stops clicking the object.

An action can be setting the state of a variable, for example:

`o0=1, m200=4, _depart cycle_=3`

Or a preset key word.

A configuration example where the input `i10` reflects the clicked state of an object (`i10` to 0 if the object is not clicked, `i10` to 1 if the object is clicked):

Clicked object: `i10=1`

Not clicked object: `i10=0`

### Permanently connect with ..

This area can receive the identifier of a sister object. If this object exists then the position of the object is modeled on it. The identifier of an object is an integer value between 1 and 32767. It is specified in the « Identifier » editing area of the « Links » section.

### Aspect/Color/Filling

This area of the dialogue box contains 8 editing areas which can be used to set different types of object behavior based on the processing application variables.

No matter what their behavior they will always have a position which depending on the type of object will design:

- an aspect contained on a bitmap for the « n bitmap aspects » type
- a color number for « n bitmap colors » or « n format colors »
- filling for the « gauge bitmap » or « gauge format » types.

The « Position » area can contain a numeric variable name (C or M). The areas « + Position » and « - Position » can contain a name of boolean variables.

Two types of operation are possible:

- if the « + Position » and « - Position » areas are documented then the boolean variables contained in them will drive the position: they add or delete the value specified in the speed area. If the « Position » area is documented then the current position is written in the variable which contains the name.
- if the « + Position » and « - Position » areas are blank then the value containing the variable where the name is written in the « Position » area will be read as the object position.

The position can vary between the values set in the « Min » and « Max » areas.

Sensors can be added (boolean variable names) which will be true for the minimum and maximum position (position equal to minimum or maximum).

### **Horizontal movement, vertical movement**

These dialogue box areas each contain 8 editing areas respectively used to set the horizontal and vertical position of the object. The principle is identical to that described above.

#### **« Formats » tab**

### **Formats**

For « n format colors » this section is used to select a rectangle, a rectangle with rounded corners or an ellipse.

#### **« Bitmap » tab**

### **File name**

For « n bitmap aspects, n bitmap colors and gauge bitmap » this editing area must contain a complete access name to a « .BMP » file. These files can be created with PAINTBRUSH or another graphics editing program able to create « .BMP » files.

The « Scan » and « Edit » pushbuttons are respectively used to search for « .BMP » files and to edit (launch of PAINTBRUSH) « .BMP » file if its name is in the editing area.

### **Number of aspects**

This editing area must contain the number of aspects (images) contained in a « .BMP » file. This option is used for « n bitmap aspects ». The

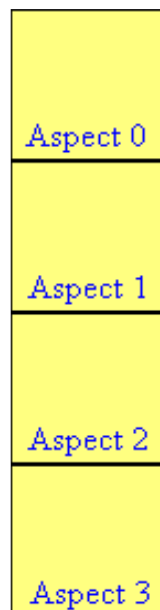
different object aspects must be designed one under the other. The highest aspect is the number 0.

« Wmf » tab

### File name

For « Meta files » this editing area must contain a complete access name to a « .EMF » file.

Example of a « .BMP » file with 4 aspects:



### The bitmap has transparent areas ...

This option is used to create an object with certain transparent areas (the background of the parent console will be displayed). The transparent areas are set by pixels of the same color, a color established by the three components, red, green and blue. To set these components use the three scroll bars. The color must be precisely set: exactly the same proportion of red, green and blue as the color of the pixels in the transparent areas.

« Colors » tab

### Colors

This area is used for « n bitmap colors » and « n format colors ». Each line contains the setting for a color. The syntax used for each line is:

proportion of red (between 0 and 255), proportion of green (between 0 and 255) and proportion of blue (between 0 and 255). The first line designates color number 0, the second line number 1, etc.

This area is used for « gauge bitmap » and « gauge format ». The first line (color 0) and the second (color 1) establishes the two colors of the gauge (active and inactive part).

### « Gauge » tab

#### Gauge

This section is used for « gauge bitmap » and « gauge format ». The « Minimum value » and « Maximum value » establish the limits for the gauge drive variable.

#### Gauge print direction

This establishes one of the four possible directions for the gauge.

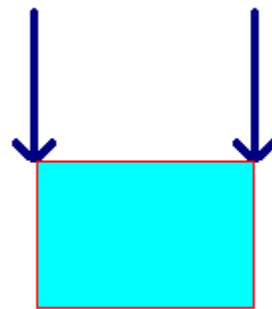
### « Sensor » tab

The OBJECT object can be used as a sensor. The sensor is associated with a boolean variable where the result is true if the sensor is in contact with one or more of the preset colors (see below), otherwise it is false.

#### Detection position

This is used to set the side of the object which must be detected. Detection is effected on the two edges of the selected side.

Example for detection from below:



#### Detected colors

A sensor is capable of detecting up to three different colors. If one of these three colors is at the test points then the boolean variable associated to the sensor (see chapter « Links » tab) is positioned at 1, otherwise it is positioned at 0.

The three editing areas can contain a color setting in the format of three values between 0 and 255 which respectively correspond to the

percentages of red, green and blue. The percentages of these three colors must exactly correspond to the colors of the object to be detected in order for the sensor to work.

### « Options » tab

#### Key

Set a key used to simulate a click on an object.

Different syntaxes can be used to set the key code:

- a simple character: For example A, Z, 2,
- the \$ character is followed by hexadecimal key code,
- the name of a function key, for example F5.

For combinations of keys « CTRL+ » or « SHIFT+ » must be added to the beginning

For example: « CTRL+F4 » or « SHIFT+Z ».

#### The TAB key is used to access this object

If this is not checked then the TAB key cannot be used to activate the object.

### Advanced techniques

#### Dynamic object linking

This possibility is used to momentarily link one object to another. The « + Position » and « - Position » parameters which manage the horizontal and vertical position are used in a special way for linking one object to another. These two parameters must contain the name of a numeric variable (M). The « + Position » variable must contain the f000 value (hexadecimal) and the « - Position » the identifier of the object to be connected. The « + Position » variable is reset once the connection has been made. To cancel the object connection the value f001 (hexadecimal) must be put in the « + Position » variable. See chapter: Example of operating part simulation 1

#### Interchanging parameters between two objects

A object can access the parameters of a sister object by using the key word « SISTERPARAM ».

The syntax is:

SISTERPARAM(identifier of the sister object, parameter)

« parameter » can assume the following values:

STATE      object state: Aspect/Color/Filling value

STATE	same as above but with negative value
POSX	position on horizontal axis
POSX	same as above but with negative value
POSY	position on y axis
POSY	same as above but with negative value
POSX+STATE	position on horizontal axis plus state
POSX+STATE	position on horizontal axis minus state
POSY+STATE	position on vertical axis plus state
POSY+STATE	position on vertical axis minus state

## Details of « Sound » object

### « Aspect » tab

#### Object size

This establishes object dimensions in number of dots. These values can be modified to precisely set the size of an object.

### « Sounds » tab

#### Name of sound files

Complete access name to « .WAV » files.

#### Associated variables

The boolean variable associated to each sound.

## Details of « Dialogue box » object

### « Aspect » tab

#### Type of box

This is used to select the various controls present in the dialogue box: only one OK button, two buttons OK and CANCEL, or two buttons YES and NO.

#### Icons

This is used to select the icon that will appear in the dialogue box. There are four different icons, but it is possible not to display any of them. It is also important to note that a special system is associated to each icon. See the section on the BEEP option for more information on the subject.

**Object size**

This establishes object dimensions in number of dots. These values can be modified to precisely set the size of an object.

**Beep**

This is used to specify if the dialogue box display must be accompanied by a sound warning.

**Title**

This is used to specify the title of the dialogue box.

**Message type**

There are two possibilities. A preset message is a message present in the processing application user variables. The other possibility is to specify a message list in this case the displayed message is a function of the monitored variable state.

**« Links » tab****Variable name**

This specifies the name of the variable to monitor. Boolean or numeric variables can be entered.

For example:  
m200, i0

If the variable is boolean, then message no. 1 on the list will be displayed when the state of that variable passes to 1.

For a numeric variable, if the « Message list » configuration option is checked, then the dialogue box will be displayed when the value is between 1 and the number of messages memorized on the list.

For example, if the list contains 8 messages, then it will not display anything when the variable assumes negative values or those over 8. On the other hand, when the value is between 1 - 8, then the appropriate message is displayed.

If the « Preset message » option is activated, then the dialogue box will display a message of the length contained in the variable, and situated in the processing application variables based on that variable.



For example. if  $m200=4$ , this means that a message 4 characters long is situated in the 4 variables following  $m200$ , or rather  $m201$ ,  $m202$ ,  $m203$ ,  $m204$ .

### Dialogue box return code

With a boolean variable, no matter what action the user effects, its contents will go to 0. For a numeric variable, there are different return codes:

Press on an OK button: the variable assumes the value 8000 (hexa)

Press on an CANCEL button: the variable assumes the value 8001 (hexa)

Press on an YES button: the variable assumes the value 8002 (hexa)

Press on an NO button: the variable assumes the value 8003 (hexa)

**Comment:** Activation of a dialogue box is based on a rising edge, this means passage from 0 to 1 for a boolean variable, and passage from a value outside the message list range to a value included in it, for a numeric variable.

### Identifier

This is used to refer to an object in relation to the other objects.

« Messages » tab

### Message list

Enter the different preset messages in this area.

## Details of « Program » object

### Run time distribution

IRIS objects are run by turns. The run time distribution is managed in a straightforward manner by the object manager. two priority levels are possible for « PROG » objects: if « Priority run » is checked on the « Program » tab, then the whole program is run while the object is present. Otherwise, only one line is run before the object yields. There are exceptions to this rule: access functions to the processing variables (« READVAR » and « WRITEVAR ») may cause yielding, the YIELD

function sets a yield. In priority run mode, this function must be used inside a loop in order not to block running of other objects.

## Display

The object surface can be used for displaying information. The « PRINT » function is used to display information.

## Syntax

The character « ; » (semicolon) is used as a separator. Comments can be written between the chains « (\* » and « \*) ». There is no difference between upper and lower case letters for key words and function names, on the other hand, for variable names there is a difference.

## Stating variables

The variables used in a program must be stated before the program between the key words « BEGINVAR; » and « ENDVAR; ».

The following types of variables can be used:

INT	16 bit signed integer
UINT	16 bit unsigned integer
LONG	32 bit signed integer
ULONG	32 bit unsigned integer
STRING	string of characters
FLOAT	float

The general syntax of a statement is:

<type> <variable name>;

The general syntax for stating a variable table is:

<type> <variable name> [<length>;

For example:

```
BEGINVAR;
INT counter;      (* a 16 bit signed integer *)
STRING string;    (*a string*)
(*a table of 100 32 bit unsigned integers*)
ULONG table[100];
ENDVAR;
```

## Writing a program

The program must be written between the two key words « BEGIN; » and « END; »

Example:

```
BEGIN;  
print "Good morning !";  
END;
```

## Constants

- 16 bit integer: a decimal number between -32768 and 32727 where "S" follows a hexadecimal number between 0 and FFFF. Example: 12, -4, \$abcd
- 32 bit integer: a decimal number between -2147483648 and 214743648 where "L" or "S" follows a hexadecimal number between 0 and FFFFFFFF followed by "L". Example: 10000L, -200000L, \$12345678L
- string of characters: quotation mark characters followed by a string followed by quotation mark characters. Controls characters can be entered in a string. « \n » replaces an LF character (ASCII code 10), « \r » a CR character (ASCII code 13). Example: "Abcdef", "" (zero string), "Follow\r\n"
- float: a decimal number followed by the character "R", the characters "." are used to divide the integer part from the decimal part. Example: 3.14r, -100.4r

## Assignment

The string «:= » indicates an assignment.

Example:

```
counter:=4;  
var:="ABCDEF";
```

## Calculations

Calculation operators are evaluated from left to right. Parentheses can be used to specify a calculation priority.

List of calculation operators:

- + addition (chaining for strings)
- - subtraction
- \* multiplication
- / division
- << shift to the left
- >> shift to the right
- ^ raise by a power
- binary "and" AND
- binary "or" OR
- binary "exclusive or" XOR

Examples:

```
result:=var1*(var2+var3);  
result:=result<<2;
```

## Tests

Syntax:

```
IF <condition> THEN ... ENDIF;
```

or

```
IF <condition> THEN ... ELSE ... ENDIF;
```

Example:

```
IF (count<100) AND (count>10)  
    THEN  
    count:=count+1;  
    ELSE  
    count:=0;  
    ENDIF;
```

## Loops

### Syntax:

```
WHILE <condition> DO ... ENDWHILE;
```

### Example:

```
count:=0;
```

```
WHILE count<1000
```

```
    DO
```

```
        table[count]:=table[count+1];
```

```
        count:=count+1;
```

```
    ENDWHILE;
```

## Variable or variable table address

The syntax `&variable name` or `&variable table name` provides the address of a variable or variable table. This syntax is necessary for some functions.

## List of functions

For the proposed examples below, the following is supposed:

`vint` is an INT type variable, `vlong` is a LONG type variable, `vuint` is a UINT type variable, `vulong` is a ULONG type variable, `vfloat` is a FLOAT type variable, `vstring` is a STRING type variable.

### PRINT

Display function. The data to be displayed is written after and separated by commas. Example:

```
print "The result is:",vint/12,"\n";
```

### NOT

Complement. This function can be used with the if test to complement a result.

### Example:

```
if not(1<2) then ...
```

### ABS

Absolute value.

### Example:

```
print abs(0-4); (* display 4 *)
```

## VAL

Provides the value of a string expressed in decimal number format.

Example:

```
vlong=val("-123456"); (* vlong will contain -123456 *)
```

## HVAL

Provides the value of a string expressed in hexadecimal number format.

Example:

```
vuint=hval("abcd"); (* vuint will contain abcd hexa *)
```

## ASC

Provides the ASCII code of the first character of a string.

Example:

```
vuint:=asc("ABCD"); (* vuint will contain 65: ascii code of 'A' *)
```

## CHR

Provides a string composed of one character where the ASCII code is changed into a parameter.

Example:

```
vstring:=chr(65); (*vstring will contain string "A" *)
```

## STRING

Provides a string composed of n characters. The first subject is the number of characters, the second the character.

Example:

```
vstring:=string(100, " ");  
(* vstring will contain a string composed of 100 spaces *)
```

## STR

Converts an integer numeric value into a string representing the value in decimals.

Example:

```
vstring:=str(100); (* vstring will contain the string "100" *)
```

## HEX

Converts an integer numeric value into a string representing the value in hexadecimal.

Example:

```
vstring:=str(100); (* vstring will contain the string "64" *)
```

## LEFT

Provides the left part of a string. The first subject is the string, the second the number of characters to extract.

Example:

```
vstring:=left("abcdef",2); (* vstring will contain"ab" *)
```

## RIGHT

Provides the right part of a string. The first subject is the string, the second the number of characters to extract.

Example:

```
vstring:=right("abcdef",2); (* vstring will contain "ef" *)
```

## MID

Provides part of a string. The first subject is the string, the second the position where the extraction begins, the third the number of characters to extract.

Example:

```
vstring:=mid("abcdef",1,2); (* vstring will contain "bc" *)
```

## LEN

Provides the length of a string.

Example:

```
vuint:=len("123"); (* vuint will contain 3 *)
```

## COS

Provides the cosine of a real value expressed in radians.

Example:

```
vfloat:=cos(3.14r); (* vfloat will contain the cosine of 3.14 *)
```

## SIN

Provides the sine of a real value expressed in radians.

Example:

```
vfloat:=sin(3.14r); (* vfloat will contain the sine of 3.14 *)
```

## TAN

Provides the tangent of a real value expressed in radians.

Example:

```
vfloat:=tan(3.14r); (* vfloat will contain the tangent of 3.14 *)
```

## ATN

Provides the tangent arc of a real value.

Example:

```
vfloat:=atn(0.5r); (* vfloat will contain the tangent arc of 0.5 *)
```

## EXP

Provides the exponential of a real value.

Example:

```
vfloat:=exp(1r); (* vfloat will contain the exponential of 1 *)
```

## LOG

Provides the logarithm of a real value.

**Example:**

```
vfloat:=log(1r); (* vfloat will contain the logarithm of 1 *)
```

**LOG10**

Provides the base 10 logarithm of a real value.

**Example:**

```
vfloat:=log10(1r);  
  
(* vfloat will contain the base 10 logarithm of 1 *)
```

**SQRT**

Provides the square root of a real value.

**Example:**

```
vfloat:=sqrt(2); (* vfloat will contain the square root of 2 *)
```

**DATE**

Provides a string representing the date.

**Example:**

```
print "The date is:",date(),"\n";
```

**TIME**

Provides a string representing the time.

**Example:**

```
print "The time is:",time(),"\n";
```

**RND**

Provides a random number.

**Example:**

```
print rnd();
```

**OPEN**

Opens a file. The first subject is the file name, the second the access mode, which can be: « r+b » opening in reading and writing, « w+b » opening in writing (if the file exists it is destroyed). The function provides a long which identifies the file. If the opening fails, the value provided is 0.

**Example:**

```
vulong:=open("new","w+b");
```

**CLOSE**

Closes a file. The subject is the file identifier provided by the OPEN function.

**Example:**

```
close(vulong);
```



## WRITE

Writes data in a file. The first subject is the file identifier provided by the OPEN function. The second subject is a variable address, the third the number of bytes to be written. The function provides the number of bytes actually written.

Example:

```
vuint:=write(vulong,&buff,5);
```

## READ

Reads data in a file. The first subject is the file identifier provided by the OPEN function. The second subject is a variable address, the third the number of bytes to be read. The function provides the number of bytes actually read.

Example:

```
vuint:=read(vulong,&buff,5);
```

## SEEK

Moves a file pointer. The first subject is the file identifier provided by the OPEN function, the second the position.

Example:

```
seek(vulong,01);
```

## GOTO

Effects a jump to a label in the subject. The subject is a string.

Example:

```
goto "end"
```

```
...
```

```
end;;
```

## CALL

Effects a jump to a subprogram. The subject is a string containing the subprogram label.

Example:

```
BEGIN;
```

```
(* main program *)
```

```
call "sp"
```

```
END;
```

```
BEGIN;
```

```
(* subprogram *)
```

```
sp:
```

```
print "In the subprogram\n";
```

```
return;
```

END;

## RETURN

Indicates the end of a subprogram.

## READVAR

Reads one or more variables of the processing application. The first subject is the processing variable name (variable or symbol name). The second subject is the variable or 32 bit (longs or floats) variable table address. The third subject is the number of variables to be read. If the function is executed with no errors, the value of 0 is provided.

Example:

```
readvar("i0",&buff,16); (* read 16 integers starting from i0 *)
```

## WRITEVAR

Writes one or more variables of the processing application. The first subject is the processing variable name (variable or symbol name). The second subject is the variable or 32 bit (longs or floats) variable table address. The third subject is the number of variables to be written. If the function is executed with no errors, the value of 0 is provided.

Example:

```
writevar("o0",&buff,16);  
(* write 16 outputs starting from o0 *)
```

## CMD

Executes a command. The subject is a string which specifies the command to be executed. This function makes it possible to use preset IRIS commands. For more information see the chapter Special orders . If the command is executed with no errors, the value of 0 is provided.

Example:

```
cmd("run");
```

## YIELD

Yields control. This function is used so as not to monopolize the execution when the object is run in priority mode.

Example:

```
WHILE 1  
    DO  
        ...  
        yield();  
ENDWHILE;
```

## DLL

Calls up a DLL. The first subject is the DLL file name. The second is the function name. The third is a pointer on a 32 bit variable which will receive the function return code. The other subjects are passed to the function.

Example:

```
dll "user", "messagebeep", &vulong, -1;
```

## Error messages

« separator ‘;’	missing »a semicolon is missing
« syntax error »	syntax error detected
« variable set more than once »	a variable set more than once
« not enough memory »	the program run has saturated the available memory
« variable not set »	a variable used in the program has not been set
« constant too big »	a constant is too big
« program too complex »	an expression is too complex, it must be broken down
« incompatible variable or constant type »	a variable or constant is not the expected type
« ’)’ missing »	A closing parenthesis is missing
« ENDIF missing »	The key word ENDIF is missing
« ’ENDWHILE’ missing »	The key word ENDWHILE is missing
« label cannot be found »	a goto or subprogram label cannot be found
« ’)’ missing »	the closing square bracket is

	missing
« element number outside limit »	a table element outside of the limits has been used
« too many overlapping 'CALL' »	too many overlapping subprograms have been used
« 'RETURN' found without 'CALL' »	RETURN found outside a subprogram
« variable size too small »	the size of a variable is insufficient
« DLL file cannot be found »	the DLL file cannot be found
« function cannot be found in DLL »	the function cannot be found in the DLL file
« division by zero »	a division by 0 has been produced»
« mathematical error »	a mathematical function has caused an error

## « Aspect » tab

### Colors

This is used to select the object background and character color.

### Object size

This establishes object dimensions in number of dots. These values can be modified to precisely set the size of an object.

### Text

This is used to specify a bubble text which is displayed when the cursor is on the object.

## « Program » tab

### Program

This editing area contains the program.

**Run**

If this is checked then the program is run.

**Priority run**

If this is checked then the program is run more rapidly.

**Run at start-up**

If this is checked then the program is run when the object is opened. This option is used to save an object with the « Run » option not checked by requesting a run when the object is loaded.

**Go to the error**

If an error has been detected when a program is running, then the pushbutton is used to place the cursor in the place that caused the error.

## IRIS 2D examples

The examples file names refer to the « Examples » subdirectory of the directory where AUTOSIM is installed.

### Example of composed objects

This example is used to let you understand how to create a « Decimal keyboard » object composed of keys. « 0 » to « 9 » plus a key [ENTER] for validating.

You will create a « Console » object, then starting from the console menu you will create an « Illuminated Button » object. We are going to set parameters for this object then we will duplicate it to obtain other keys. Then we will adjust the duplicated key properties to customize them: text display on the key and action We will then have a keyboard with a uniform key aspect.

Link with the application will be effected by using a word.

When a key is pressed it will write its code (0 to 9 or 13 for the validation key) in that word.

To specify that word we can give its name in the action section of the properties for each object. The problem is that when we reuse the « Decimal keyboard » object and if we want to use another word, it is necessary to modify the properties of the 11 « Illuminated button » objects.

To get around this problem we are going to use the possibility that sister objects have of accessing a parameter set in the properties of the parent console. The « Links » tab of the console property window is used to set the parameter. Only write on one line in the editing area. « KEYBOARD=M200 ». This line means that the keyboard parameter is equal to M200.

The keyboard keys refer to the « KEYBOARD » parameter and not directly to word M200. To change the word used, just change the parameter setting in the console properties.

Going back to the aspect of our keyboard...

In order for the aspect of the keyboard to be satisfactory we are going to set a grid to align the keys. In the console properties window and the « Options » tab write the value « 10 » in the two « Grids » sections. This way the function moved from the console menu will use a 10 pixel grid. We are also going to set the dimensions for the first key. We can directly modify the dimensions of the key by dragging it by its edges, but for greater precision we are going to modify the dimensions from the « Object size in pixels » section of the « Illuminated Button » object window property tab.

For example, enter « 30 » for the width and height.

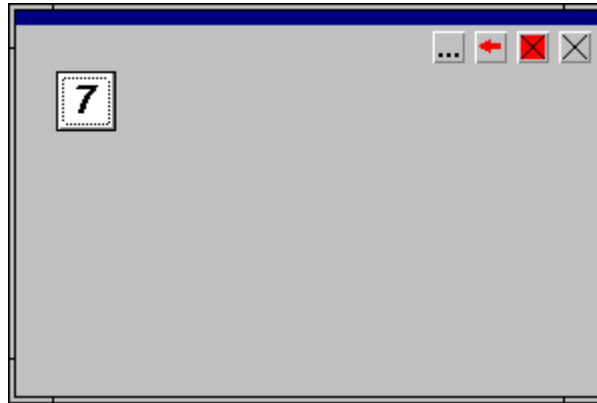
At this point you can also customize the style of the key. the color and font used for marking etc.

We are going to place this first key to the upper left of the keyboard (this is an arbitrary choice). The keyboard we are going to create will look like the numberpad of a computer keyboard. We are then going to mark this key with the text « 7 » in the « Text » section of the « Aspect » tab.

We are also going to set parameters for the functional aspect of the key: in the « Action when the button is pressed » section of the « Links » tab we are going to write: « PARENTPARAM(KEYBOARD)=7 ». This means that when the pushbutton is pressed the word designated for the « KEYBOARD » parameter of the parent console will receive the value 7. Delete whatever is in the « Action when the pushbutton is released' » section.

We can also assign a computer keyboard key to the « Illuminated Button » object. Then it will be possible to use the keyboard with the mouse or computer keyboard. To assign a key to the « Illuminated Button », object use the « Key » section of the « Options » tab. For example, enter « 7 » to associate computer keyboard key « 7 » to the object.

Then place key « 7 » at the upper left of the keyboard, like this:



To move this key, select the object ((SHIFT) key pressed, then click with the left side of the mouse on the object), then use the « Move » function from the console menu. This function is the only one which uses the grid instead of moving by dragging the bar of sister objects.

To create other keys, duplicate the existing key:

- select the first key,
- select « Copy » from the console menu, then « Paste »
- move the previously pasted key,
- set parameters for the new key: (text, links and computer keyboard key).

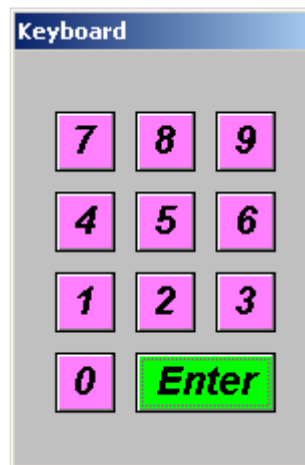
When you have finished the above row (keys « 7 », « 8 » and « 9 ») you can then select all three keys together and duplicate them.

You can create a validation key (wider for filling the surface of the keyboard).

To finish, resize the console and put the objects in « Employ » mode.



The final result should look like this:



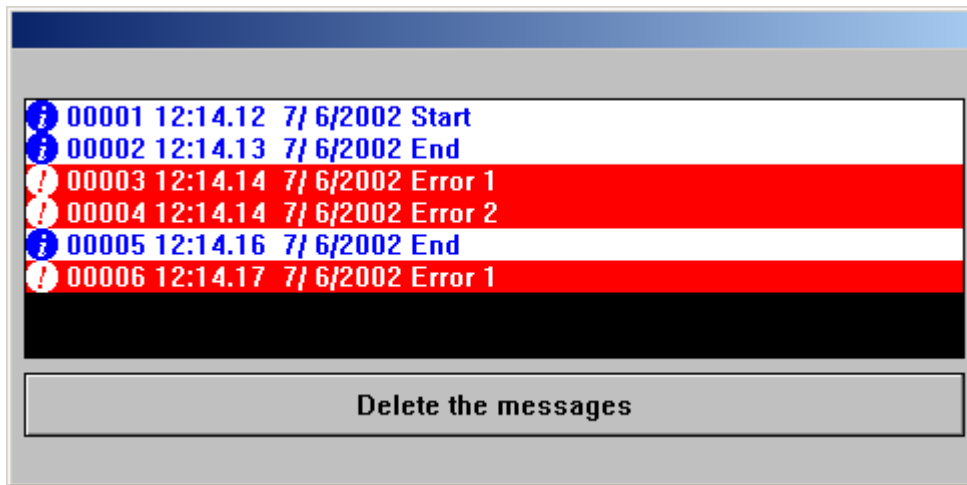
 « Examples\IRIS2D\keyboard.agn »

Example of using the « Screen, keyboard, message list » object as a message list

Instructions:

- the object must display four different messages based on the state of four inputs (i0 to i3),
- for input 0: an information message « Start cycle »,
- for input 1: an information message « End cycle »,
- for input 2: an error message « Error 1 »,
- for input 3: an error message « Error 2 ».
- the messages must be displayed when the rising edge of the inputs appears,
- a record of 50 messages will be kept in the object and saved on the disk,
- the messages will be duplicated by a printer connected on « LPT1: »,
- a pushbutton must be used to delete the messages.

Solution:



 « Examples\IRIS2D\screen keyboard 1.agn »

Variation:

Pressing on the pushbutton « Delete the messages » causes the « Do you want to delete messages » dialogue box to open with a choice of YES or NO.

Solution:

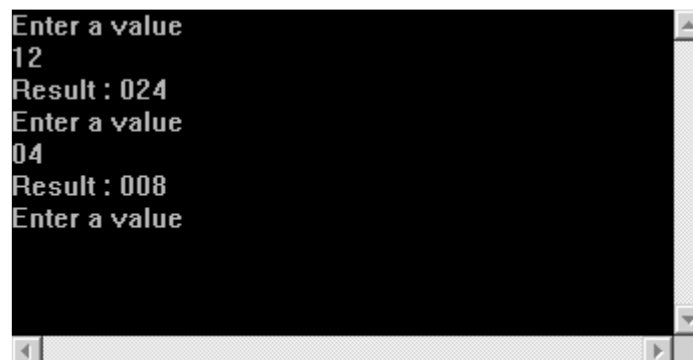
 « Examples\IRIS2D\Screen keyboard 2.agn »

### Example of using the « SCREEN KEY » object as a terminal

Instructions:

Display a message « Enter a value », requires that a decimal value be typed on the keyboard (two characters) then displays that value multiplied by two after the « Result: » text.

Solution:



 « Examples\IRIS2D\terminal 1.agn »

**Variation:**

The displayed messages are stored in the object and no longer in the processing application.

**Solution:**

 « Examples\IRIS2D\terminal 2.agn »

### Example of an application composed of multiple pages

This example will let you understand how to create an application composed of multiple elements: in this case a menu is used to access two different pages.

 « Examples\IRIS2D\menu.agn »

### Example of using the «OBJECT » object

Simulation of a jack.

**Instructions:**

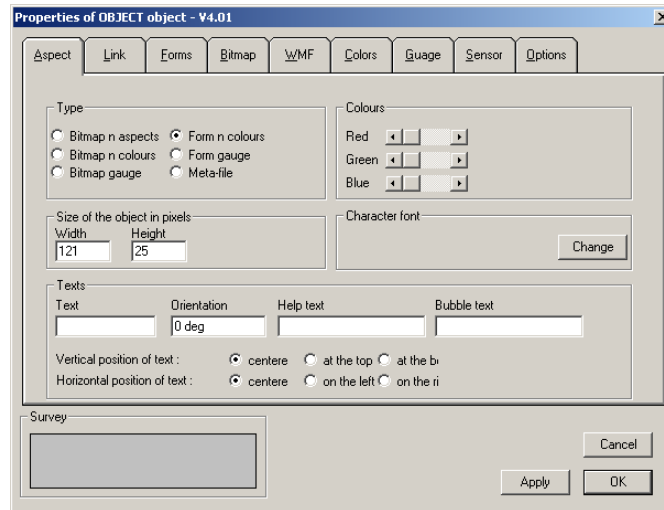
- jack driven by two o0 outputs (extract the jack) and o1 (retract the jack).
- two limit inputs i0 (jack retracted) and i1 (jack extracted).

Three objects will be used:

- a « Console » object acting as support,
- an « Object » for the jack body,
- an « Object » for the jack shaft.

Solution:

The jack body is an OBJECT object which remains static, only its aspect is configured:



Properties of OBJECT object - V4.01

Aspect Link Forms Bitmap WMF Colors Gauge Sensor Options

Type

☐ Bitmap n aspects ☒ Form n colours

☐ Bitmap n colours ☐ Form gauge

☐ Bitmap gauge ☐ Meta-file

Colours

Red

Green

Blue

Size of the object in pixels

Width: 121 Height: 25

Character font

Texts

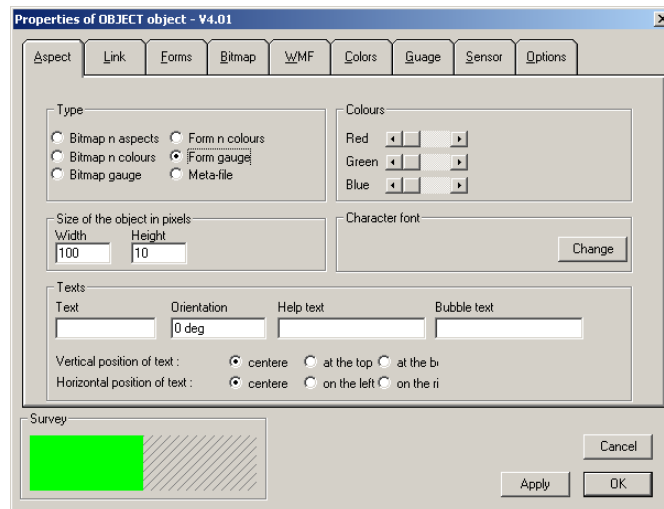
Text:  Orientation: 0 deg Help text:  Bubble text:

Vertical position of text: ☒ centere ☐ at the top ☐ at the b

Horizontal position of text: ☒ centere ☐ on the left ☐ on the ri

Survey

The jack shaft is an OBJECT object configured as follows:



Properties of OBJECT object - V4.01

Aspect Link Forms Bitmap WMF Colors Gauge Sensor Options

Type

☐ Bitmap n aspects ☐ Form n colours

☐ Bitmap n colours ☒ Form gauge

☐ Bitmap gauge ☐ Meta-file

Colours

Red

Green

Blue

Size of the object in pixels

Width: 100 Height: 10

Character font

Texts

Text:  Orientation: 0 deg Help text:  Bubble text:

Vertical position of text: ☒ centere ☐ at the top ☐ at the b

Horizontal position of text: ☒ centere ☐ on the left ☐ on the ri

Survey

Properties of OBJECT object - V4.01

Aspect Link Forms Bitmap WMF Colors Gauge Sensor Options

Links

Action

Object clicked Object not clicked

Horizontal transfer (x axis)

Position + Position - Position Capt.min

Link permanently with(identifier)

Mini Maxi Speed ms Capt. max

0 200 5

Aspect / Colour / Filling

Position + Position - Position Capt.min

0 01 0

Mini Maxi Speed ms Capt. max

0 100 5 1

Identifier

0

Vertical transfer (y axis)

Position + Position - Position Capt.min

Mini Maxi Speed ms Capt. max

0 200 5

Sensor

Survey

Cancel

Apply OK

Properties of OBJECT object - V4.01

Aspect Link Forms Bitmap WMF Colors Gauge Sensor Options

0 0,255,0

1

2

3

4

5

6

7

8

In the lines above the colours must be individualised by writing in the order of the three components red, green and blue separated by a comma. these components are represented by a value between 0 and 255

Survey

Cancel

Apply OK

Properties of OBJECT object - V4.01

Aspect Link Forms Bitmap WMF Colors Gauge Sensor Options

Minimum

0

Maximum

100

Orientation of the gauge:

from the bottom towards the top

from the top towards the bottom

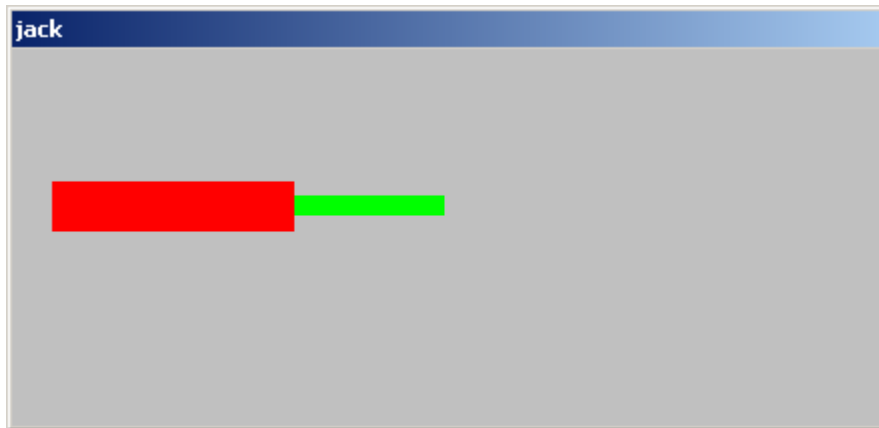
from the left towards the right

from the right towards the left

Survey

Cancel

Apply OK



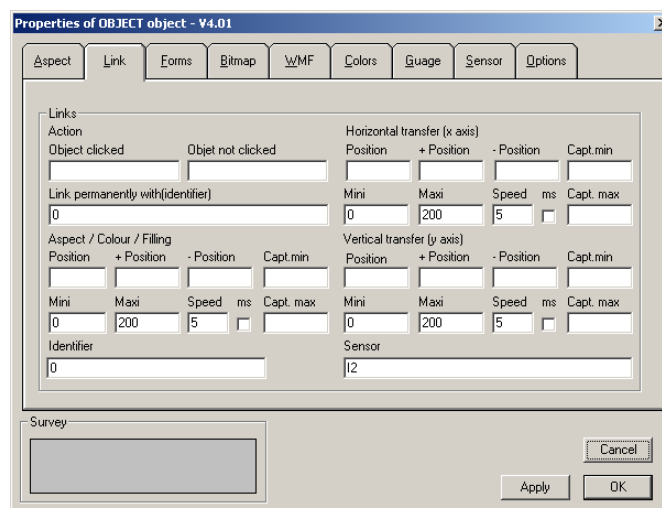
« Examples\Process Simulation\2D\tutorial1.agn »

### Variation:

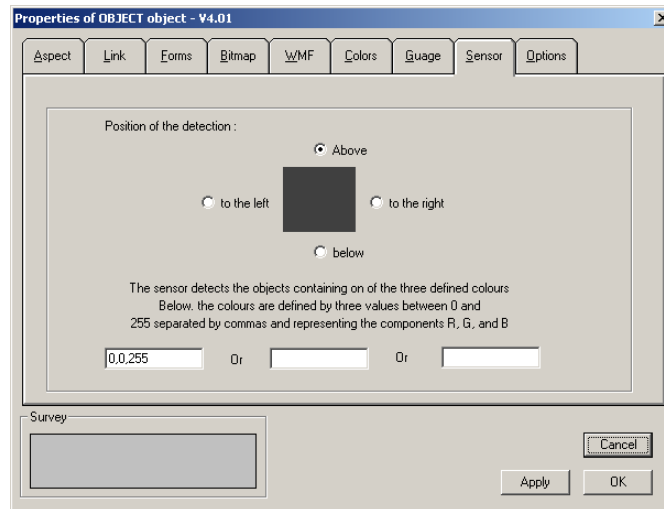
An intermediate position needs to be added on the jack. We are going to use two supplementary objects for this: a piece attached to the jack shaft which will activate a sensor and a sensor.

To connect the piece activating the sensor to the jack shaft, the jack shaft needs to be associated to an identifier: in the « Identifier » section of the « Links » tab write « 100 ». To connect the piece to the shaft, in the « Horizontal movement, Position» section of the « Links » tab write: « SISTERPARAM(100,STATE) ». This connects the piece with the jack shaft state.

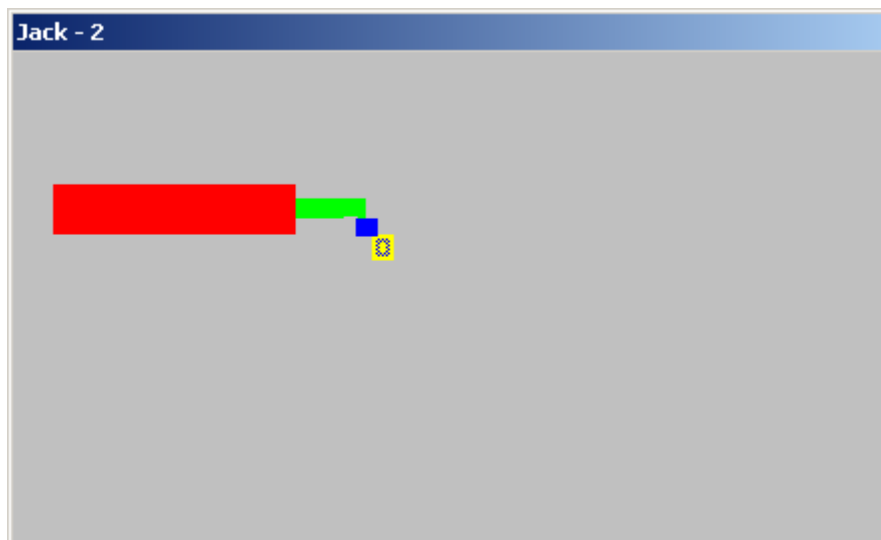
The object used as a sensor is set with parameters as follows:



Properties of OBJECT object - V4.01									
Link									
Links									
Action									
Object clicked	Object not clicked								
Link permanently with(identifier)		0							
Aspect / Colour / Filling									
Position	+ Position	- Position	Capt.min						
Mini	Maxi	Speed	ms	Capt. max					
0	200	5							
Identifier		0							
Horizontal transfer (x axis)									
Position	+ Position	- Position	Capt.min						
Mini	Maxi	Speed	ms	Capt. max					
0	200	5							
Vertical transfer (y axis)									
Position	+ Position	- Position	Capt.min						
Mini	Maxi	Speed	ms	Capt. max					
0	200	5							
Sensor		12							
Survey									
<div style="border: 1px solid black; width: 100px; height: 20px;"></div>									
<div style="text-align: right;"> <span>Cancel</span> <span>Apply</span> <span>OK</span> </div>									



The result is as follows:

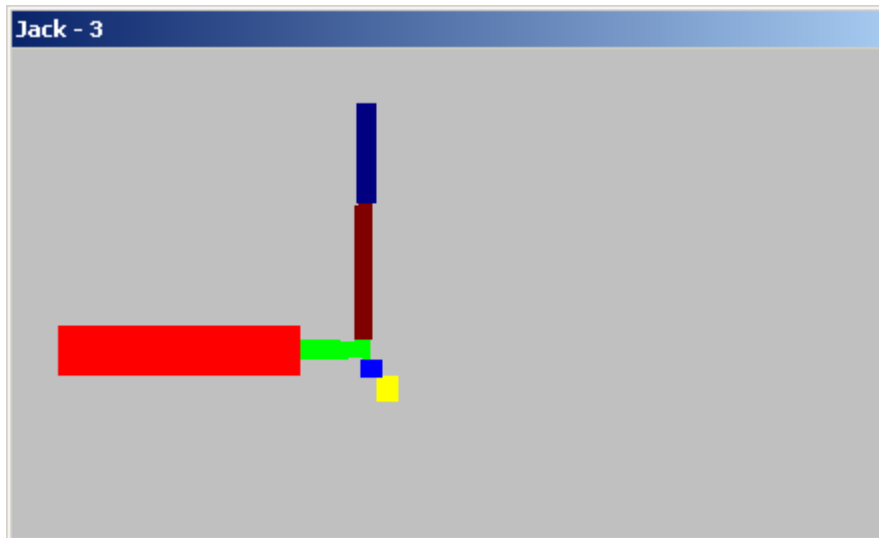


 « Examples\Process Simulation\2D\tutorial2.agn »

Second variation:

A vertical jack attached to the horizontal jack shaft is added. This jack is activated by one output (O2=1 to extract the jack, O2=0 to retract it). Two limits are associated to i3 and i4.

The result is as follows:



« Examples\Process Simulation\2D\tutorial3.agn »

Two OBJECT objects are added: one for the body of the jack and one for the shaft.

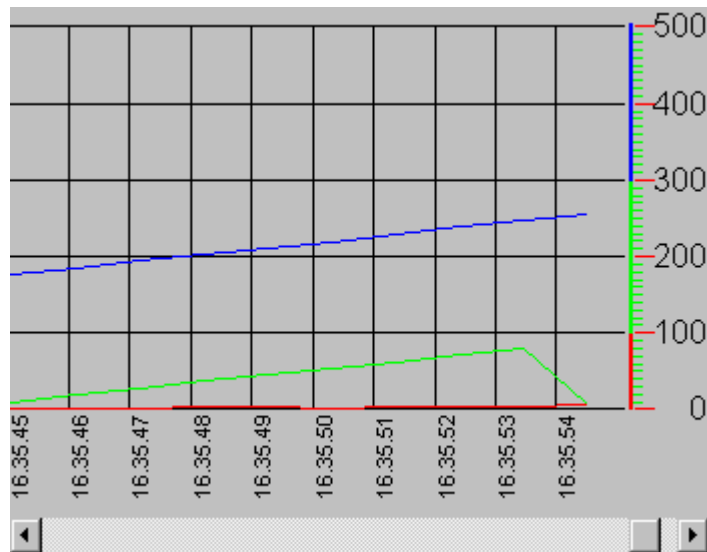
### Example of using the «ARCHIVE» object

Instructions:

- archive the state of 3 words of the processing application (m31 to m33) every second.
- the state of 4 words will be displayed on a graph left on display for 10 seconds of acquisition.
- 1000 values will be memorized in the object.
- the acquisitions will be archived in a text format « data.txt » file.

Solution:





« Examples\IRIS2D\archiving »

## Example of using the «PROG » object

Instructions:

- pressing on a pushbutton must cause the inversion of the output states O0 to O99.

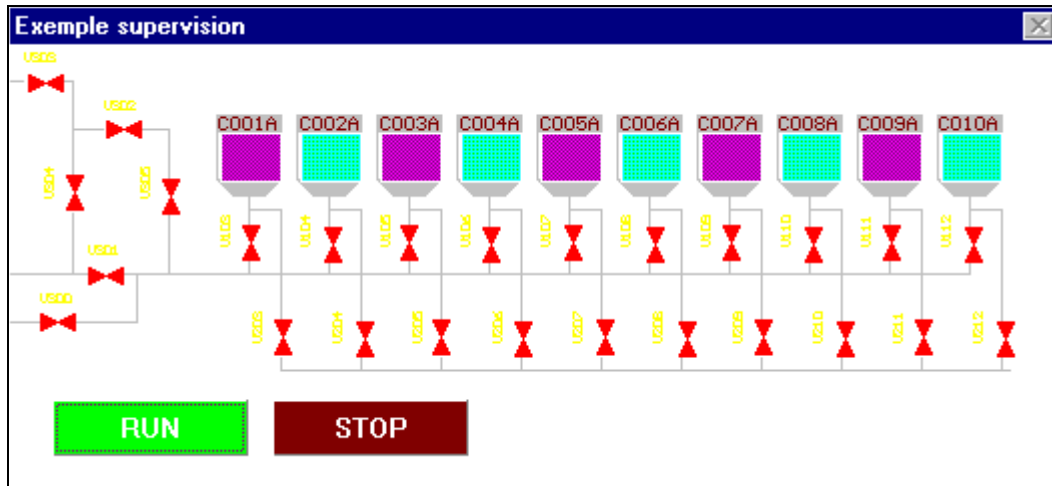
Solution:

« Examples\IRIS2D\program.agn »

## Examples of supervision application 1

The following example illustrates the creation of a supervision application. The supervision application displays the state of gates and the level of tanks. The user's actions on the gates will invert the gate state (open or closed). The RUN/STOP state of the application will also be displayed and two pushbuttons will be used to go from RUN to STOP.

The result is as follows:



« Examples\IRIS2D\supervision 1 »

OBJECT objects will be used to represent the gates. A bitmap file is created to represent the gates: open state (green) and closed state (red):



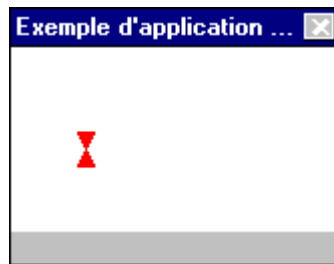
## Examples of supervision application 2

This example illustrates the use of a more evolved OBJECT object. The application displays the state of a gate which can be:

- gate open (commanded opening and open gate sensor true): green,
- gate closed (commanded close and closed gate sensor true): red,
- gate opening in progress (commanded opening and open gate sensor false): blue,
- gate closing in progress (commanded closing and closed gate sensor false): purple.

The user can invert the gate state by clicking on it.

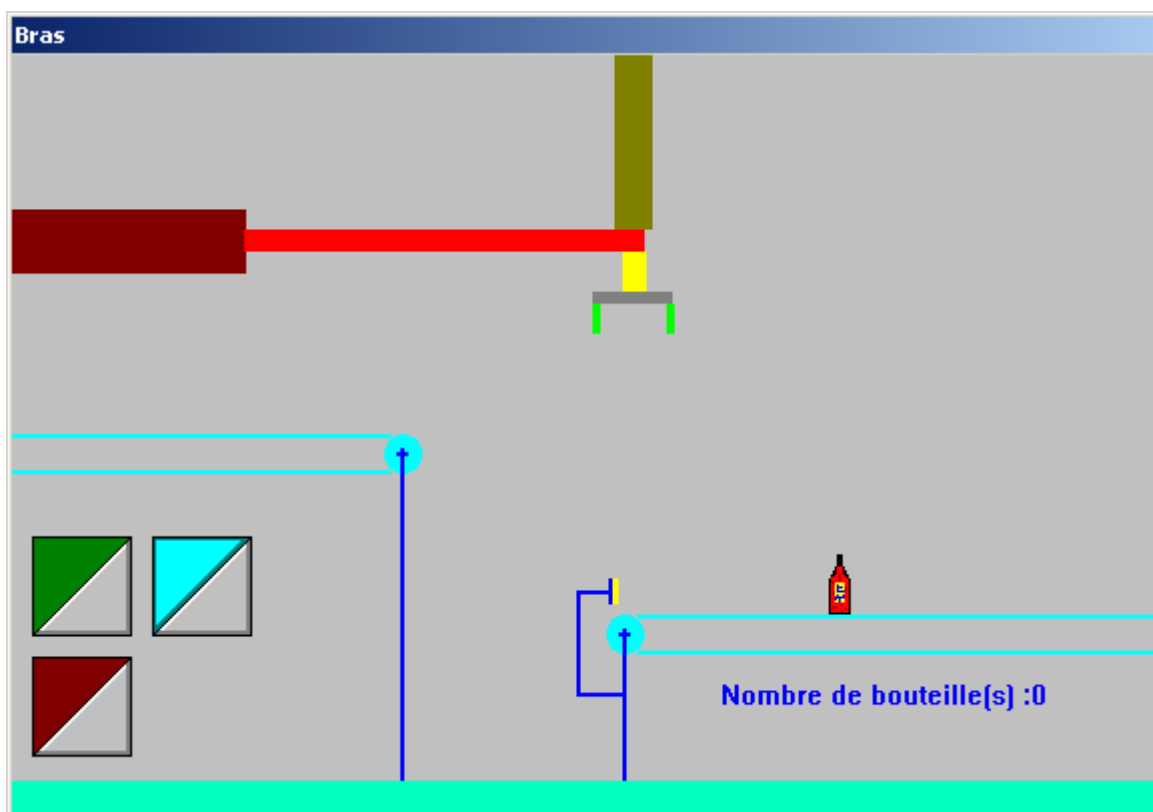
The processing application manages the gate state.



« Examples\IRIS2D\supervision 2.agn »

## Example of operating part simulation 1

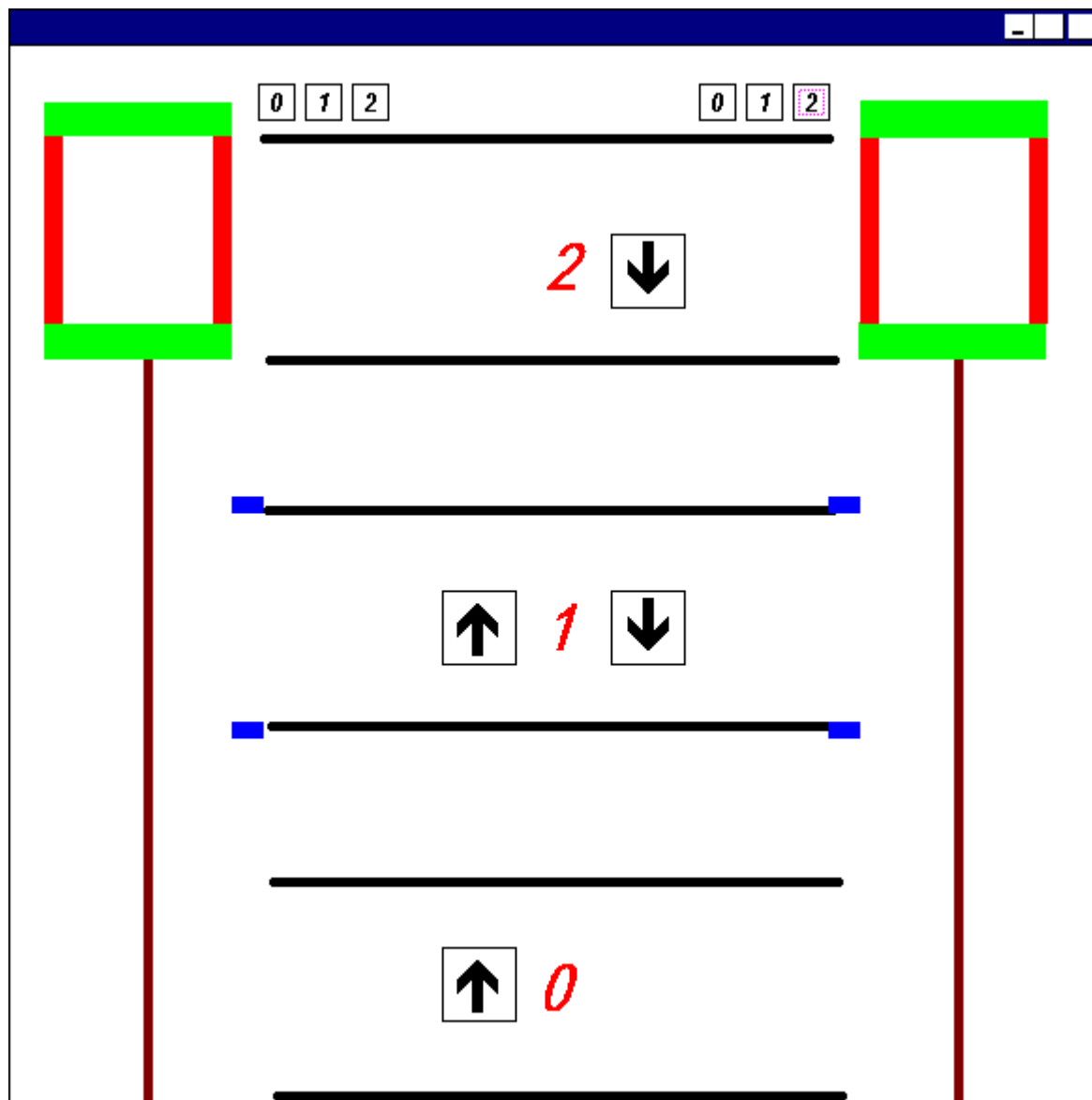
Simulation of a manipulator arm



« Examples\Process Simulation\2D\manipulator arm.agn »

## Example of operating part simulation 2

Simulation of an elevator



« Examples\Process Simulation\2D\elevator.agn »

IRIS 2D objects are used to create supervision and simulation applications of 2D operating parts.

## IRIS 3D references

IRIS 3D allows you to create simulation applications for 3D operational units. The TOKAMAK engine is integrated to IRIS3D to enable a realistic physical simulation: gravity, interactions between objects.

IRIS 3D is used to animate 3D objects using standard model makers: 3D STUDIO, SOLIDWORKS, SOLIDCONCEPTER, etc ...

The native format of the files processed by IRIS 3D is « .X » files set by Microsoft's DIRECTX 8.

A « .3DS » to « .X » converter is integrated into the environment.

The CROSSROADS program provided on the AUTOSIM installation CD-ROM or downloaded from [www.smctraining.com](http://www.smctraining.com) is used to convert a significant number of 3D files to « .3DS » format.

IRIS 3D is in a window format enclosed in the IRIS 2D console. 3D objects are animated on the console.

Each 3D file represents an object in IRIS 3D. The elements in an operating part must have their own movement and must be represented by separated files. For example, for a jack composed of a body and a shaft, files must be created for the jack body and for the jack shaft.

To create animation of objects in a 3D world, one or more behaviors can be applied to each of the objects. A behavior is composed of an object modification (moving, changing color etc.) and a link with the processing application variables to condition this modification. For example: extract the jack shaft if the output for the processing application is true.

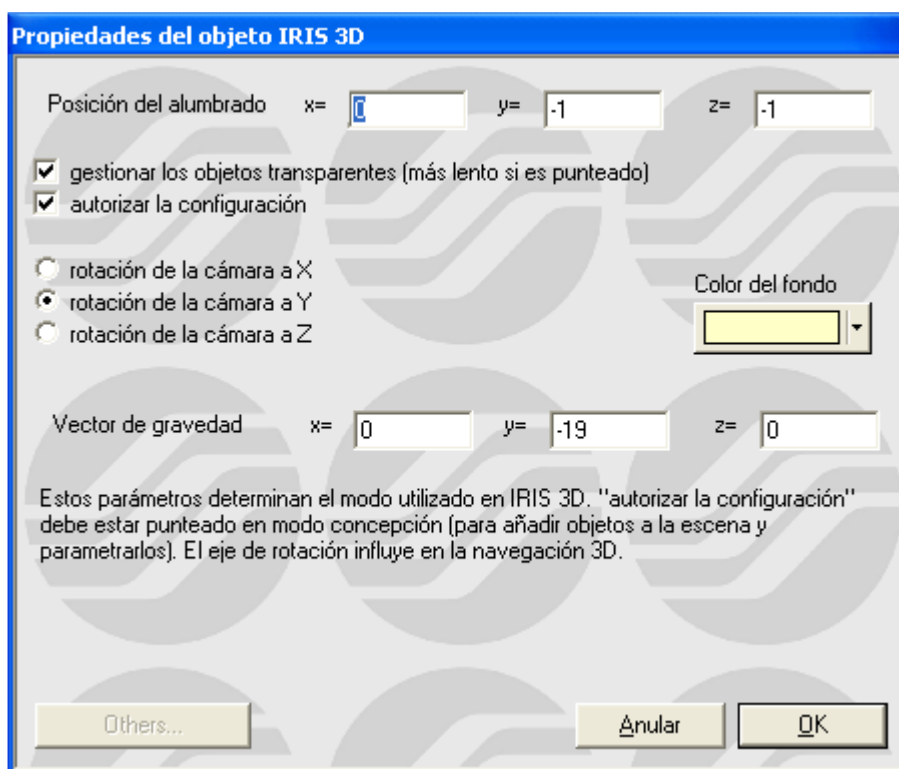
## Tutorial

The “examples\process simulation\3d\tutorial 2” sub-directory in the AUTOSIM installation directory has a WORD file that contains a tutorial devoted to creating 3D operational units.

The list of objects is shown in the list. The objects linked to an object are shown as sub-elements if the “Tree structure display” checkbox is checked.

## Creating an IRIS 3D console

With the right side of the mouse click on the « Iris » element on the browser and then select « Add an IRIS 3D console ».



**Propiedades del objeto IRIS 3D**

Posición del alumbrado x= 0 y= -1 z= -1

☒ gestionar los objetos transparentes (más lento si es punteado)

☒ autorizar la configuración

☐ rotación de la cámara a X

☒ rotación de la cámara a Y

☐ rotación de la cámara a Z

Color del fondo  

Vector de gravedad x= 0 y= -19 z= 0

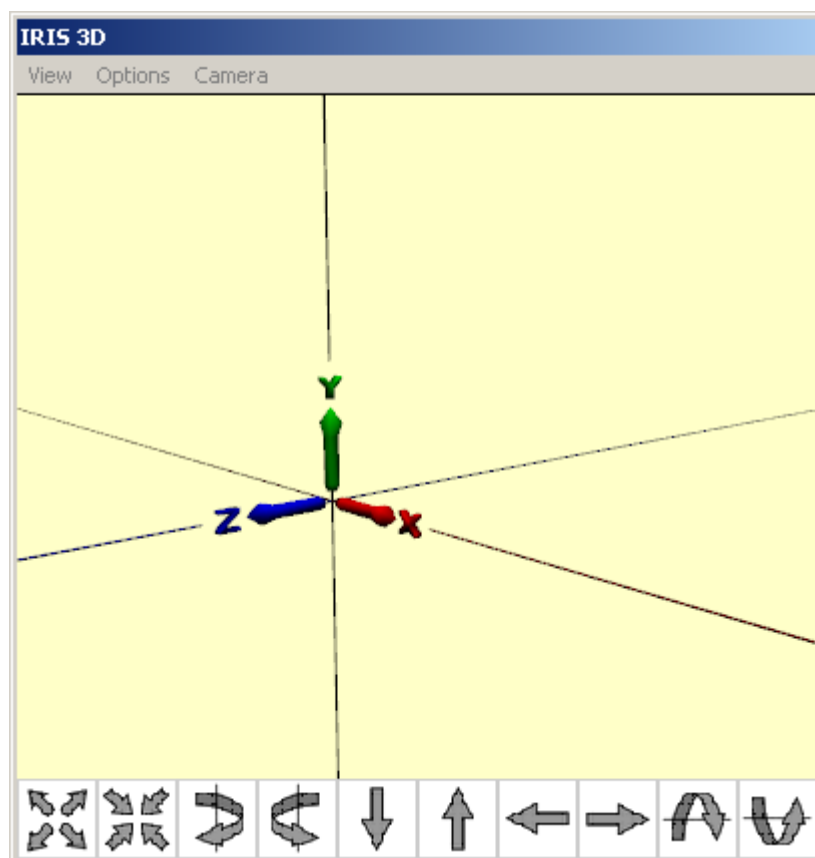
Estos parámetros determinan el modo utilizado en IRIS 3D. "autorizar la configuración" debe estar punteado en modo concepción (para añadir objetos a la escena y parametrarlos). El eje de rotación influye en la navegación 3D.

Others... Anular OK

*Creating an IRIS 3D console*

## Adding 3D files to the project

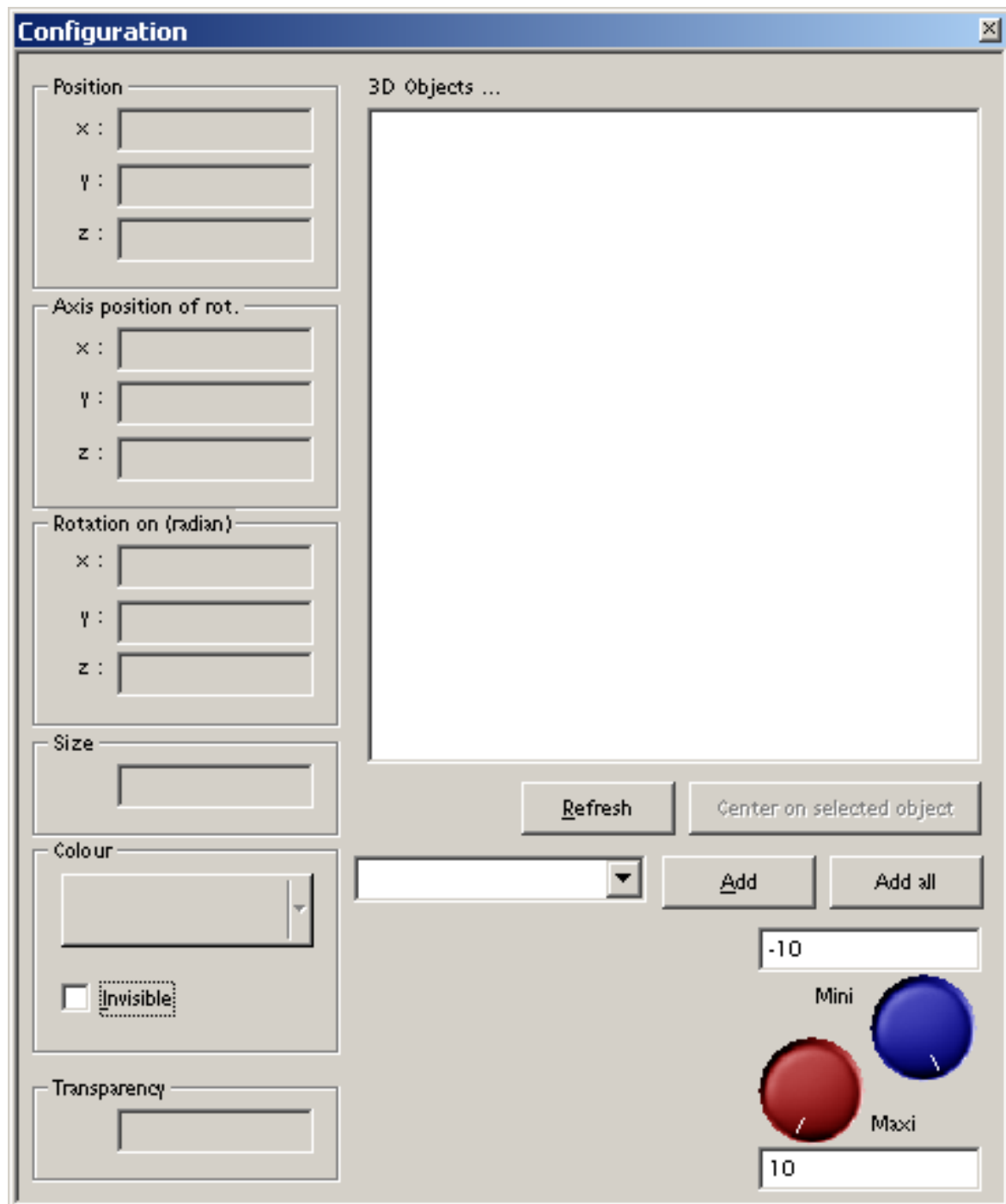
With the right side of the mouse click on the « Resources » element on the browser and select « Import one or more 3D files » from the menu. Select one or more « .3DS » files. (if your files are not in « .3DS » format, use « CROSSROAD » to convert them).



*The IRIS 3D console*

## Configuring the objects


Select « Open the configuration window » from the « Options » menu on the IRIS 3D window.

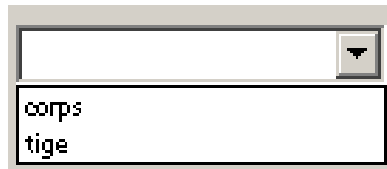


*The IRIS 3D configuration window*

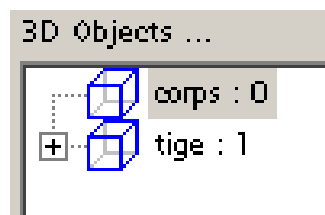


## Adding objects to the 3D world

By clicking on the  element you access the list of 3D objects present in the resources. For example:



By selecting an object on that list and clicking on « Add » you add the selected object to the 3D world. By clicking on « Add all » you add all the objects on the list to the 3D world. The objects you have added will appear on the list in the configuration window.



## Removing a 3D file from the resources

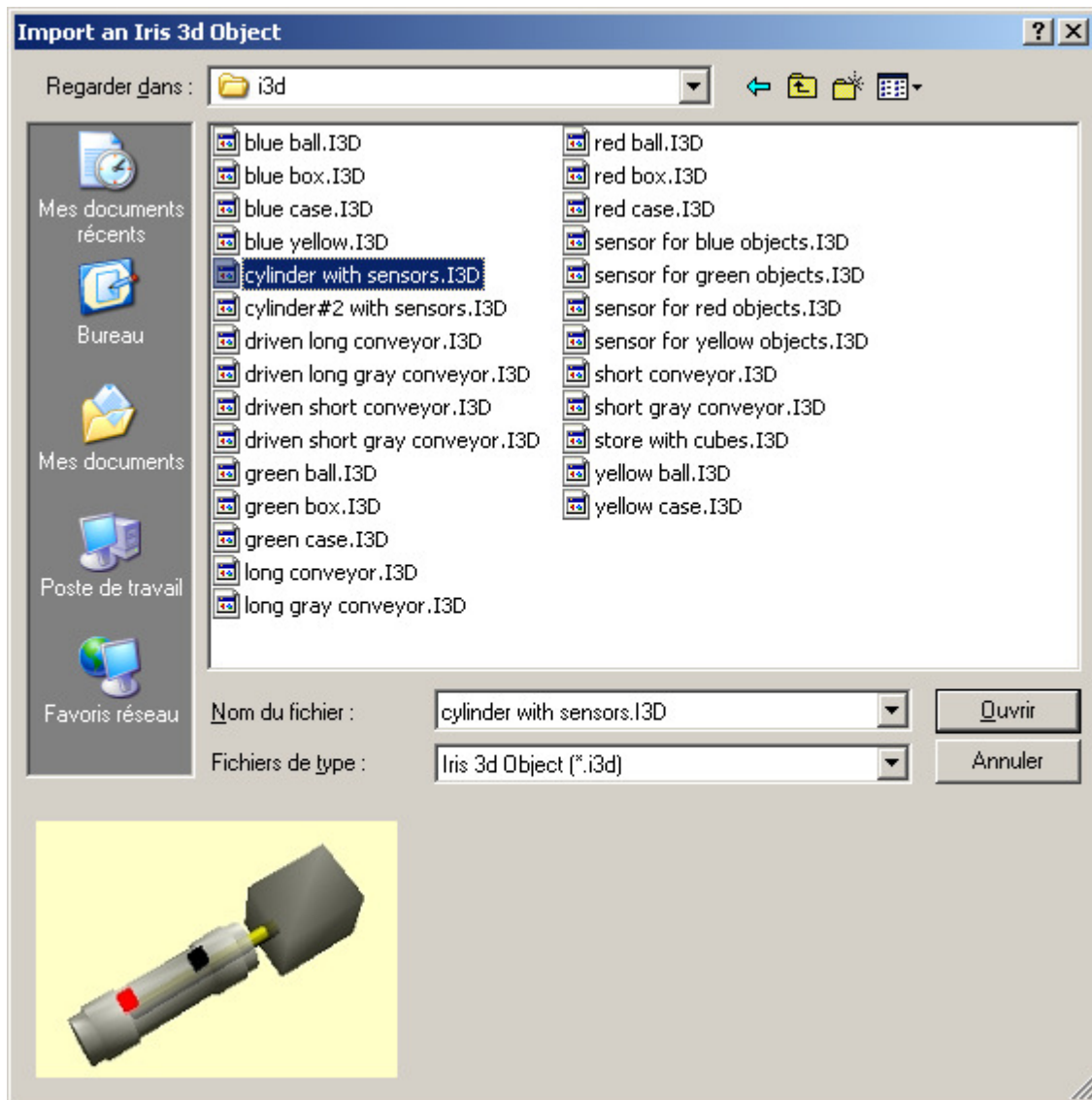
With the right side of the mouse click the 3D file on the browser and select « Delete ». The object needs to be deleted from the 3D world.

## Removing an object from a 3D world

Click with the right button of the mouse on the object in the IRIS 3D configuration window and select « Delete from the menu. »

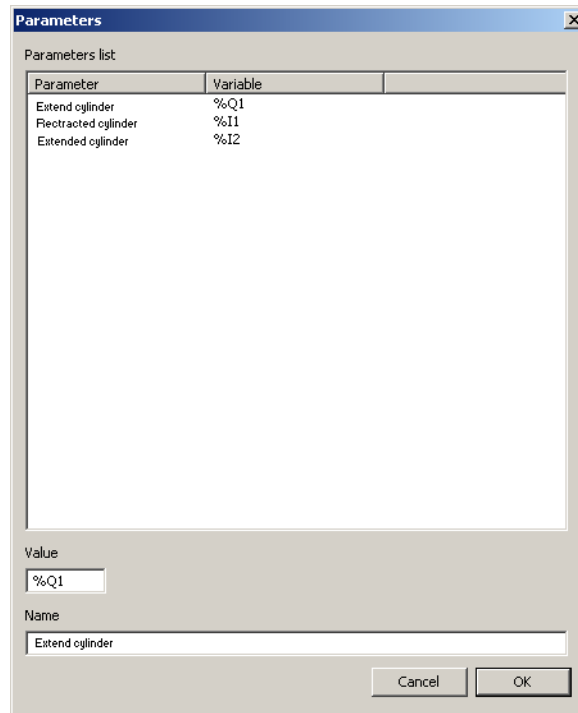
## Importing an “enhanced” object

Click on the “Import” button. A browser allows you to select the object to be imported.



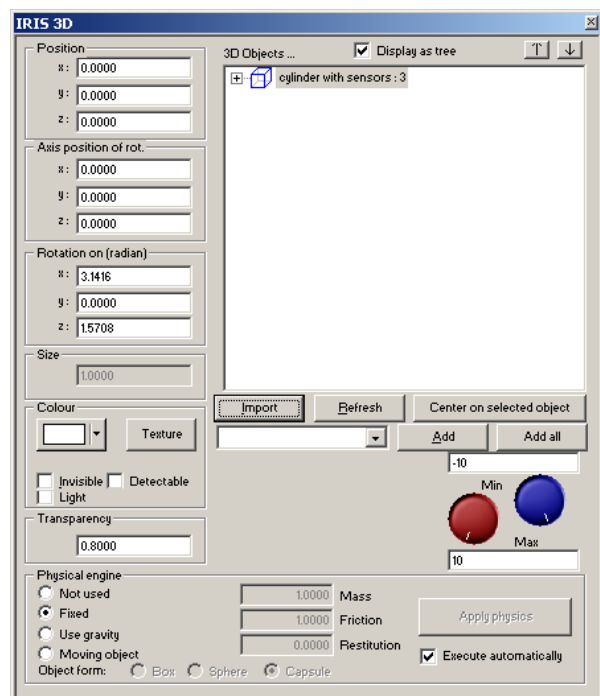
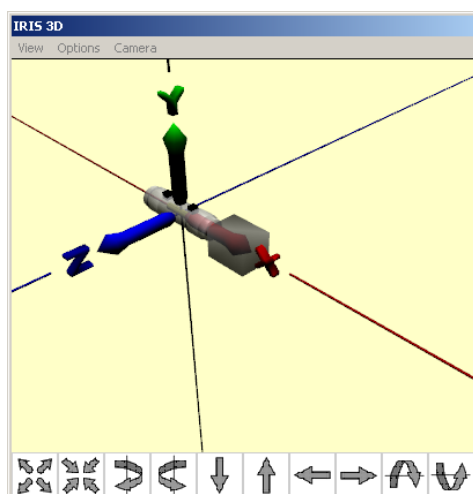
*The browser for selecting “enhanced” objects*

Once the object has been selected, click on “Open”. A parameter window then allows you to define the variables that will be associated with the object.



The window for defining the object's parameters

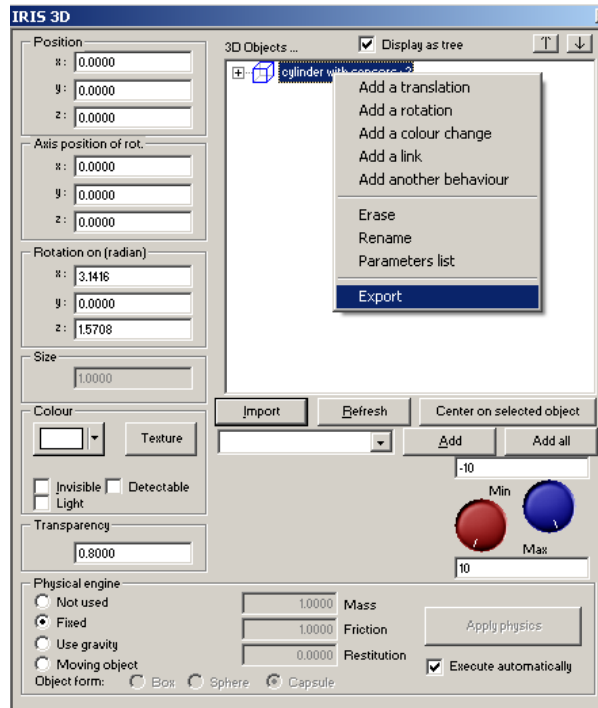
In this example (for the cylinder), the cylinder's piloting variable and the two ends of stroke are to be parameterized. The object is then shown in the 3D world and in the list of objects.



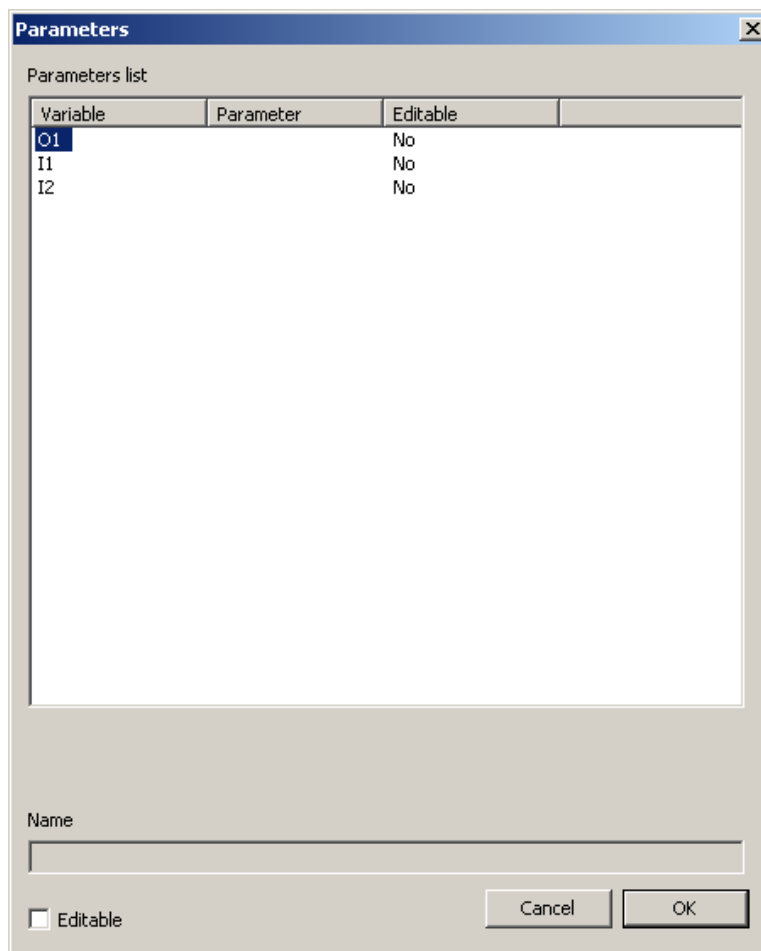
The object's position and orientation can be modified.

## Exporting an “Enhanced” object

To export an object, right-click with the mouse on the object and select “Export”. The linked objects and all of the behaviors are saved.

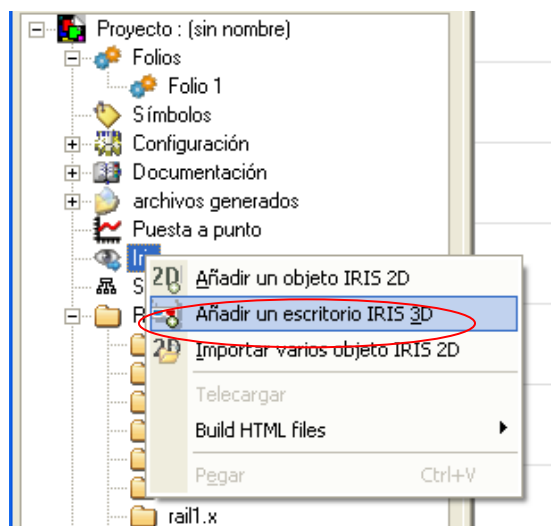


After entering a name for the file, a dialogue box allows you to assign a name to each variable used in the behaviors and to define whether this parameter can be modified or not when it is re-read.



## Example of creating a 3D simulation based on enhanced objects

Let's create a simulation in a couple of clicks for an operational unit: a part destacker.



**Propiedades del objeto IRIS 3D**

Posición del alumbrado x=  y=  z=

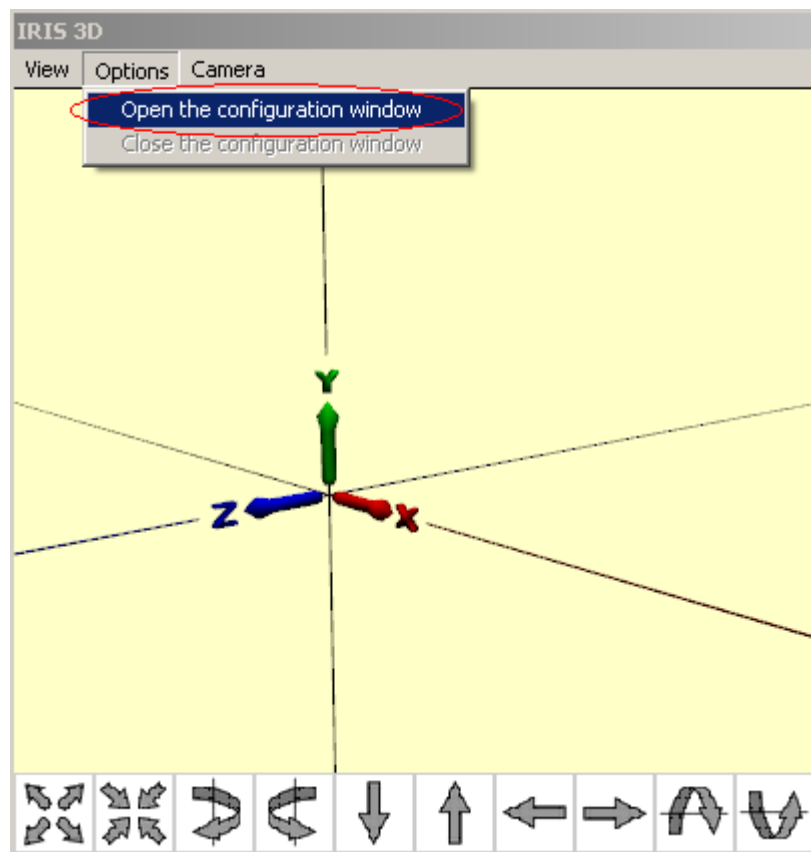
☒ gestionar los objetos transparentes (más lento si es punteado)  
☒ autorizar la configuración

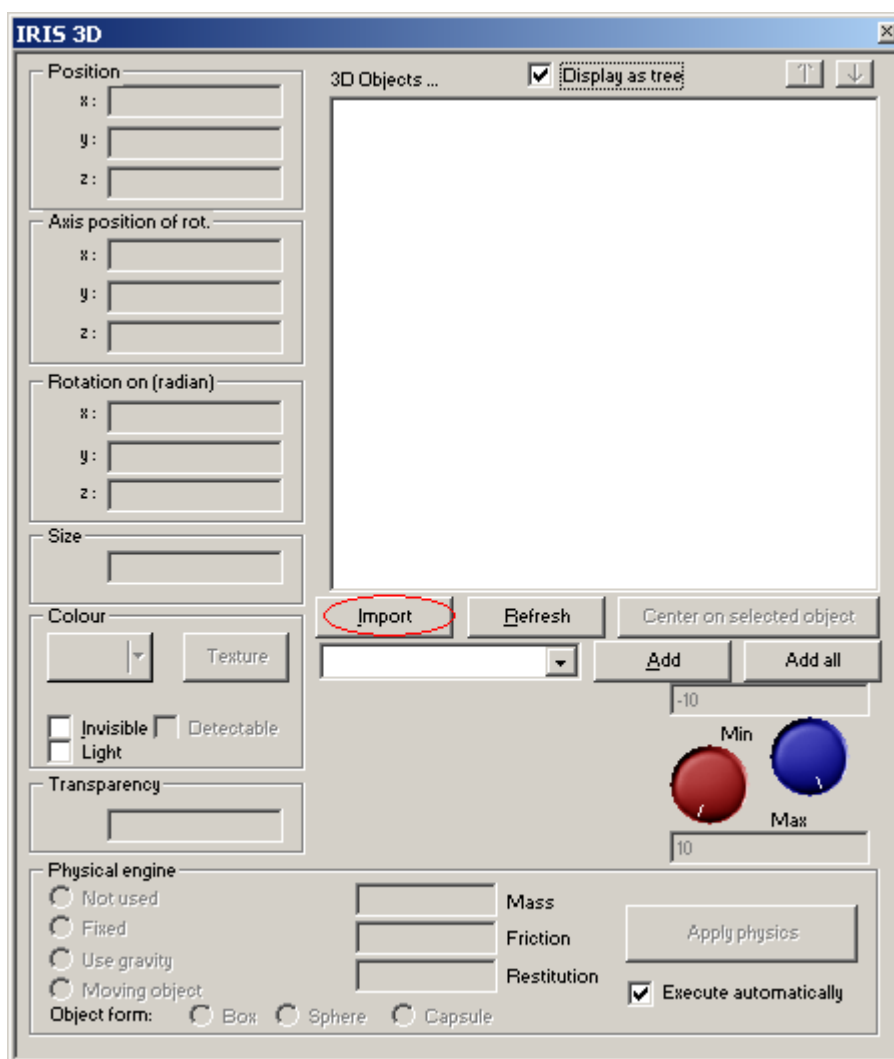
☐ rotación de la cámara a X  
☒ rotación de la cámara a Y  
☐ rotación de la cámara a Z

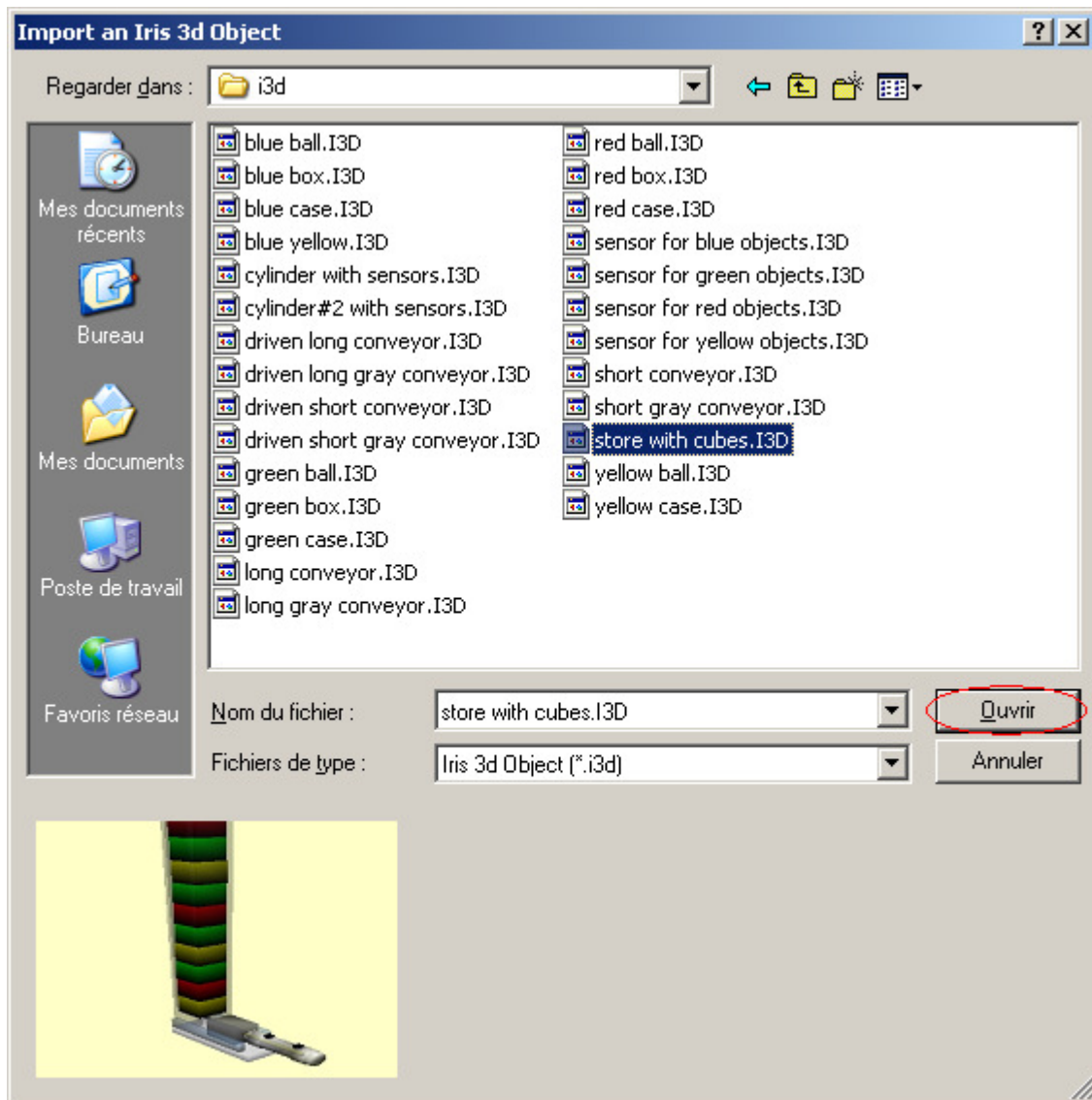
Color del fondo

Vector de gravedad x=  y=  z=

Estos parámetros determinan el modo utilizado en IRIS 3D. "autorizar la configuración" debe estar punteado en modo concepción (para añadir objetos a la escena y parametrizarlos). El eje de rotación influye en la navegación 3D.

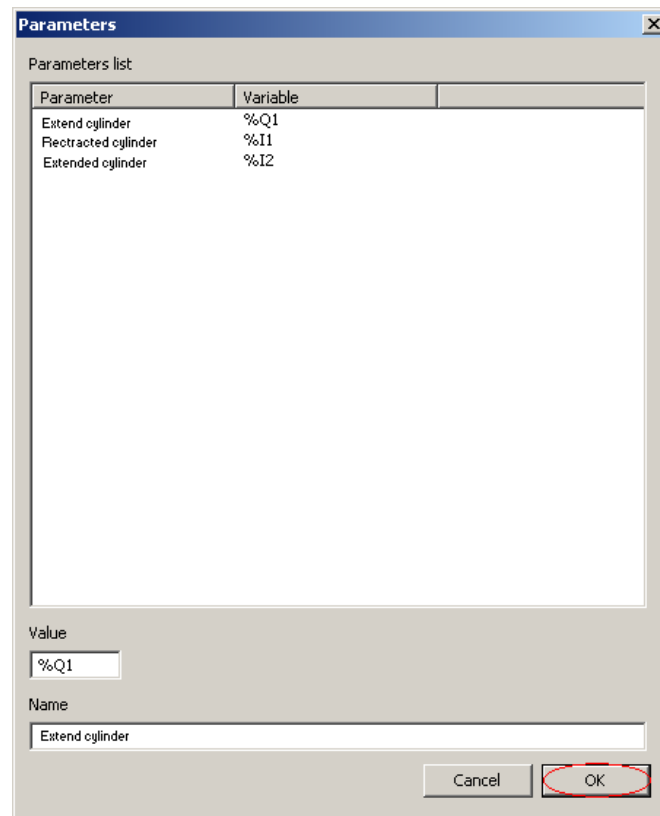




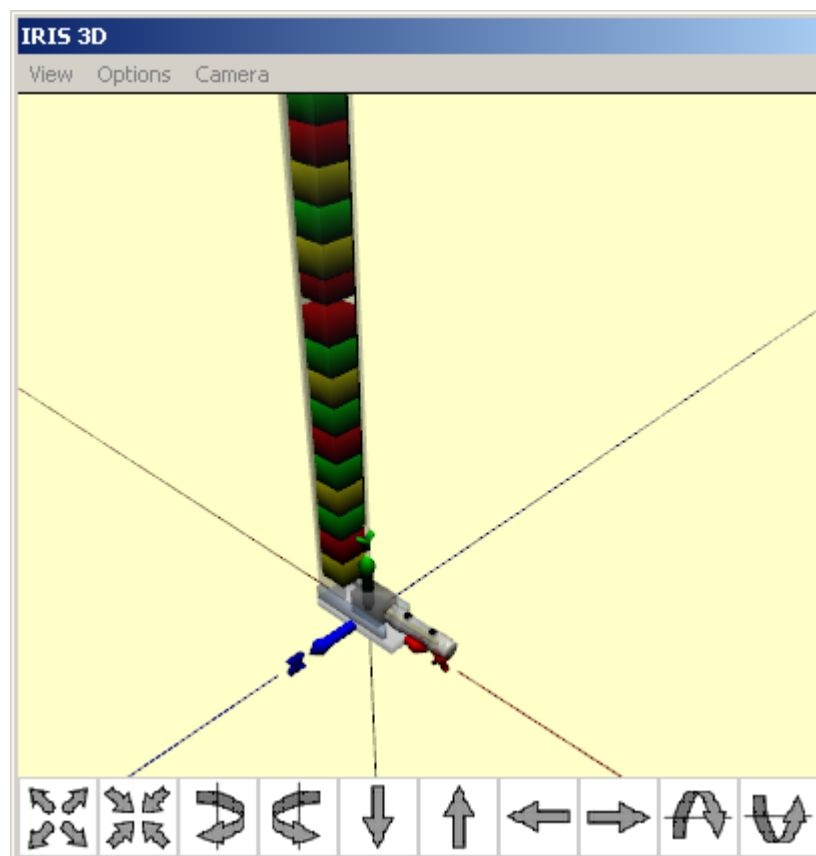


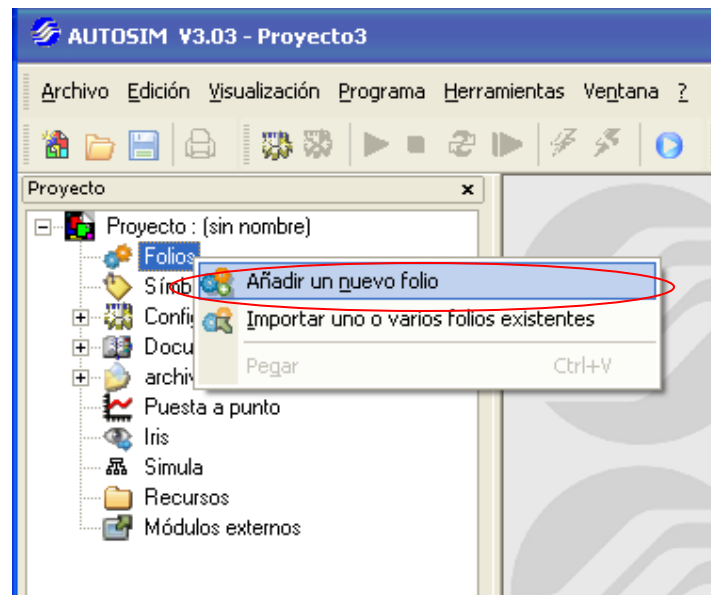
The pre-defined objects are located in the “i3d” sub-directory of the AUTOSIM installation directory.





The object is shown in IRIS3D:





**Crear un nuevo folio**

Nombre (un nombre genérico), deje el nombre predeterminado o ingrese un nombre significativo.

Folio 2

Tamaño (las dimensiones de la superficie del folio). En forma predeterminada, XXL permite crear folios muy grandes (recomendado). Para crear un Gemma elija "Gemma".

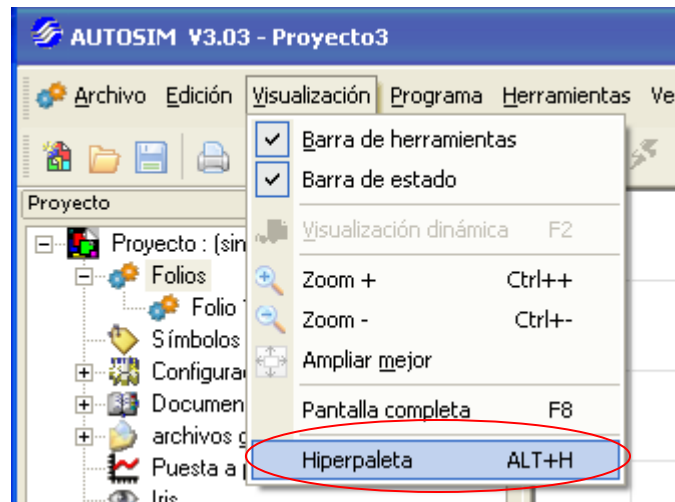
XXL (folios demasiado grandes)

Comentarios (por ejemplo, la última modificación, el autor, etc ...).

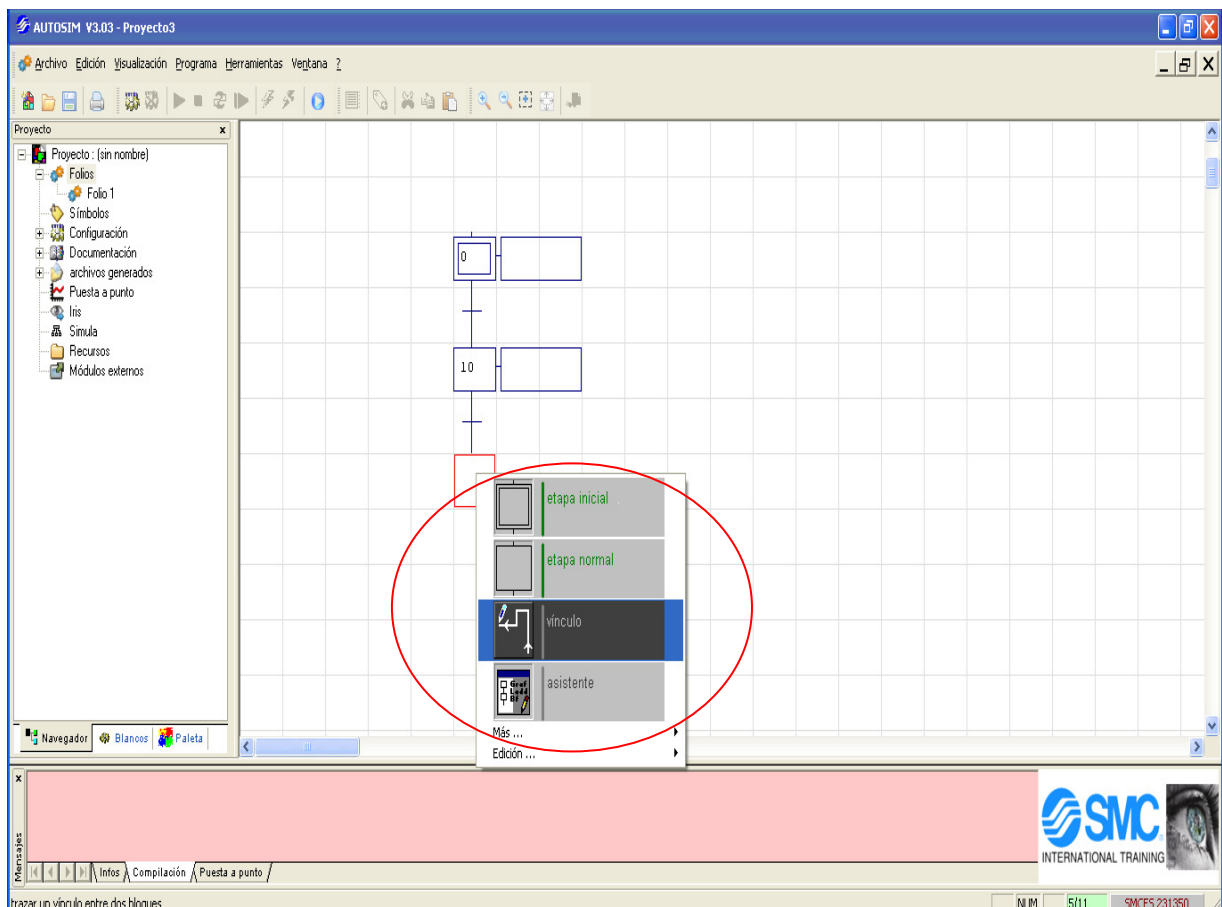
Creado el 29/12/2011.  
Modificado el 29/12/2011.

Defina las características del nuevo folio, su tamaño, su nombre y eventuales comentarios. Todo esto podrá modificarse posteriormente.

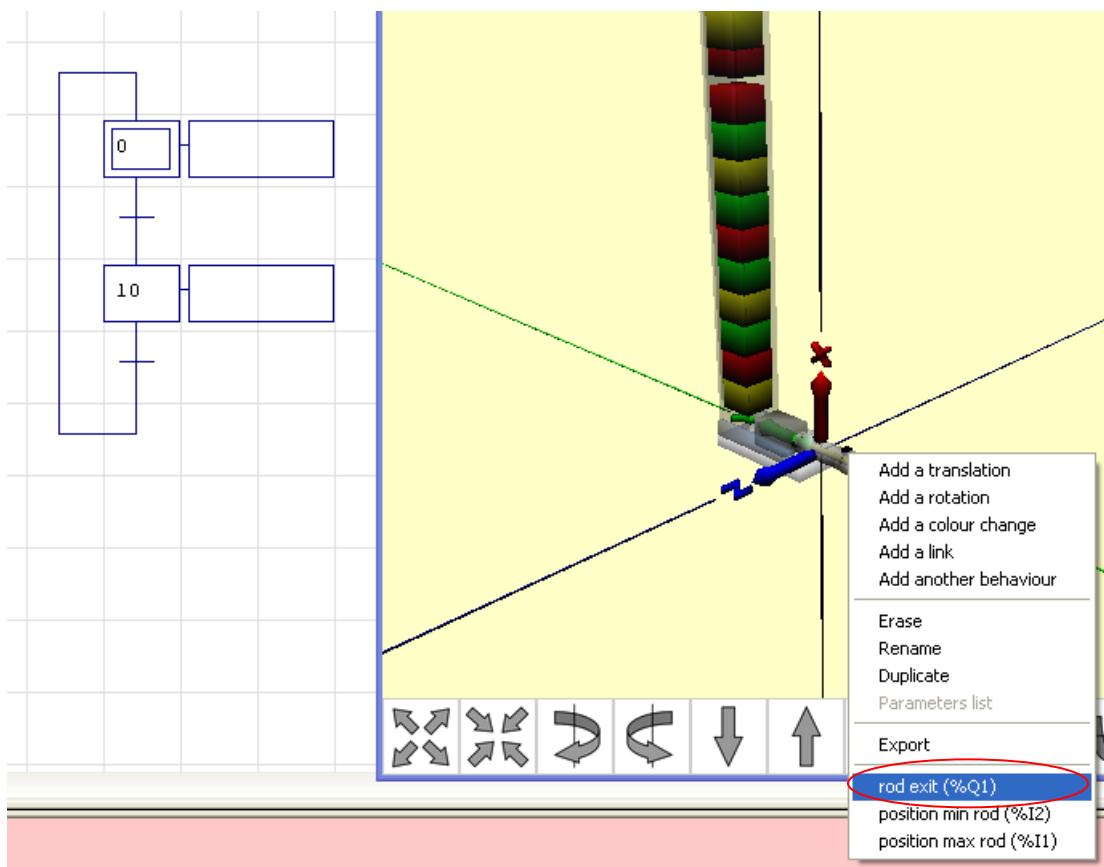
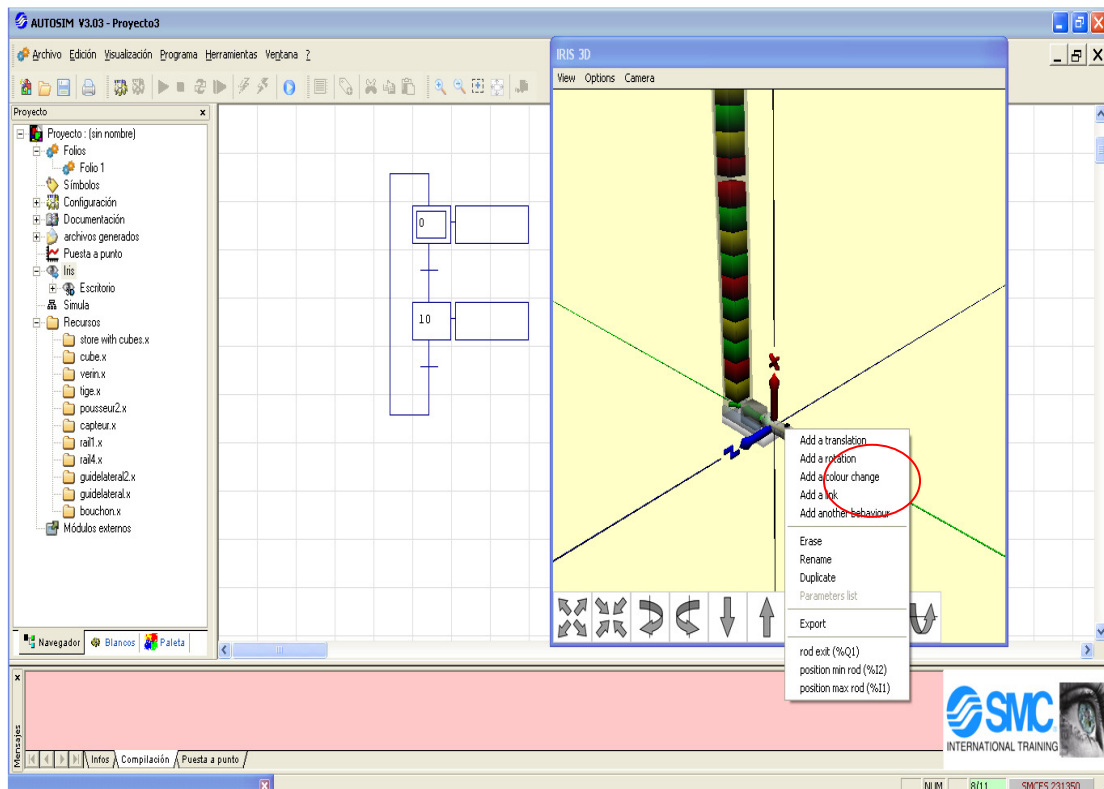
**OK**  
**Anular**



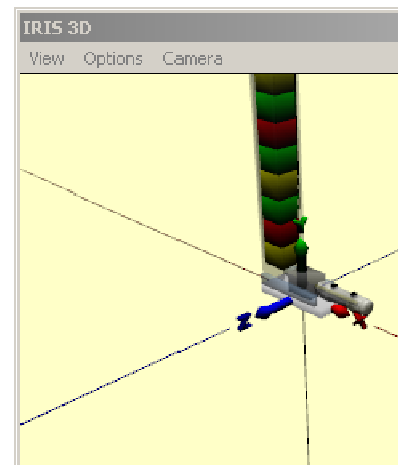
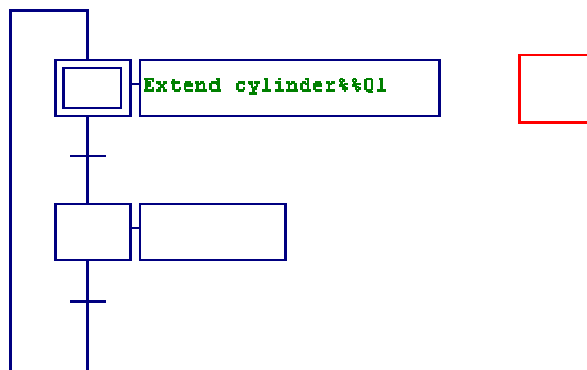
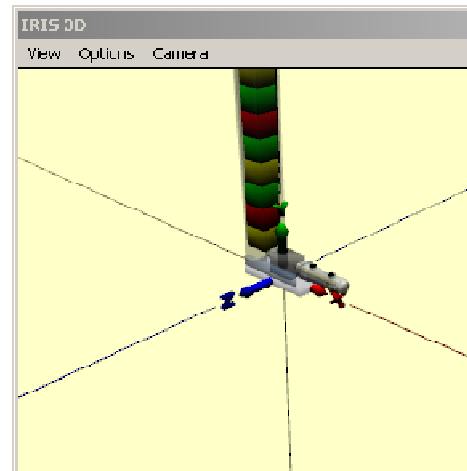
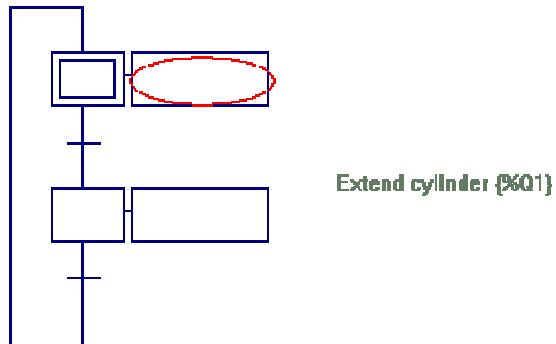
With the palette, design a Grafcet with two steps. A right click on the sheet lets you access the link drawing application to loop back the Grafcet.



A right click on the cube store allows you to access the list of variables.

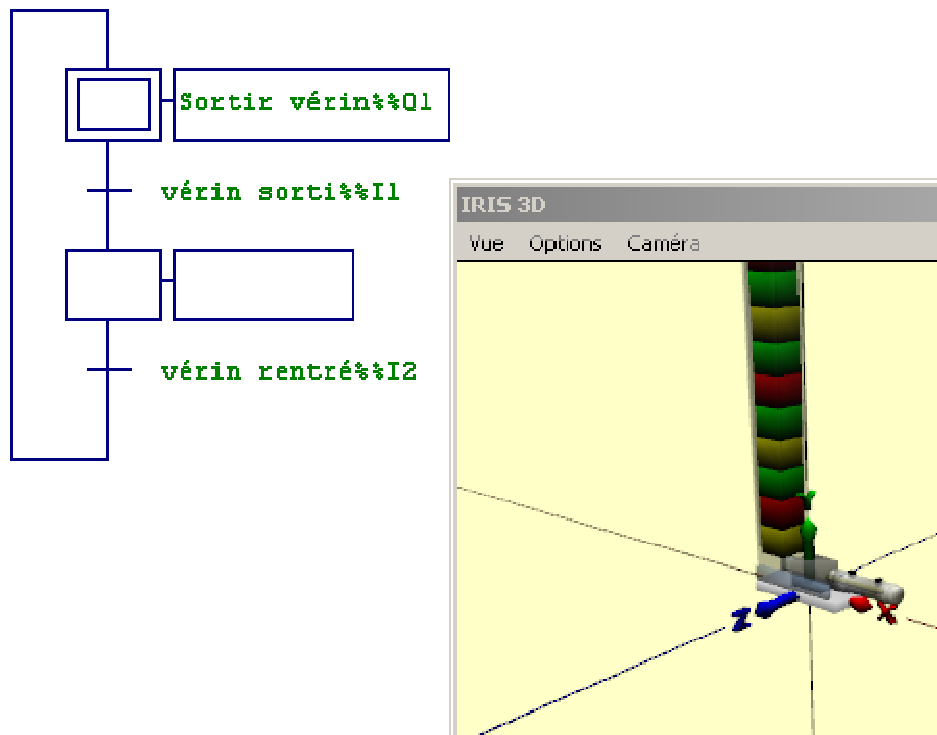


Move the cursor over the action rectangle and left-click.



Repeat this operation to place the “cylinder extended” element on the first transition and “cylinder retracted” on the second.

This is the final result:



You can now click on the “GO” button in the tool bar to launch the application.

This complete example is in the “examples\Process simulation\3D\physical engine” sub-directory with the name “destacker.agn”.

## Applying a behavior to an object

Click with the right button of the mouse on the object in the IRIS 3D configuration window and select « Add ... » from the menu. ».

## Name of AUTOSIM variables

The name of AUTOSIM variables used in the behaviors are limited to the following syntaxes:

## Access to boolean variables

On: output « n », for example O8, O10,

/On: complement of the output « n », for example /O1, /O15,

In: input « n », for example 10,14,

/In: complement of the input « n », for example /I4, /I56,

Bn: bit « n », for example B100, B200,

/Bn: complement of bit « n », for example /B800, /B100,

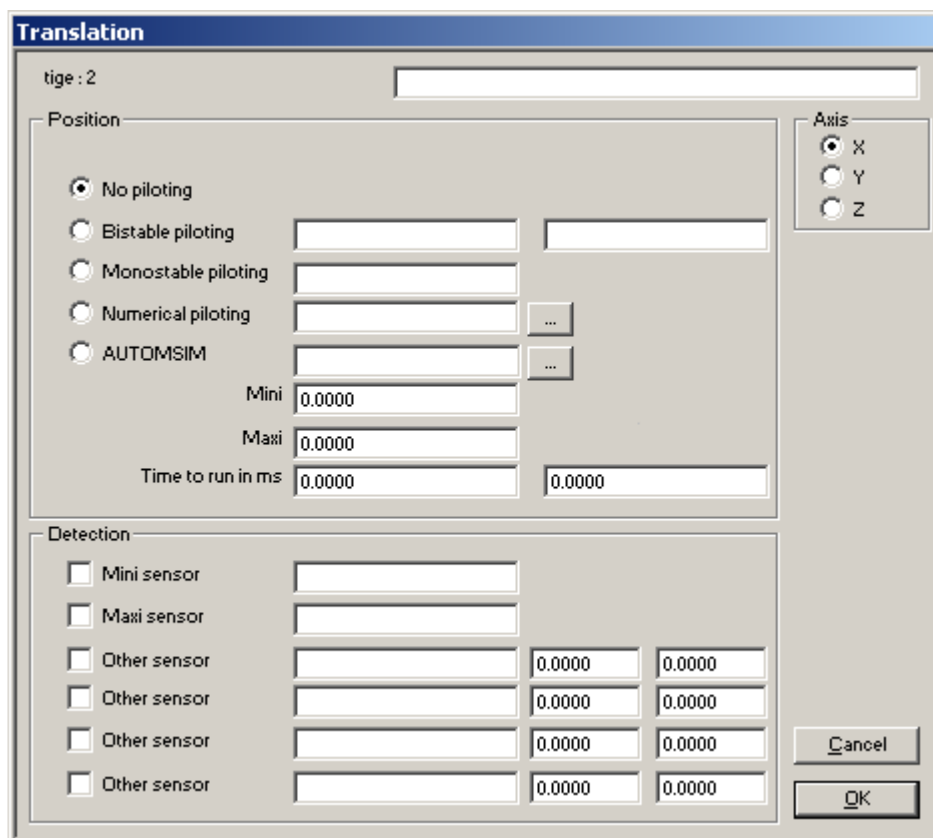
The access to bits B is limited to a table of linear bits, a command #B must be used to reserve bits (see the language manual),

Access to numeric variables

Mn: word « n », for example: M200, M300

Fn: float « n », for example: F200,F400

## Adding a translation



The dialog box is titled "Translation". It contains a text field for "tige : 2". Below this, there are two main sections: "Position" and "Detection".

**Position section:**

- Radio buttons for "No piloting" (selected), "Bistable piloting", "Monostable piloting", "Numerical piloting", and "AUTOMSIM".
- Input fields for "Mini" (0.0000) and "Maxi" (0.0000).
- Input fields for "Time to run in ms" (0.0000) and another empty field (0.0000).

**Detection section:**

- Checkboxes for "Mini sensor", "Maxi sensor", and four "Other sensor" entries.
- Input fields for each sensor, with numerical values (0.0000) for the "Other sensor" entries.

On the right side, there is an "Axis" section with radio buttons for X (selected), Y, and Z. At the bottom right, there are "Cancel" and "OK" buttons.

*Properties of a translation*

## Name

The first area is used to enter a generic name for the translation. This name appears in the list of the IRIS 3D configuration window, it is only used for comments and can be left blank.

## Axis

Establishes the dimension to be applied to the translation.

## Type

- without driving: no translation, this is used to make a translation inoperable without needing to delete it (to run tests for example)/
- bistable driving: two boolean variables: the translation is driven by two boolean variables: the first drives the translation in one direction (from min to max), the second in the other direction (from max to min).

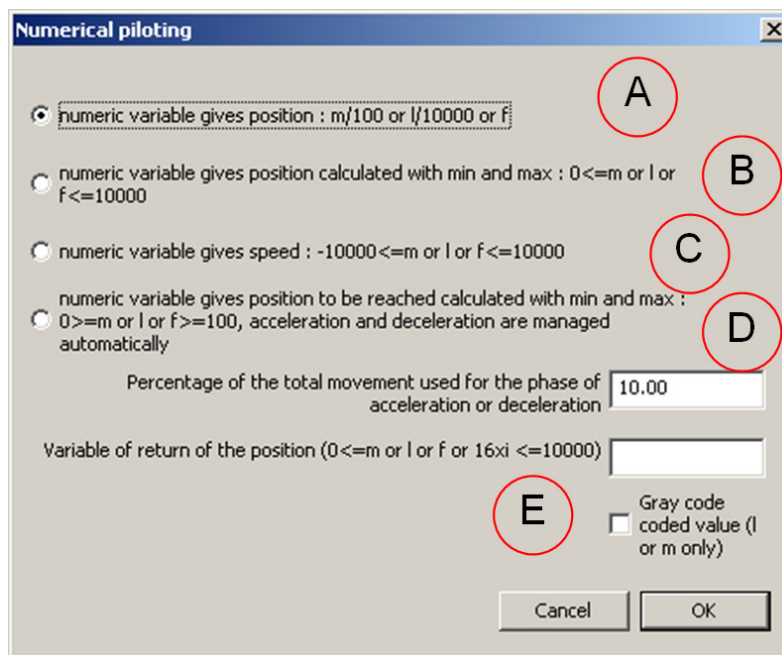
State of the first variable	State of the second variable	Object
0	0	Immobile
1	0	Translation of min to max
0	1	Translation of max to min
1	1	Immobile

- monostable driving: a boolean variable drives the translation if the variable is true

Variable state	Object
1	Translation of min to max
0	Translation of max to min

- numeric driving: the position of the object on the designated axis is equal to the specified numeric variable.
- The “...” button allows an “enhanced” mode to be defined for this type of link:





A

The content of the numeric variable defines the object's position. If it is a word the position will be set at the value divided by 100; if it is a long, it will be set at the value divided by 10000; if it is a floating-point, it will be set at the value contained in the floating-point. Min and max define the limits for these values.

B

The content of the variable defines the position between the min and max values. 0 = min position, 10000 = max position.

C

The content of the variable gives a speed of displacement ranging from -10000 to 10000.

D

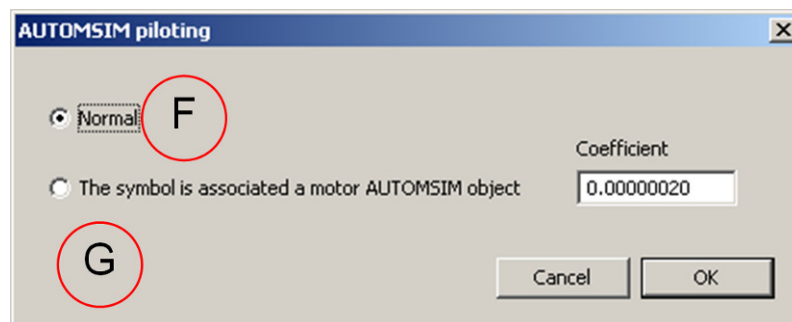
The content of the variable gives a position to be reached as a percentage of the stroke between min and max: 0 = min position, 100 = max position. The acceleration and deceleration are calculated automatically. The "percentage of movement used..." parameter defines the length of the acceleration and deceleration phases.

E

Allows a variable to be defined that will receive the position of the object constantly. The variable can be a word, a long, a floating-point or an input (in this case, this input and the next 16 inputs receive the position like an absolute encoder linked on the inputs). The “value in gray code” checkbox allows this value to be obtained like a gray encoder.

The “examples\Process simulation\3D\numerical pilotings” sub-directory contains examples of these different modes.

- SIMULA: the object’s position on the designated axis is given by the content of a variable managed by an SIMULA object. The “...” button allows an enhanced mode to be defined for this type of link:



F

The variable associated to an SIMULA object defines the position between min and max.

G

The variable associated to an SIMULA “motor” object modifies the position according to the coefficient (it makes it possible to define the relationship between the rotation speed of the SIMULA motor and the speed with which the position varies).

The “examples\Process simulation\3D\SIMULA piloting” sub-directory contains examples illustrating both these modes.

## Amplitude and origin

The « Min » and « Max » areas establish the amplitude and origin of the translation.

## Speed

The stroke time establishes the speed for going from the min point to max point (it is identical to the return speed).

## Detection

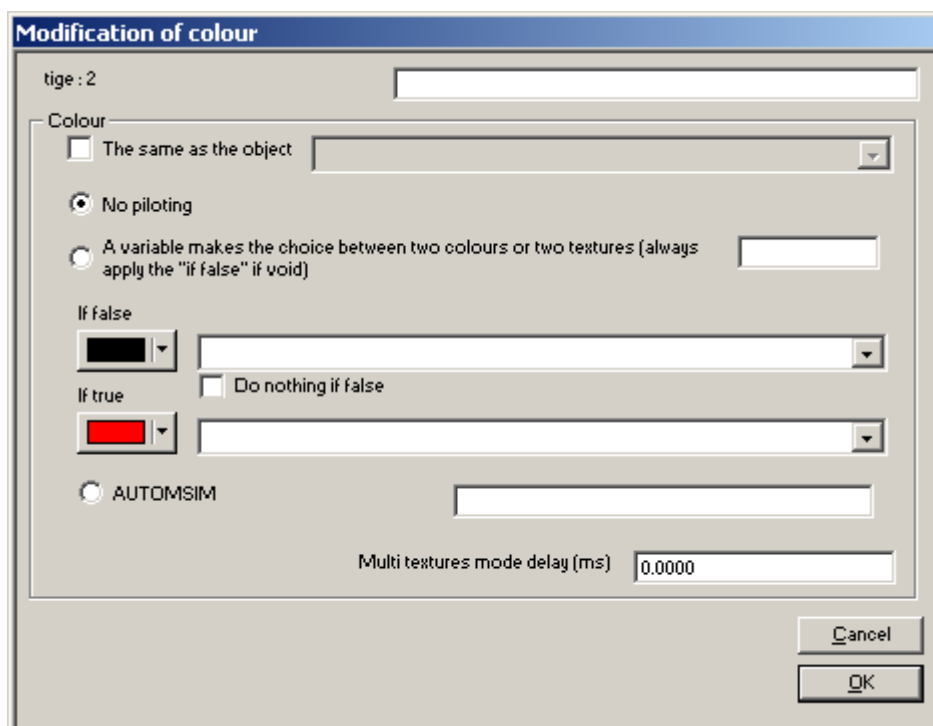
This is used to set the sensors for the translations. The min and max sensors manage the limits, the other 4 possible sensors can be used to create intermediate positions.

## Adding a rotation

The parameters are completely similar to the translations see chapter Adding a translation. The angles are expressed in radians.

The object rotation center must be set for each object in the IRIS 3D configuration window.

## Adding a color change



The dialog box titled "Modification of colour" contains the following elements:

- A text field labeled "tge : 2" with a value of 2.
- A "Colour" section with three radio buttons:
  - ☐ The same as the object (with a dropdown menu)
  - ☒ No piloting
  - ☐ A variable makes the choice between two colours or two textures (always apply the "if false" if void) (with a text field)
- An "If false" section with a color swatch (black) and a dropdown menu.
- An "If true" section with a color swatch (red) and a dropdown menu.
- A checkbox labeled "Do nothing if false".
- A radio button labeled "AUTOSIM" with a text field.
- A "Multi textures mode delay (ms)" field with a value of 0.0000.
- "Cancel" and "OK" buttons at the bottom right.

Color change

Driving of a color using a variable must refer to a boolean variable.

The “The same as object...” checkbox allows you to apply the same color as another object.

Controlling a color by a variable must reference a Boolean variable.

Color control can also be performed with an SIMULA variable (associated to an SIMULA indicator object, for example).

If the “do nothing if false” box is checked, no color is applied if the variable’s status is false. This makes it possible to associate several changes of color to a single object if more than 2 colors are needed.

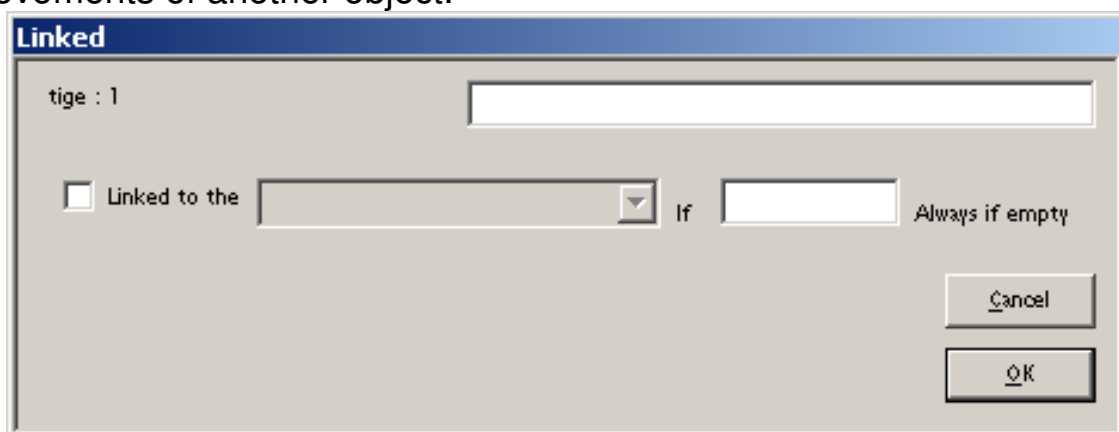
The pull-down lists allow a texture to be selected instead of a color. To have a texture shown in a pull-down list, place the (“.bmp” or “.jpg” file) in the AUTOSIM project resources.

## Multiple textures

It is possible to associate several textures that will be applied automatically. To do this, associate several “Color modification” type behaviors to a single object and document the “Time for multiple texture mode” area with the time at the end of which the texture will be applied automatically. The pre-defined “Conveyor Belt” object uses this technique.

## Adding a link

A link forces an object that this behavior is applied to, to follow the movements of another object.

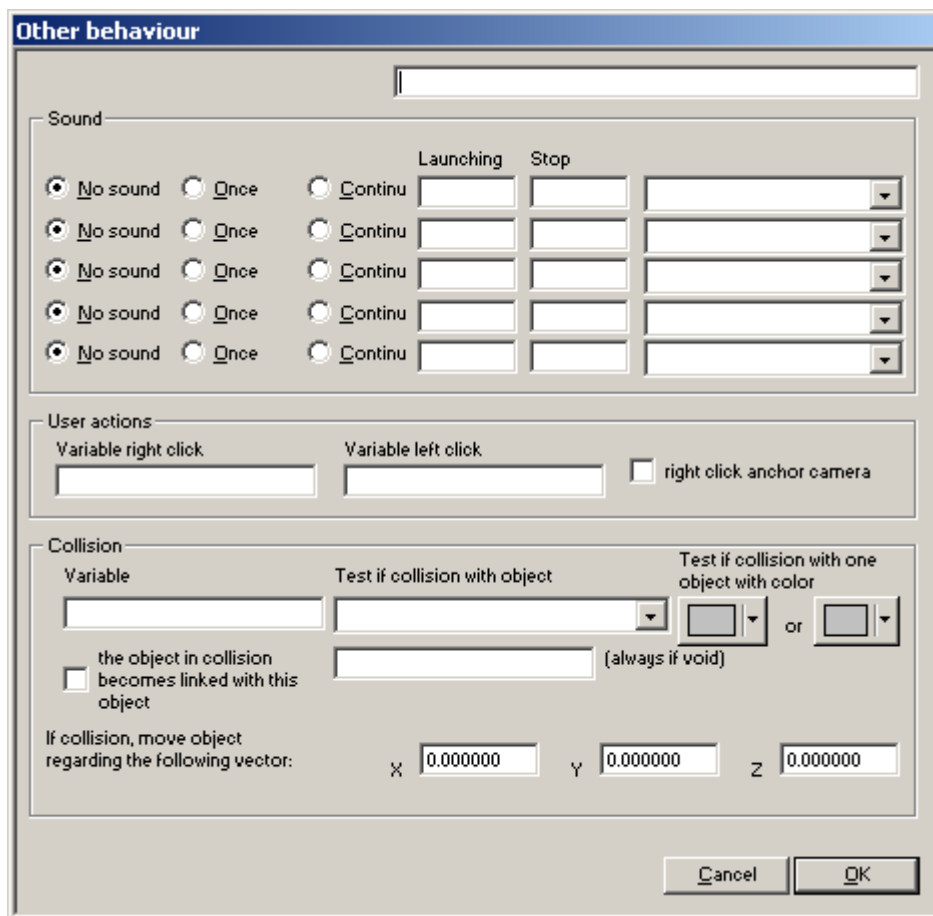


*Links between objects*

The link condition can be a boolean variable. The link is unconditional (object always linked) if the condition is left blank.

### Adding another behavior

This is used to use a sound associated to a condition, or to change a boolean variable to 1 when the user clicks with the right or left side of the mouse on the object the behavior is applied to.



Other behaviors

The elements of the “Sound” group make it possible to play a sound associated to a condition.

The elements of the “User actions” group allow a Boolean variable to be set to 1 when the user clicks with the right or left mouse button on the object that the behavior applies to. The “A right-click on the object anchors the camera” checkbox makes it possible to lock the camera (which defines the display point of view in the IRIS 3D window) on the object that the behavior applies to.

The elements of the “Collision” group make it possible to define a collision test:

- either with one object in particular,
- or with objects having a particular color (a choice of 2 colors is possible).

The “Variable” area can be documented with the name of a Boolean variable that will be set to true if the collision test is true.

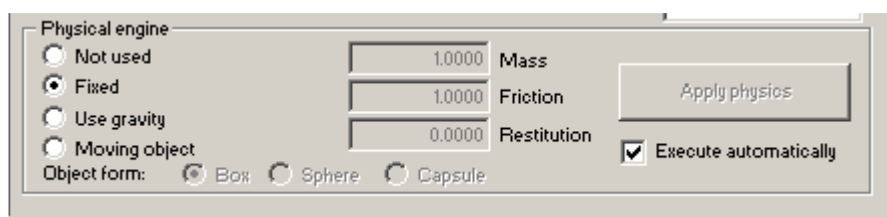
The “The object in collision becomes linked with the object if” checkbox makes it possible to link the object that comes into collision with the object to which the behavior is applied. A variable can condition this link. This technique makes it possible to easily handle the simulation of a clamp or suction cup.

The vector allows you to give a speed to an object that comes into collision with the object to which the behavior is applied. The pre-defined “Conveyor Belt” object uses this technique.

## Physical engine

The physical engine makes it possible to handle gravity and the interactions between objects so as to obtain a very realistic simulation. For objects subject to gravity, only block, sphere or capsule shapes are handled by the physical engine.

For each object you can define the type of handling used by the physical engine:



“Not used”: the object is not handled by the physical engine: it is not subject to gravity and does not interact with the other objects.

“Fixed”: an object handled by the physical engine that does not change position but which interacts with the other objects: the housing of a machine, for example.

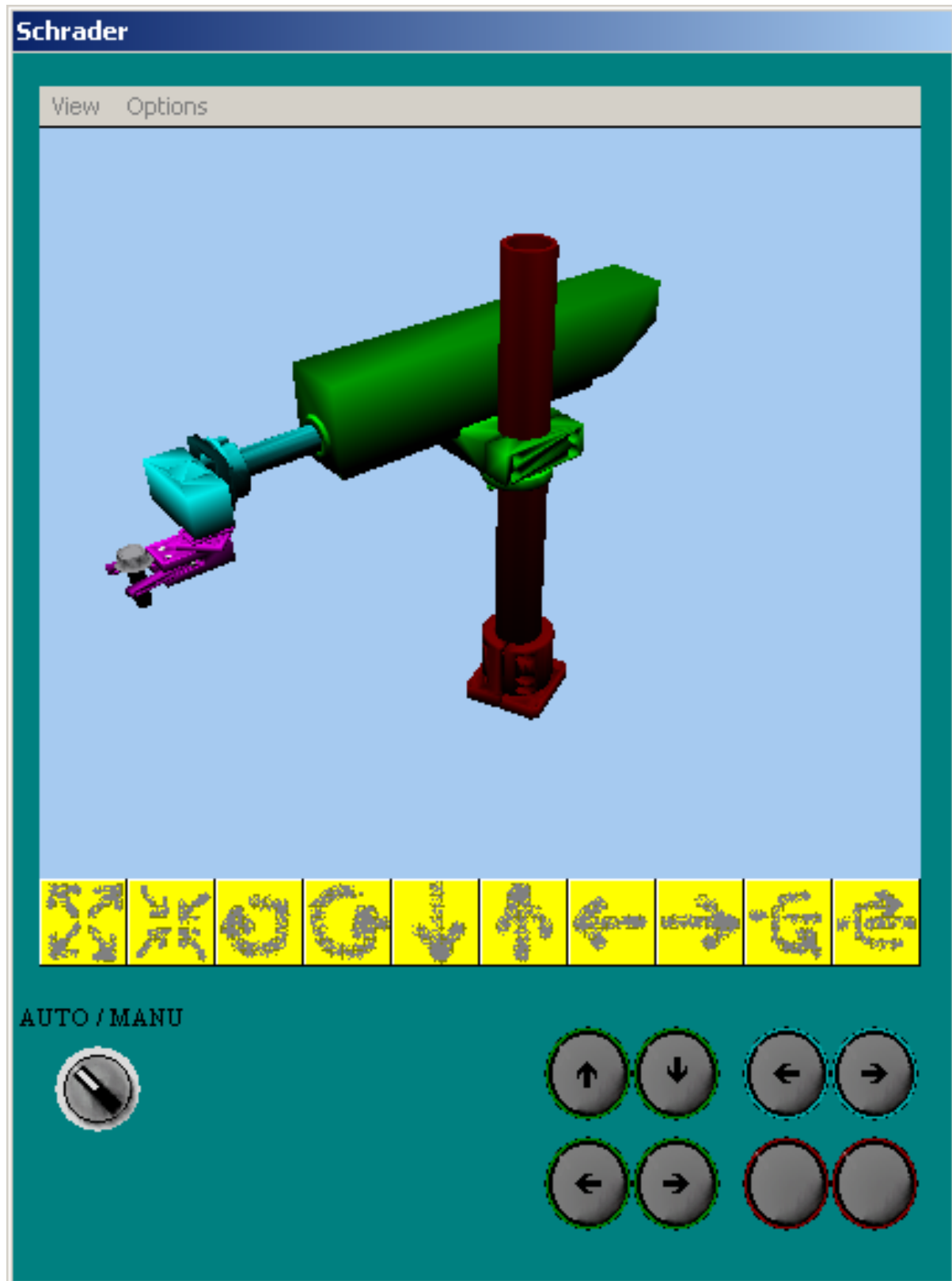
“Use gravity”: a moving object handled by the physical engine, subject to gravity and interacting with the other objects: a box moving on a conveyor belt, for example. For this type of object, the mass, coefficients of friction and restitution and the primary shape of the object (block, sphere or capsule) have to be defined.

“Moving object”: a moving object handled by the physical engine, which is not subject to gravity and which interacts with the other objects: a cylinder rod pushing objects, for example. For this type of object, the coefficients of friction and restitution and the primary shape of the object (block, sphere or capsule) have to be defined.

The “Apply physics” button allows the physical engine to be launched. The “Automatic execution” checkbox automatically launches the physical engine when the AUTOSIM PC executor is installed.

The “examples\Process simulation\3D\physical engine” sub-directory contains examples illustrating the physical engine being used.

## IRIS 3D example



« Examples\Simulation PO\3D\Scharder.agn »

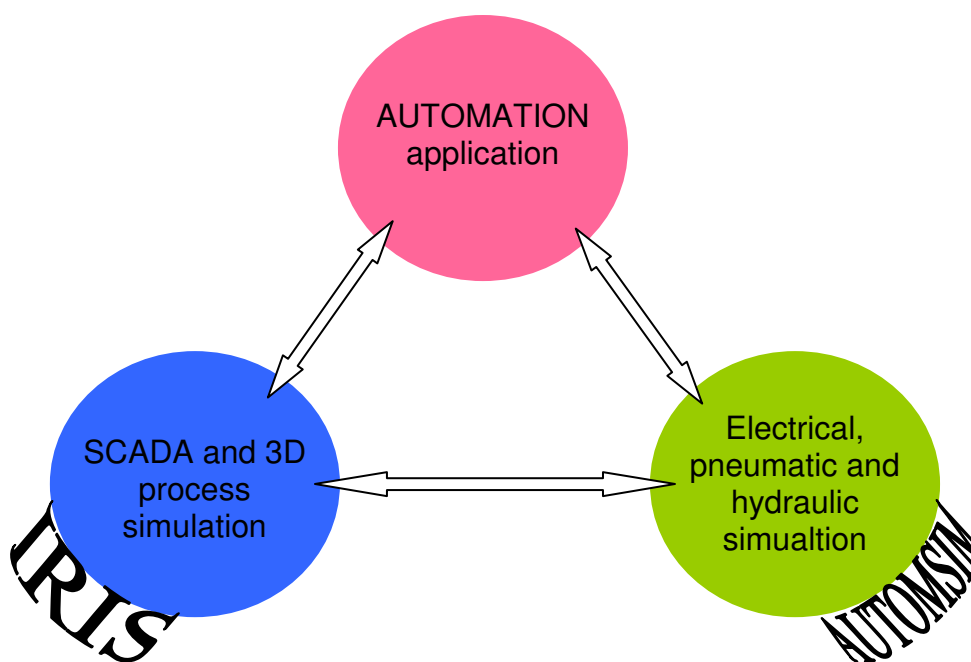


IRIS 3D is used to design simulation applications of 3D operating parts. The objects must be created in a standard model maker and imported in the AUTOSIM project resources. Behaviors are then applied to the objects to create 3D animations.

## Introduction to SIMULA

SIMULA is a pneumatic / electrical / hydraulic simulation module.

It can be used independently or in addition to the AUTOSIM<sup>3</sup> applications:

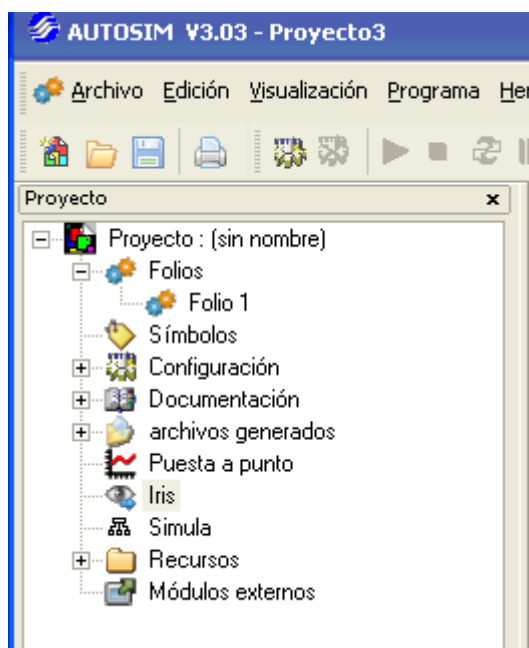


## Installation

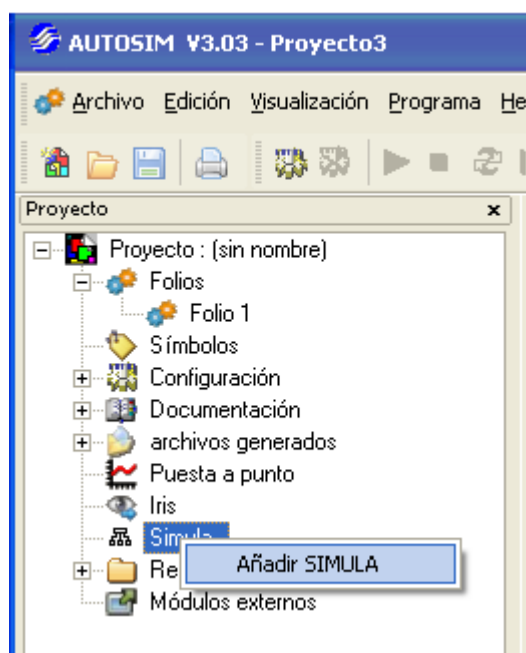
To install SIMULA, install AUTOSIM. In options, be sure that « SIMULA » is checked.

## Practical experience

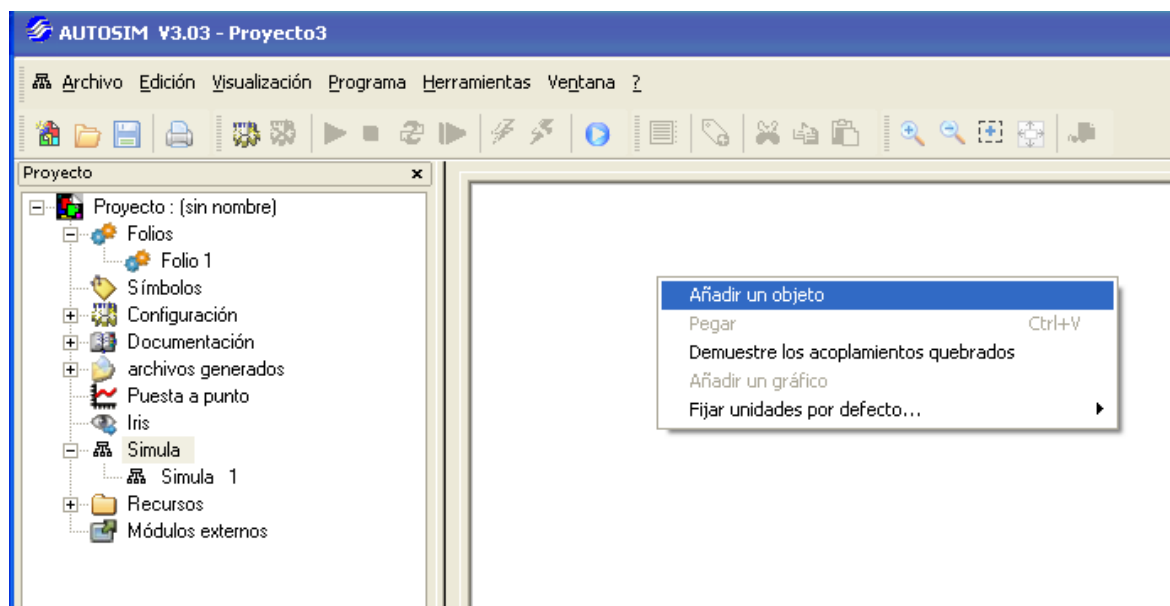
Let us do a simple example: cylinder + directional valve



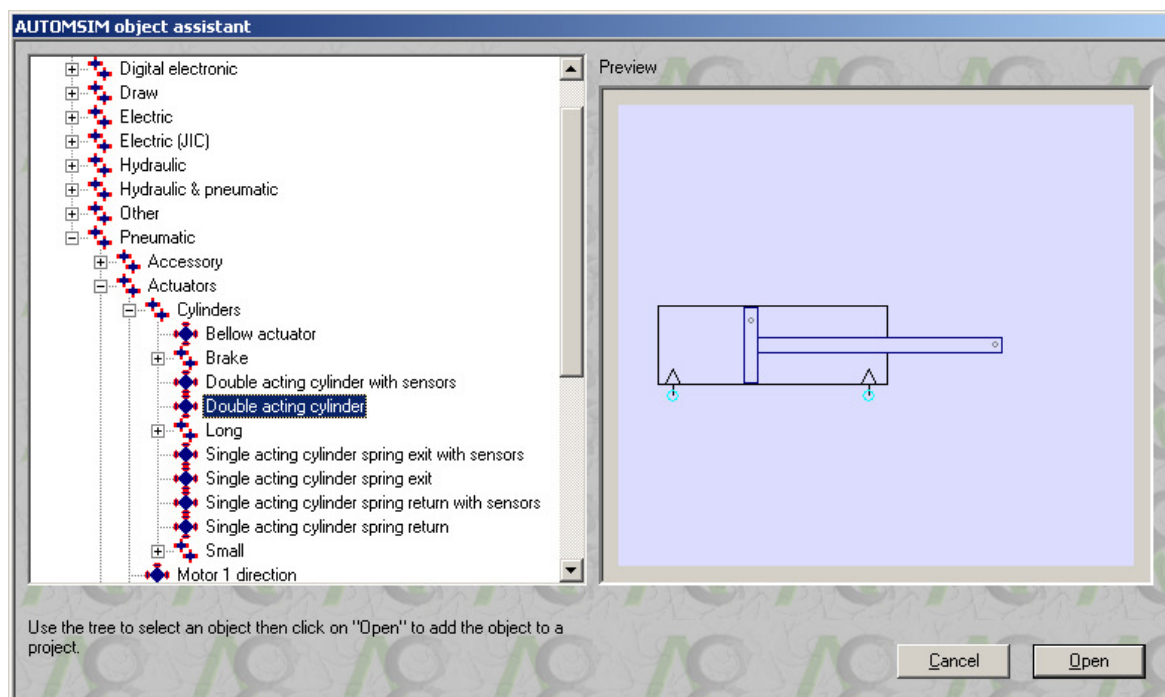
Click with the right side of the mouse on « SIMULA »



Select “Add an SIMULA page”



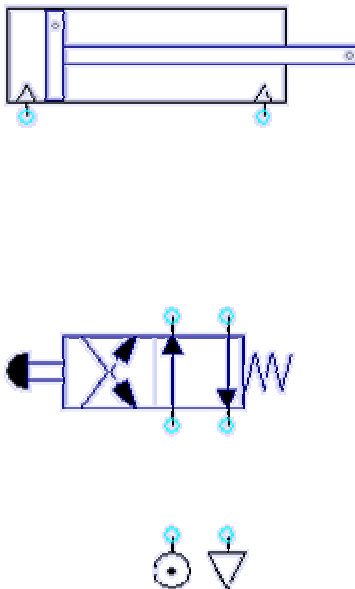
Click with the right side of the mouse on the SIMULA sheet (right part) then select “Add an object”



Select “double acting cylinder”, and then click on “Open”.

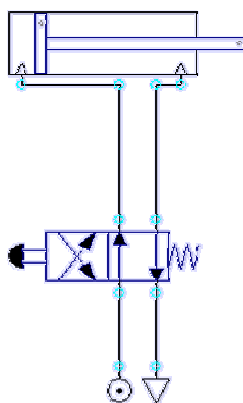
Repeat the steps above then add a 4/2 directional valve with monostable hand control, a pressure source and a pneumatic exhaust.

You should obtain the following:



Create connections between the different components: Move the cursor over the connections (light blue circles), press the left button of the mouse then release it, move the cursor of the mouse to the connection where the link must be connected, press the left button of the mouse then release it.

Repeat the above step for each connection until the following result is achieved:



Click on the “GO” button on the toolbar.

The cylinder shaft will come out. To make it go back in, click on the manual control of the distributor.

While it is running, you can make changes, add objects, move them, etc...

With SIMULA, it is not necessary to stop simulation!

To end the simulation, click again on "GO".

## Using SIMULA

### Organizing applications

SIMULA applications are written on one or more sheets that appear in the tree structure of AUTOSIM. The objects are then placed on the sheet(s): an object = a component such as a cylinder or an electrical contact.

### Opening an existing application

The subdirectory « Examples / SIMULA » of the installation directory of AUTOSIM contains examples done with SIMULA.

### Creating an SIMULA sheet

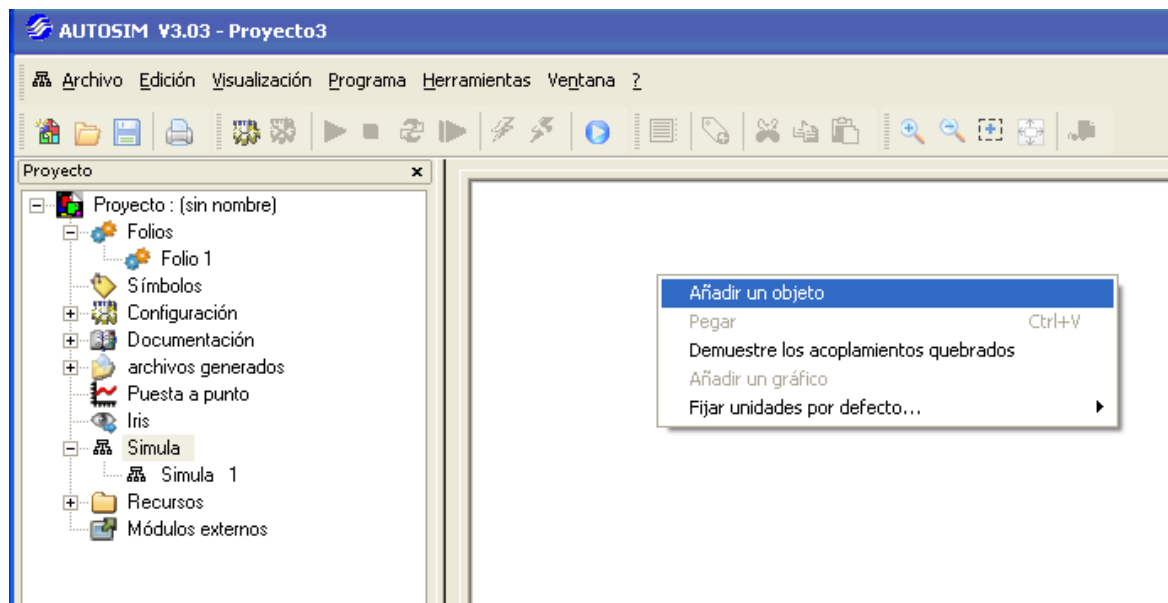
To add an AUTOSIM sheet in the tree structure of a project, click with the right button of the mouse on the “SIMULA” component in the tree structure, then select “Add an SIMULA page”.



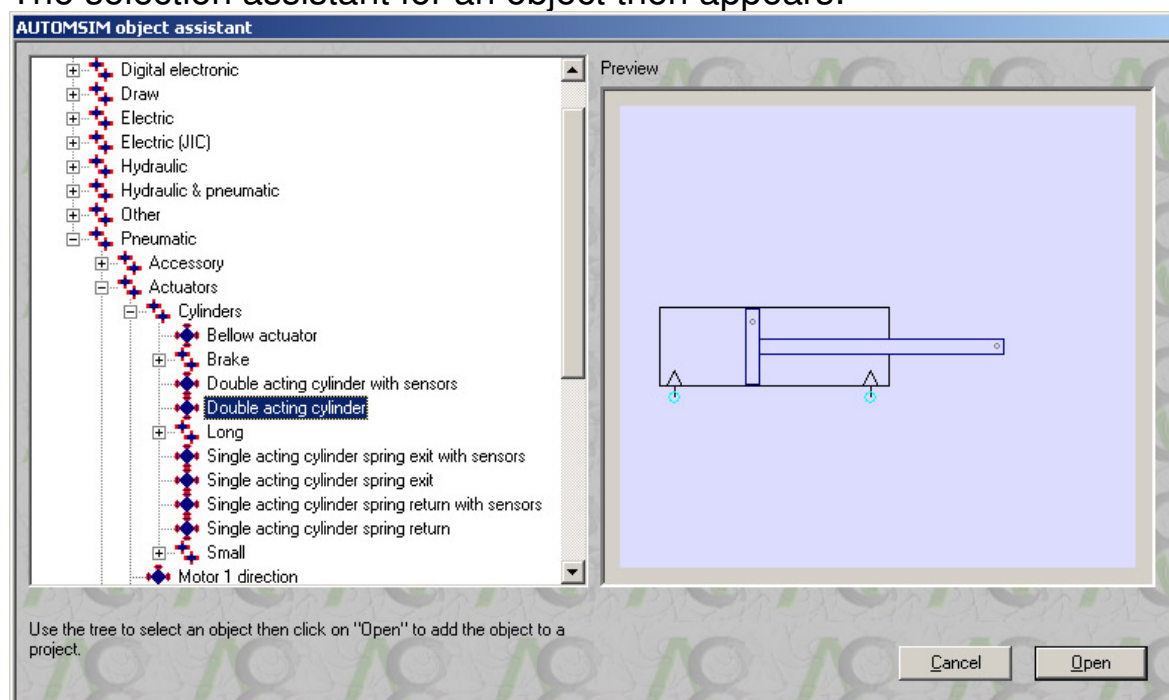
An SIMULA sheet is then created.

## Adding an object onto an SIMULA sheet

Click with the right button of the mouse on the SIMULA sheet ( shown below on the right) and select “Add an object”.



The selection assistant for an object then appears:

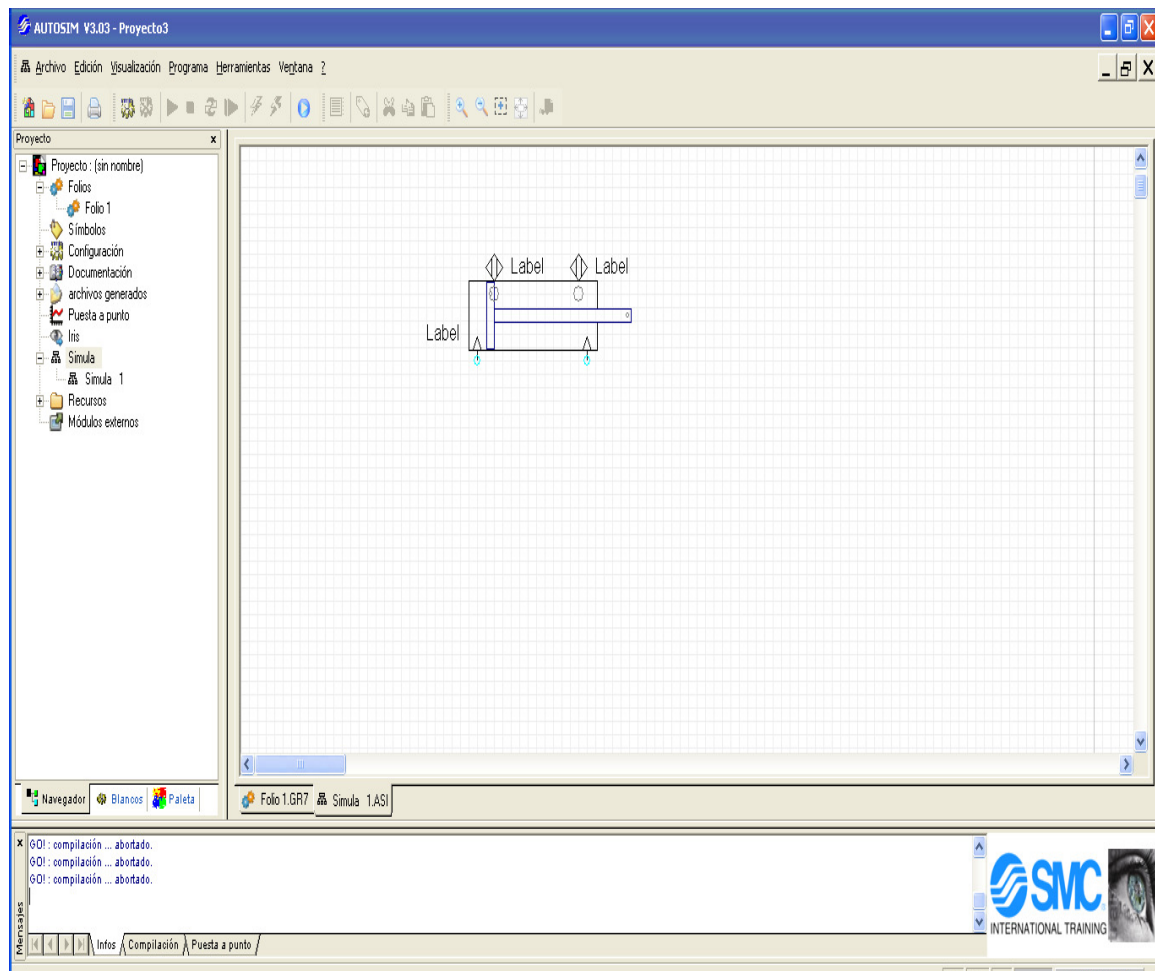


The assistant shows a preview of the object in the bottom of the window. To add the object onto the SIMULA sheet, click on “Open the object”.



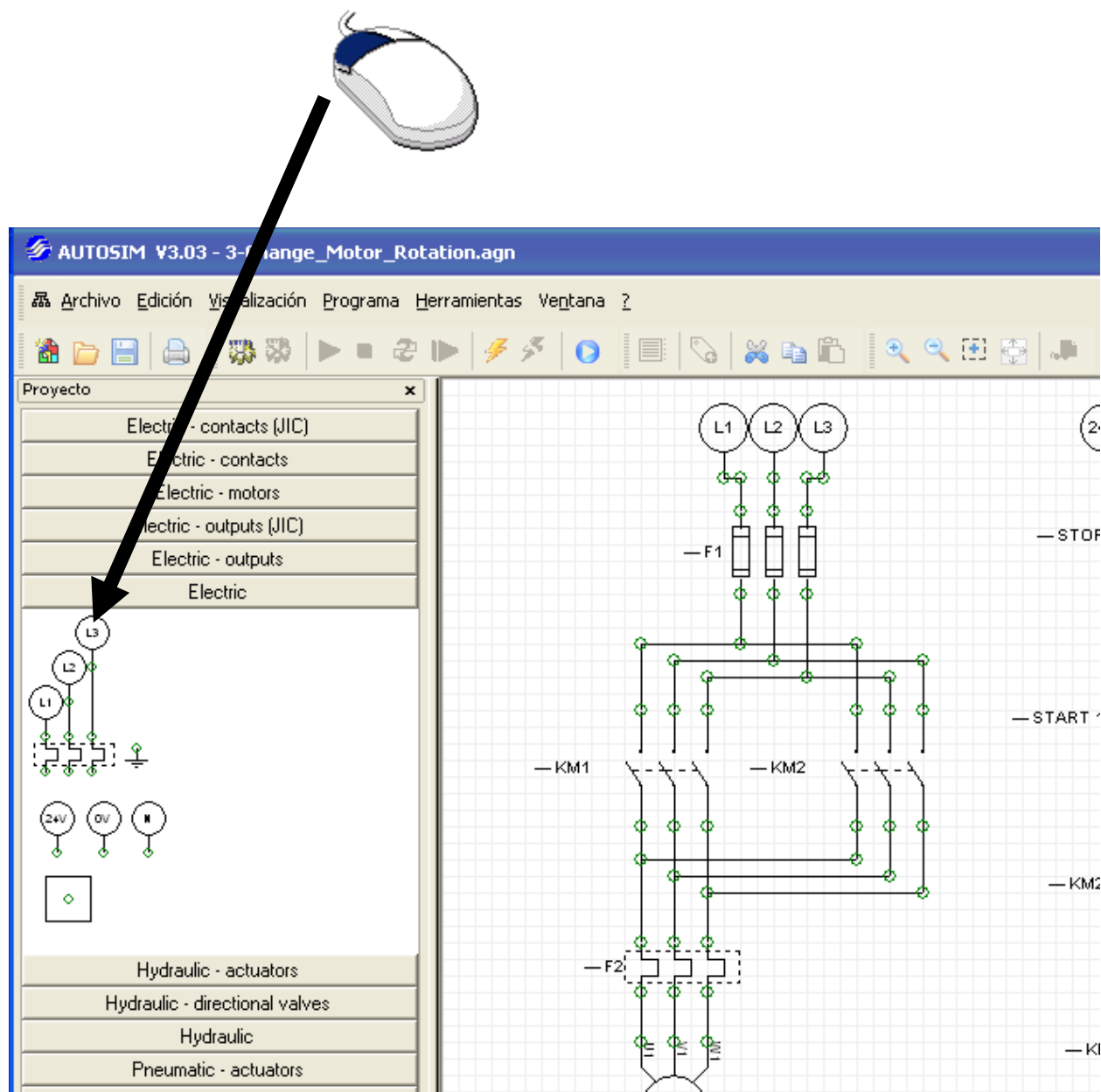
Then move the mouse to place the object on the SIMULA sheet, press the left button of the mouse and release it to leave the object.

You will obtain the following result:

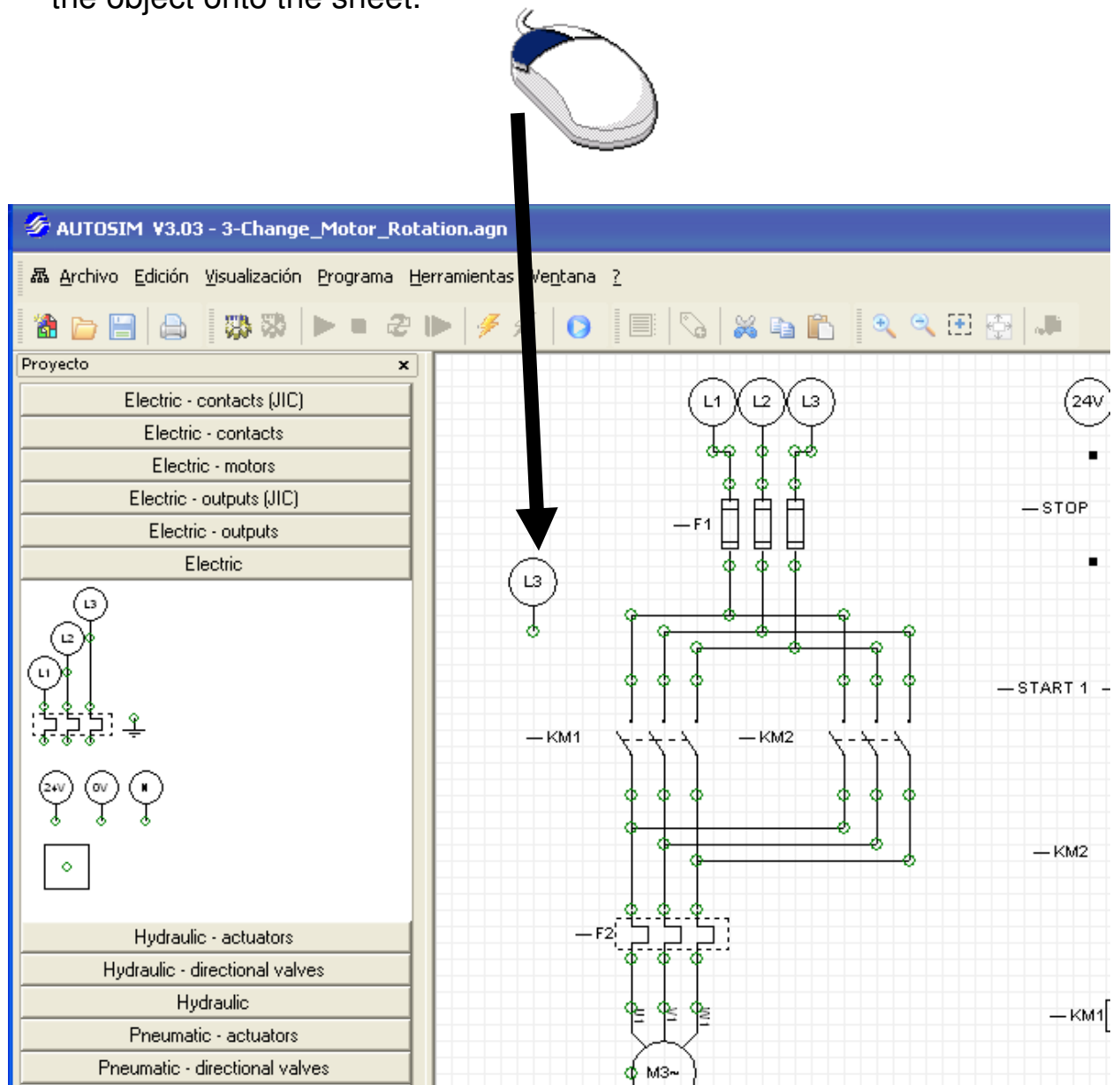


## Using the palette

- 1- Click on the object(s) in the palette (they appear as selected): framed by black squares).

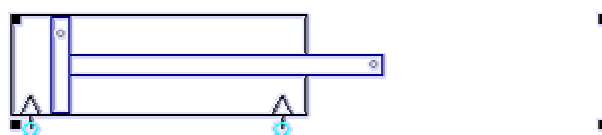


- 2- Click on the selected object(s), keep the button pressed and drag the object onto the sheet.



## Selecting one or more objects.

To select an object, move the cursor of the mouse over the object, press the left button of the mouse and release it. Black squares appear around the objects when they are selected:



To deselect an object, repeat the same step.

To select several objects: keep the SHIFT key of the keyboard pressed and select several objects following the method described above.

To select several objects that are in the same area: press the left button of the mouse, move the cursor of the mouse – a selection rectangle emerges – release the left button of the mouse when the selection rectangle is of the desired size.

To select an object that is under another object (several objects can be superimposed), click several times with the left button of the mouse on the objects covering each other: at each click, the selection moves from one object to the other.

### Selecting one or more objects

Move the cursor over one or more selected objects – the cursor of the mouse takes on the appearance of four direction arrows – press the left button of the mouse, move the objects by moving the mouse, release the left button of the mouse when the desired position for the objects is reached.

### Deleting one or more objects

Move the cursor over one or several selected objects, press then release the right button of the mouse and select “Delete”.

### Changing the orientation of one or more objects

Move the cursor over one or more selected objects, press then release the right button of the mouse and select the desired setting in the “Rotation” menu.

### Copying/cutting one or more objects to the clipboard

Move the cursor over one or more selected objects, press then release the right button of the mouse and select “Copy” or “Paste”.

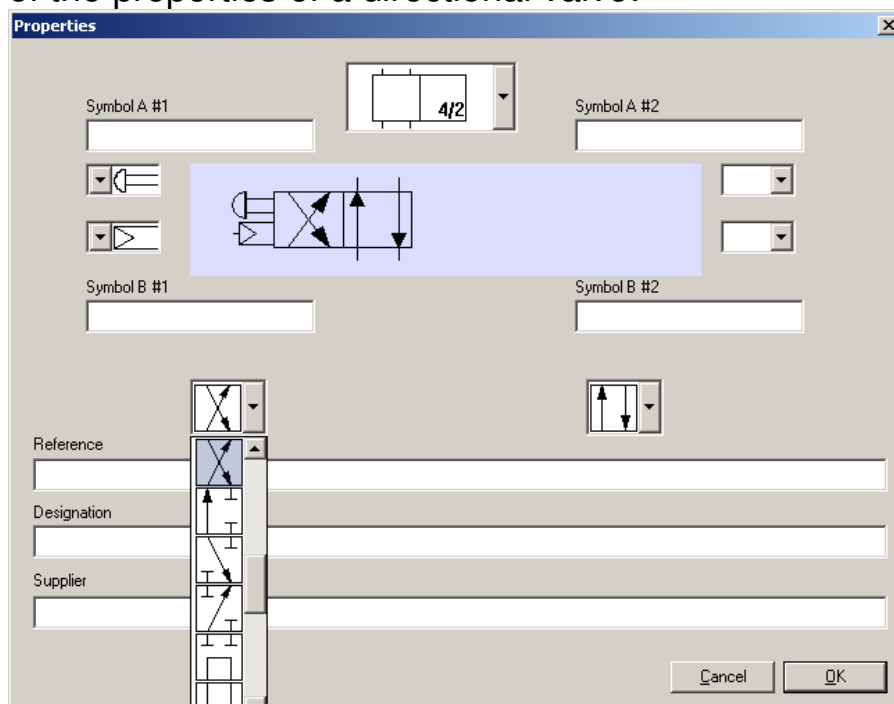
## Pasting one or more objects from the clipboard

Press then release the right button of the mouse over an empty area of the SIMULA sheet and select “Paste” in the menu.

## Modifying object properties.

Move the cursor over one or more selected objects, press then release the right button of the mouse and select “Properties”.

Example of the properties of a directional valve:



## Exporting one or more objects

Move the cursor over one or more selected objects, press then release the right button of the mouse and select “Export”.

The objects are exported to files with the extension .ASO.

By exporting to the subdirectory “SIMULA/lib” of the installation directory of AUTOSIM, the new objects created appear in the SIMULA assistant. The name of the file is the name shown in the assistant. If the name must contain the character ‘/’, substitute this character with ‘@’ in the file name.

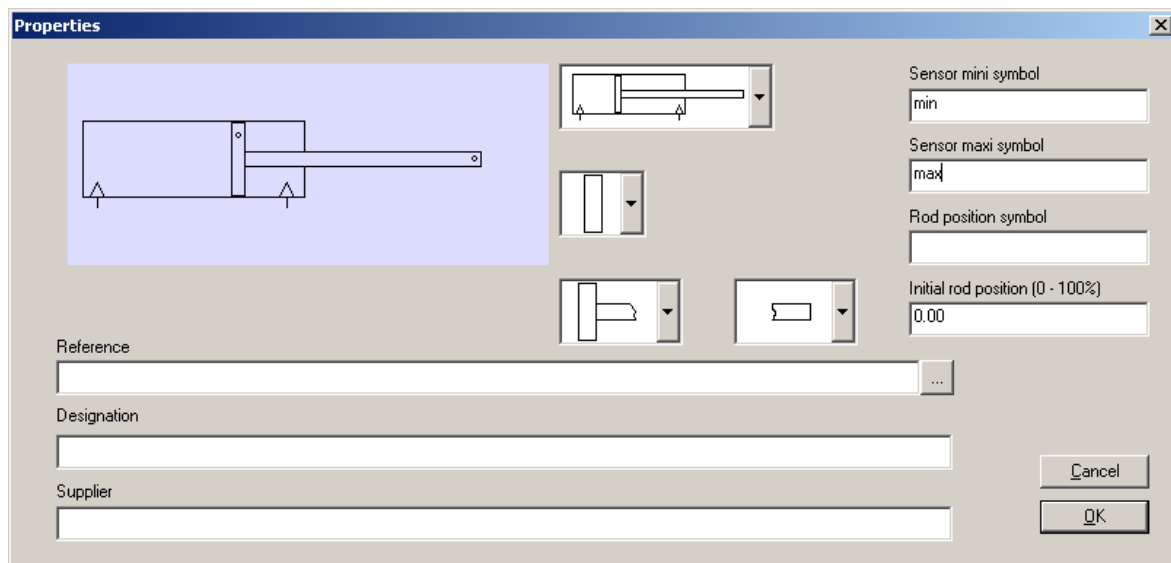
## Advanced functions

### Interactions between objects

Interactions between SIMULA objects are realized either by visual links defined on the sheets (a pneumatic or electrical line connecting two objects, for example) or by a symbol. A symbol is a generic name, for example “mini sensor”. A symbol may have any name whatsoever except for key words reserved for the names of AUTOSIM variables (see the AUTOSIM language reference manual) and symbols used in the AUTOSIM symbol table.

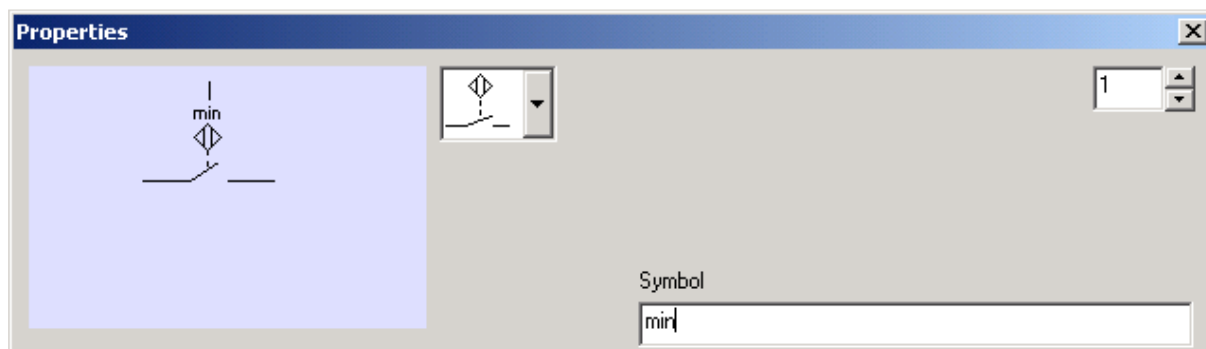
### Creating sensors associated with a cylinder

The mini and maxi end stops of a cylinder can be configured in the properties of the cylinder. Example:

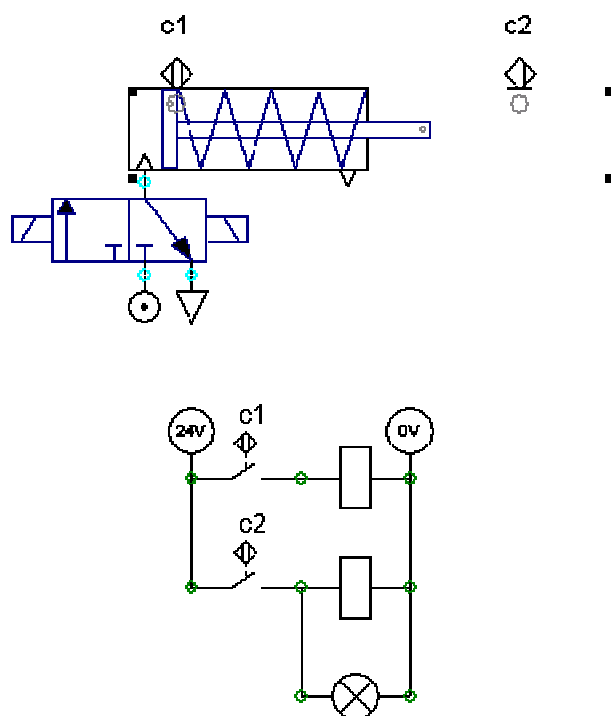


References for the symbols used can be found in the electrical contacts

For example:



The sensors can also be positioned directly on the SIMULA sheet. For example:

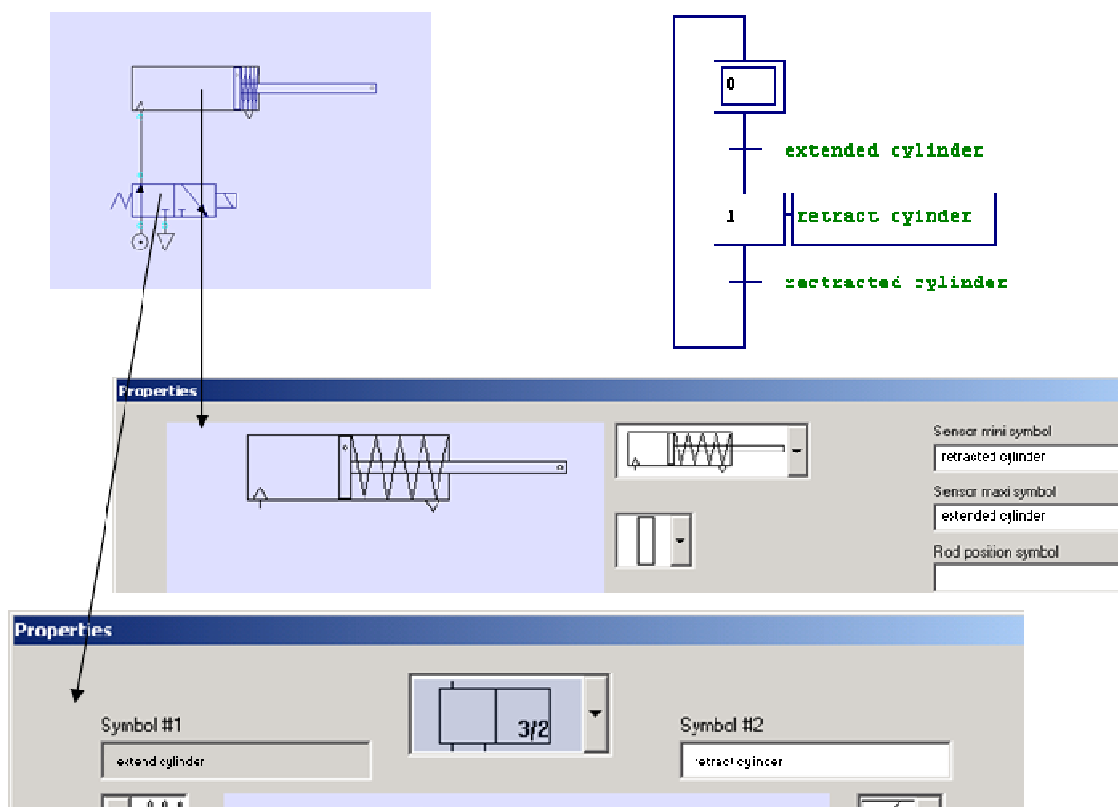


The gray circle associated with the sensor objects must coincide with the gray dot located on the piston or the cylinder shaft so that the sensor is activated.

## Interactions between SIMULA objects and the automaton program

As seen above, the symbols used in the SIMULA objects allow information to be exchanged between the objects. Where you want to communicate solely between SIMULA objects, these symbols cannot be the names of either AUTOSIM variables or AUTOSIM symbols. If you use the name of an AUTOSIM variable or an AUTOSIM symbol, these AUTOSIM objects reference the AUTOSIM variables and may therefore, depending on the actual situation, read or write to the automaton application's variables.

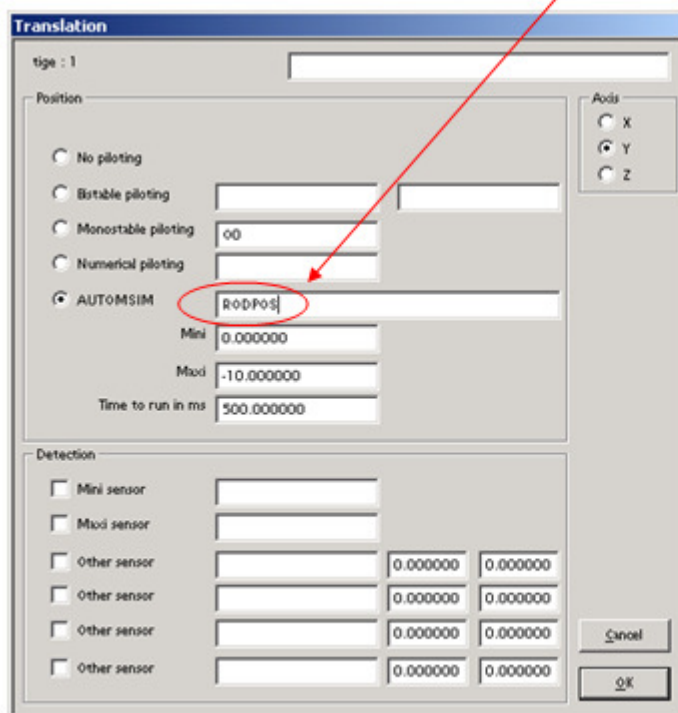
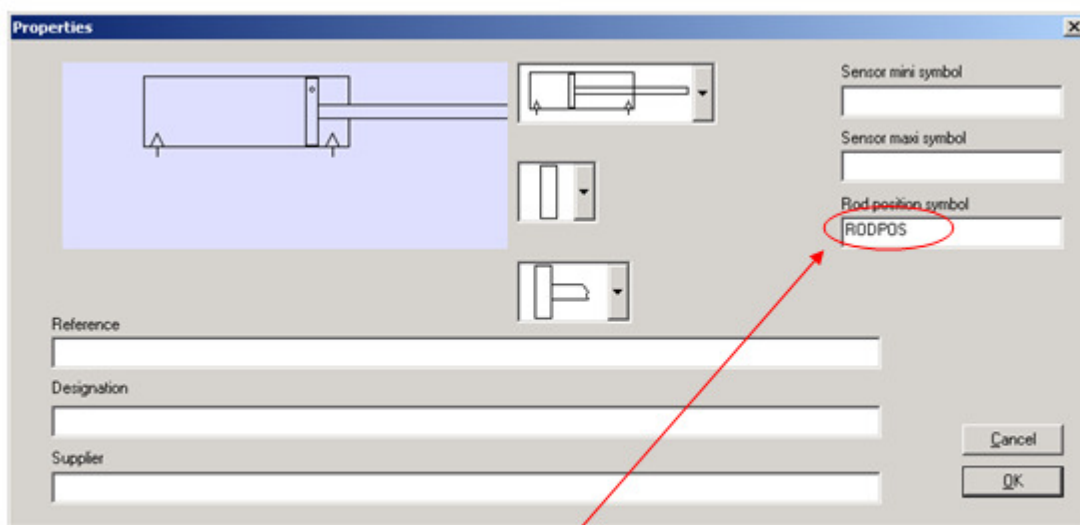
Example:





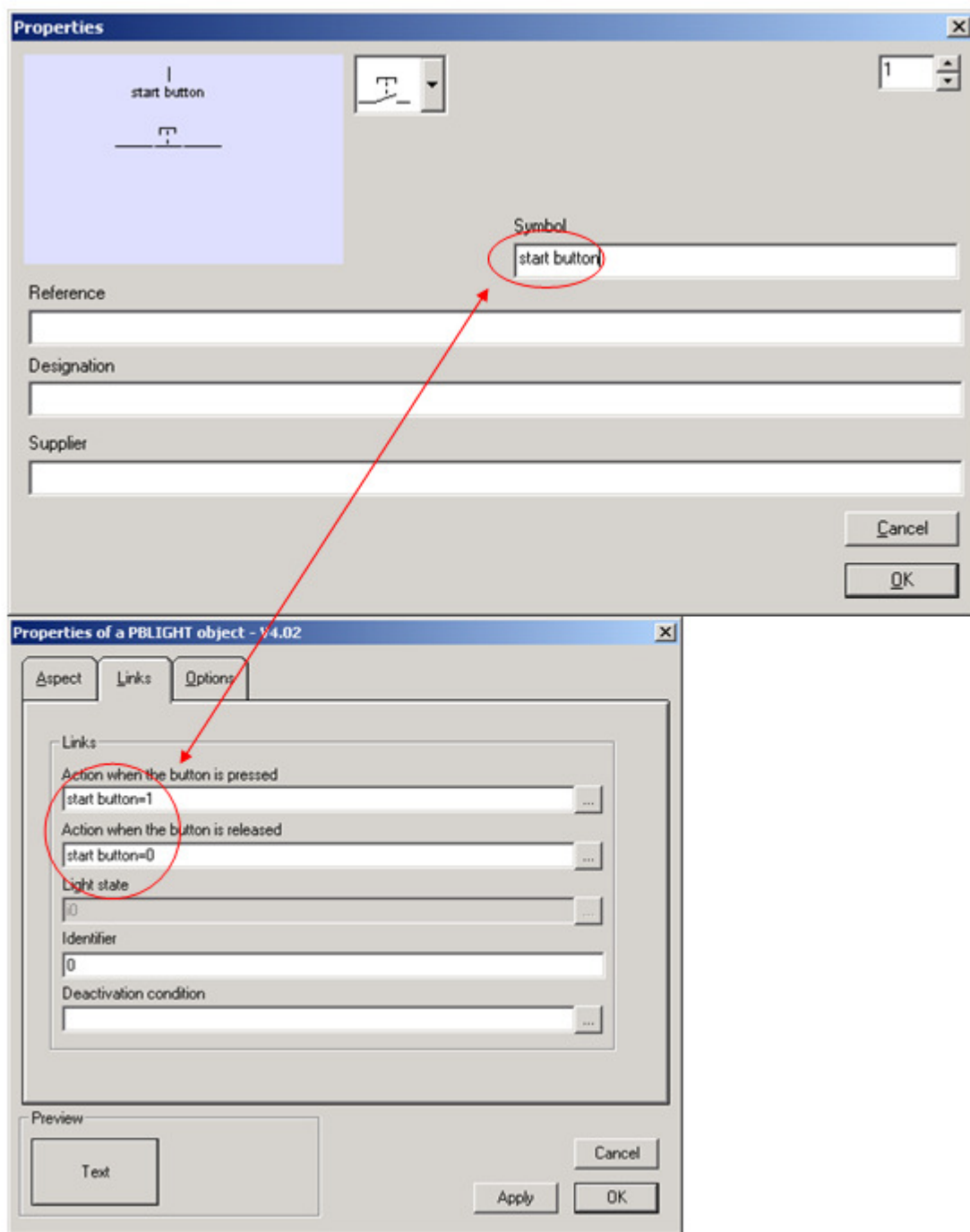
## Interactions between SIMULA objects and the IRIS 3D operational unit simulator

In the IRIS 3D “Translations” and “Rotations” behaviors, the “SIMULA” type allows you to reference the position of an SIMULA cylinder object (see the example complet2.agn).

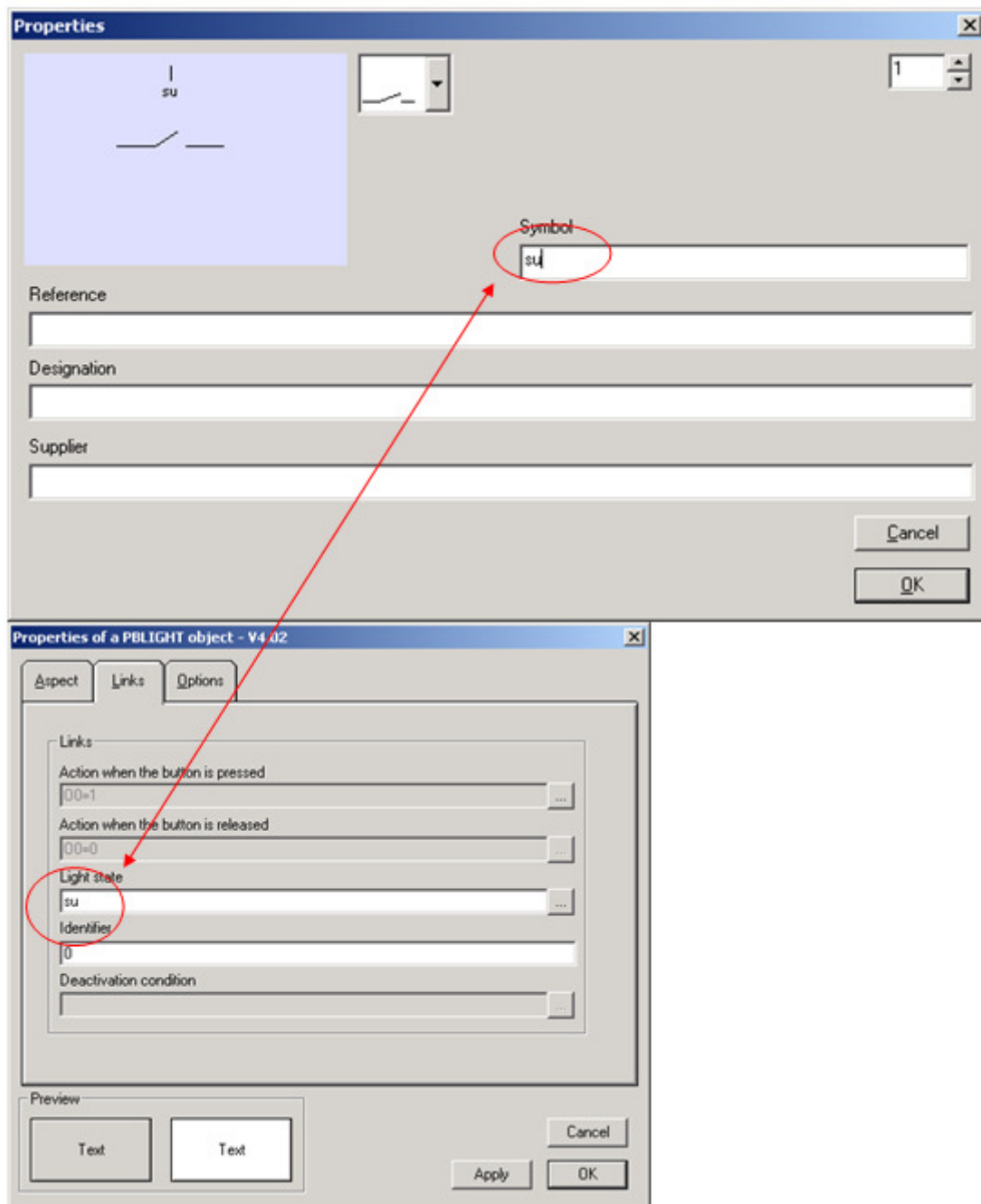


## Interactions between SIMULA objects and the IRIS2D supervision objects

How can a link be made between an IRIS2D pushbutton or switch and an SIMULA pushbutton or switch?



How can a link be made between an SIMULA object and an IRIS2D indicator light?

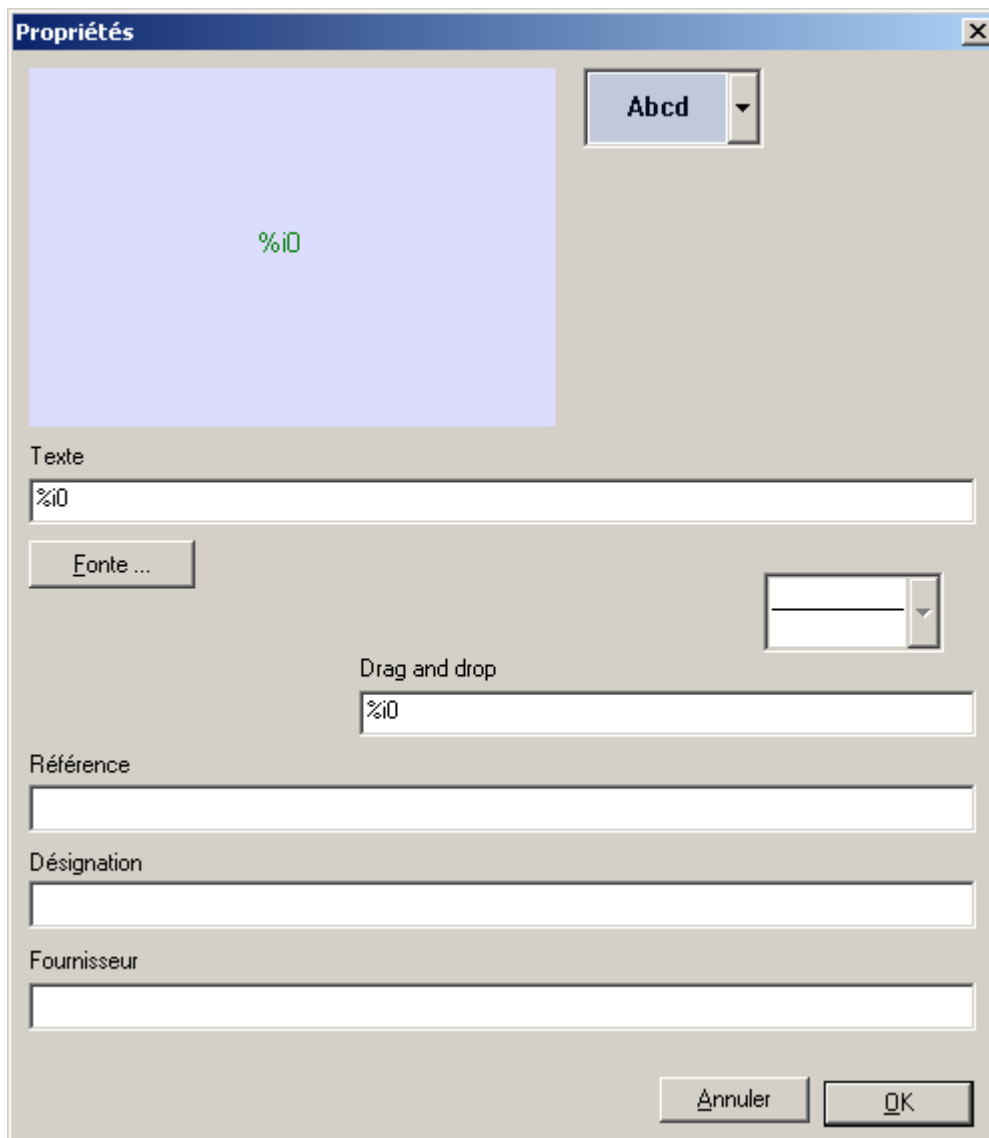


Comments: note that the SIMULA variables are considered as numerical variables. It is therefore necessary to write "su=1"

## Drag and drop from an SIMULA variable to an AUTOSIM sheet

This application is used, for example, in “Beginner” mode in order to be able to “drag” the name of inputs or outputs from the automaton to the AUTOSIM sheet.

To use this application, use a “Design”-type SIMULA object and document the “Drag and drop” section with the text that could be “dragged” from the SIMULA sheet to the AUTOSIM sheet.



Propriétés

%i0

Abcd

Texte

%i0

Fonte ...

Drag and drop

%i0

Référence

Désignation

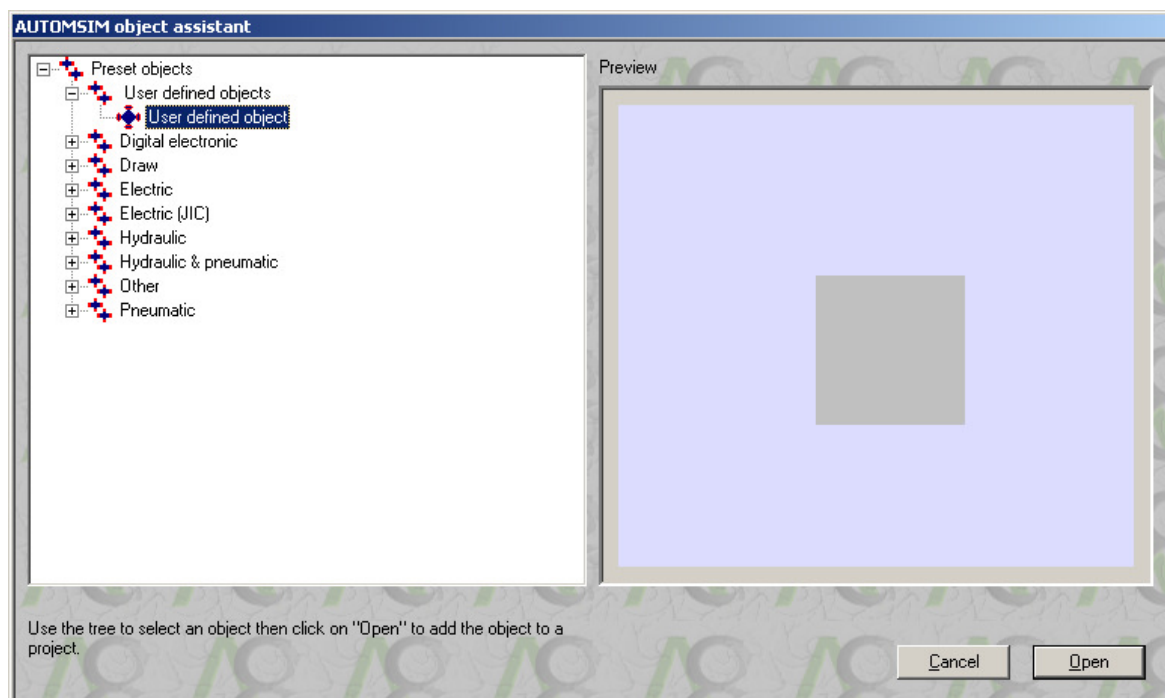
Fournisseur

Annuler OK

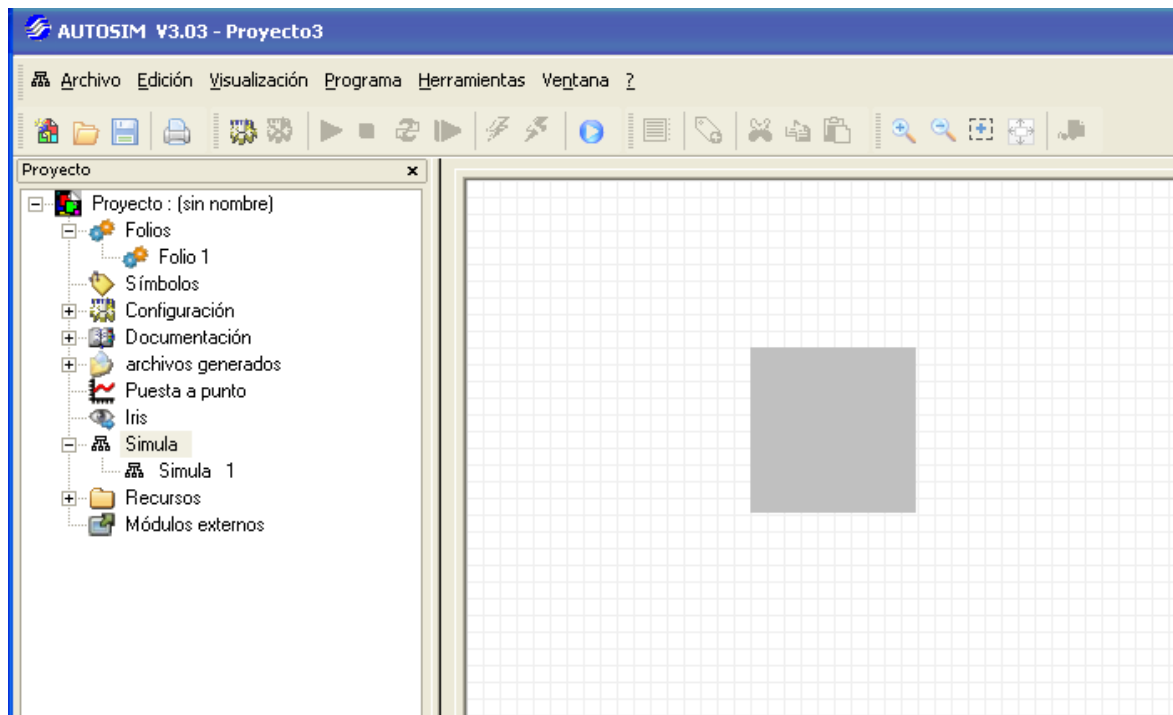
## User-definable objects

The user-definable object will allow you to create your own simulation objects.

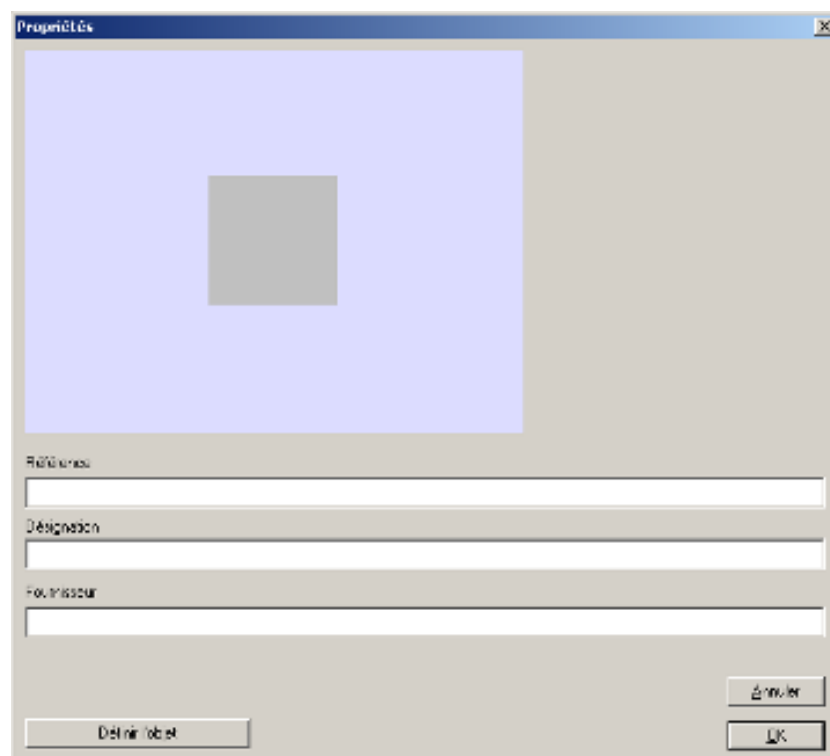
To create such an object, open the following object:

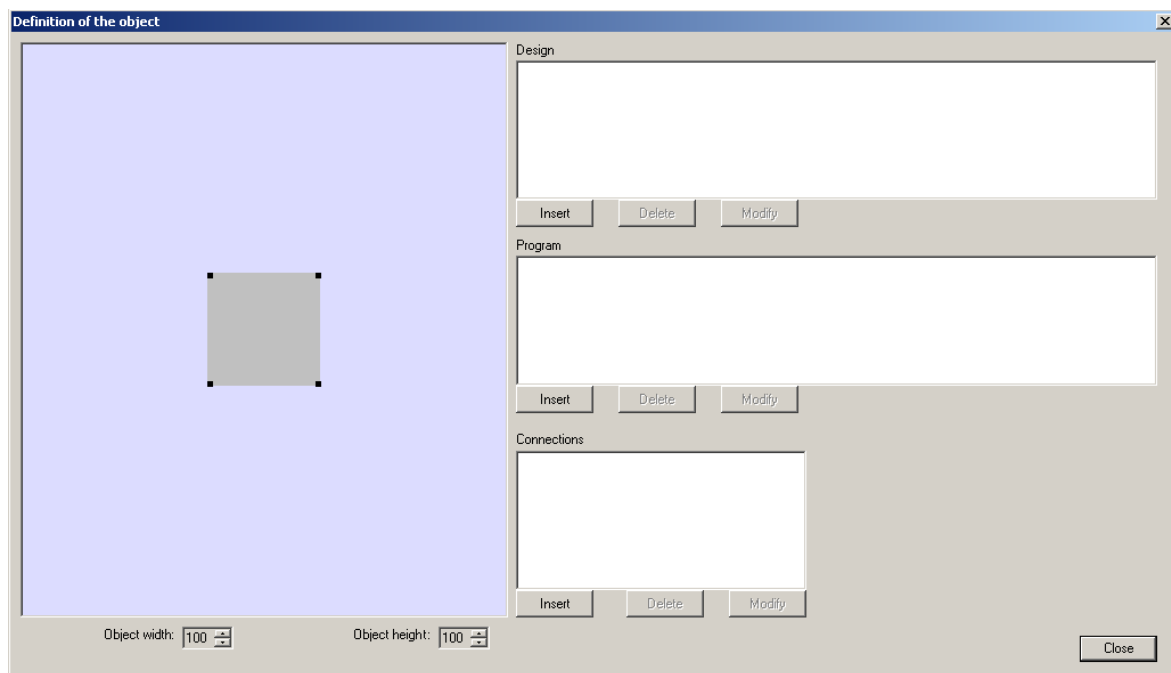


The object is shown as a grey square as long as it has not been parameterized:



To access the object definition, open the object's properties (select the object, right-click over it then "Properties") and click on "Define the object".



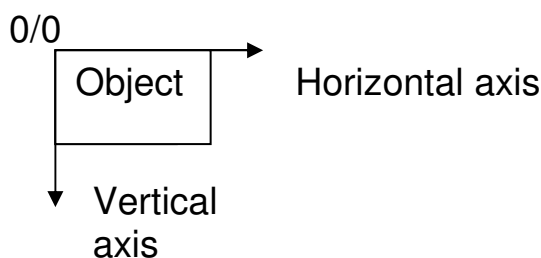


The “Object width” and “Object height” areas allow you to define the dimensions of the object.

The “Designs”, “Program” and “Connections” areas allow you to define the object’s design (its appearance), its behavior and the connections respectively.

## Designs

This area allows you to define the design of the object with the help of the design primitive. The “Insert”, “Delete” and “Modify” buttons allow you respectively to add or delete a primitive or to modify the parameters associated with a primitive. The design primitives use this system of co-ordinates:



Each primitive can receive one or more parameters.

Note that the design primitives only define objects without rotation; the design with rotation is automatically generated by SIMULA. The same is true for the scale: primitives design at scale 1; SIMULA handles scaling according to the zoom selected by the user.

By clicking on “Insert”, a dialogue box allows you to select a design primitive.



## List of design primitives

### Drawing primitive

These primitives produce a drawing.

#### **MOVE**

Moves the pen (without drawing).

Parameters:

- horizontal position,
- vertical position.

#### **LINE**

Draws a line from the pen's current position to the position indicated.

Parameters:

- horizontal position,
- vertical position.

#### **RECT**

Draws a rectangle.

Parameters:

- horizontal position of the top left corner,
- vertical position of the top left corner,
- horizontal position of the bottom right corner,
- vertical position of the bottom right corner.



## ELLI

Draws an ellipse.

Parameters:

- horizontal position of the top left corner of the rectangle containing the ellipse,
- vertical position of the top left corner of the rectangle containing the ellipse,
- horizontal position of the bottom right corner of the rectangle containing the ellipse,
- vertical position of the bottom right corner of the rectangle containing the ellipse.

## RREC

Draws a rectangle with rounded corners.

Parameters:

- horizontal position of the top left corner,
- vertical position of the top left corner,
- horizontal position of the bottom right corner,
- vertical position of the bottom right corner,
- horizontal rounded corner radius,
- vertical rounded corner radius,

## TRIA

Draws a triangle.

Parameters:

- horizontal position of point 1,
- vertical position of point 1,
- horizontal position of point 2,
- vertical position of point 2,
- horizontal position of point 3,
- vertical position of point 3.

## CHOR

Draws a chord (intersection of an ellipse and a straight line).

Parameters:

- horizontal position of the top left corner of the rectangle containing the ellipse,
- vertical position of the top left corner of the rectangle containing the ellipse,
- horizontal position of the bottom right corner of the rectangle containing the ellipse,
- vertical position of the bottom right corner of the rectangle containing the ellipse,
- horizontal position of the start of the line,
- vertical position of the start of the line,
- horizontal position of the end of the line,
- vertical position of the end of the line.

## **ARCE**

Draws an arc of an ellipse (the part of an ellipse cut by a straight line).

Parameters:

- horizontal position of the top left corner of the rectangle containing the ellipse,
- vertical position of the top left corner of the rectangle containing the ellipse,
- horizontal position of the bottom right corner of the rectangle containing the ellipse,
- vertical position of the bottom right corner of the rectangle containing the ellipse,
- horizontal position of the start of the line,
- vertical position of the start of the line,
- horizontal position of the end of the line,
- vertical position of the end of the line.

## **TEXT**

Draws a text box.

Parameters:

- horizontal position,
- vertical position,
- text.

## Attribute primitives

These primitives modify the layout of the drawing primitives (the line or fill color, for example).

### **BRUS**

Modifies the fill color for figures or the background color for text boxes.

Parameter:

- color.

### **PENC**

Modifies the color of lines or text.

Parameter:

- color.

### **FONT**

Modifies the font of the text.

## Other primitives

### **JUMP**

Unconditional jump.

Parameter:

- label.

### **JPIF**

Conditional jump.

Parameters:

- label,
- element 1,
- type of comparison,
- element 2.

(See the programming primitives below for more information).

## DISP

Displays the state of a variable. Can be used for debugging an object by displaying the value of a variable associated with the object.

Parameters:

- variable,
- horizontal position,
- vertical position.

## Program

This area allows you to define the program governing the object's working. Each object has variables:

128 32-bit integer variables,  
128 32-bit floating-point variables.

And also for each connection:

- a floating-point value on input,
- a floating-point value on output,
- an associated writing mode that can have the following values:
  - 0: no writing has been done,
  - 1: the "floating-point value on output" has been written,
  - 2: a connection has been realized with the connection whose number is in "floating-point value on output",
  - 3: locking (pneumatic or hydraulic plug).

The following internal integer variables are special:

125: contains 0 if dynamic visualization is active, 1 if not (useful in order to have a different design for dynamic visualization and otherwise).

126: contains a value representing a user event: 0=no event, 1=left mouse button released, 2=left mouse button pressed, 3=right mouse button released, 4=right mouse button pressed.

127: contains the elapsed time in ms between 2 operations of the program.

## List of programming primitives

### **MOVV**

Copies a constant or a variable into a variable.

Parameters:

- destination variable,
- source variable or constant.

### **ADDV**

Adds a constant or variable to a constant or variable and places the result in a variable.

Parameters:

- destination variable,
- source 1 variable or constant,
- source 2 variable or constant.

### **SUBV**

Subtracts a constant or variable from a constant or variable and places the result in a variable.

Parameters:

- destination variable,
- source 1 variable or constant,
- source 2 variable or constant.

### **MULV**

Multiplies a constant or variable by a constant or variable and places the result in a variable.

Parameters:

- destination variable,
- source 1 variable or constant,
- source 2 variable or constant.

**DIVV**

Divides a constant or variable by a constant or variable and places the result in a variable.

Parameters:

- destination variable,
- source 1 variable or constant,
- source 2 variable or constant.

**ORRV**

Carries out a bit-by-bit OR between a constant or variable and a constant or variable and places the result in a variable.

Parameters:

- destination variable,
- source 1 variable or constant,
- source 2 variable or constant.

**ANDV**

Carries out a bit-by-bit AND between a constant or variable and a constant or variable and places the result in a variable.

Parameters:

- destination variable,
- source 1 variable or constant,
- source 2 variable or constant.

**XORV**

Carries out a bit-by-bit exclusive OR between a constant or variable and a constant or variable and places the result in a variable.

Parameters:

- destination variable,
- source 1 variable or constant,
- source 2 variable or constant.

## JUMP

Unconditional jump.

Parameter:

- label.

## JPIF

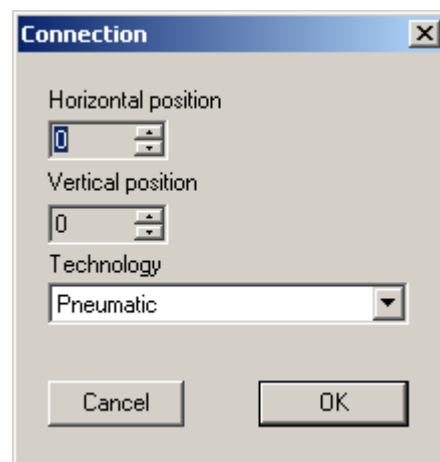
Conditional jump.

Parameters:

- label,
- element 1,
- type of comparison,
- element 2.

## Connections

Makes it possible to create the object's connection points. By clicking on "Insert", the following dialogue box is opened:



For each connection, define the position and the technology. The number shown against each connection must be used to access the value in the object's programming.

## Example

The “Examples\SIMULA” sub-directory of the AUTOSIM installation directory contains an example illustrating use of the user-definable object: a contact:



## Common elements

This chapter describes the common elements for all the languages used in AUTOSIM.

### Variables

The following types of variables are present:

- ⇒ boolean type: the variable may have a true (1) or false (0) value.
- ⇒ numeric type: the variable may have a numeric value, different from the existing types: 16 bits variables, 32 bits and floating point.
- ⇒ time delay type: structured type, it is a combination of a boolean and numeric type.

Starting from version 6 the variable name syntax may be AUTOSIM's or the syntax of IEC standard 1131-3.

### Booleen variables

The following table provides a complete list of the Booleen variables used

Type	Syntax AUTOSIM	Syntax IEC 1131-3	Comments
Input	I0 to I9999	%I0 to %I9999	May or may not correspond to physical input (depending on the I/O configuration of the target).
Output	Q0 to Q9999	%Q0 to %Q9999	May or may not correspond to physical output (depending on the I/O configuration of the target).
System Bits	U0 to U99	%M0 to %M99	For information on the system bits see the manual on the environment.
User bits	U100 to U9999	%M100 to %M9999	Internal bits for general use.

Grafcet Steps	X0 to X9999	%X0 to %X9999	Grafcet step bits
Word bits	M0#0 to M9999#15	%MW0:X0 à %MW9999:X15	Word bits: the number of bits is expressed in decimals and is included between 0 (lower weight bits) and 15 (higher weight bits).

## Numeric variables

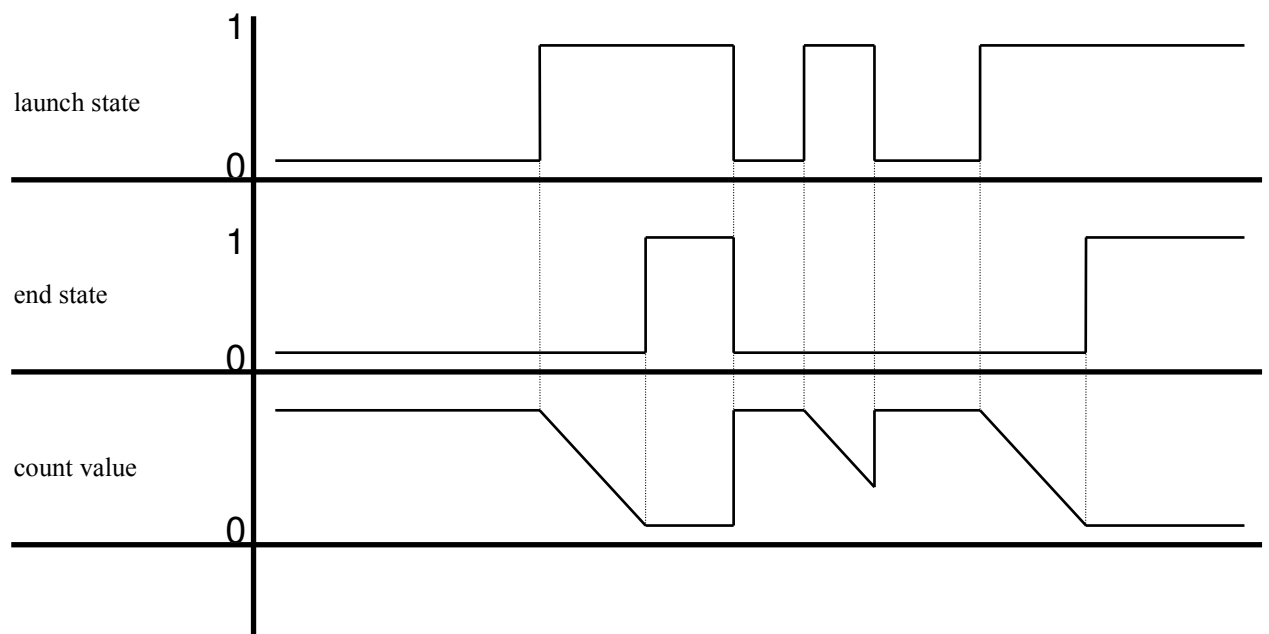
The following table provides a complete list of the numeric variables.

Type	Syntax AUTOSIM	IEC Syntax 1131-3	Comments
Counter	C0 to C9999	%C0 to %C9999	16 bit counter, can be initialized, increased, decreased and tested with boolean languages without using literal language.
System Words	M0 to M199	%MW0 to %MW199	For information on the system words see the manual on the environment.
User words	M200 to M9999	%MW200 to %MW9999	16 bit words for general use.
Long integer	L100 to L4998	%MD100 to %MD4998	Integer value of 32 bits
Float	F100 to F4998	%MF100 to %MF4998	Real value of 32 bits (format IEEE).

## Time delay

Time delay is a combined type which groups two boolean variables (launch state, end state) and two numeric variables on 32 bits (procedure and counter).

The following model shows a time chart of time delay functionality:



A time delay procedure value is between 0 ms and 4294967295 ms (a little over 49 days)

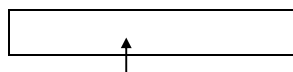
The time delay procedure can be modified by the program (instruction STA).

The time delay counter can be read by the program (instruction LDA).

## Actions

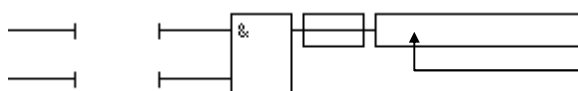
Actions are used in:

⇒ Grafcet language action rectangles,



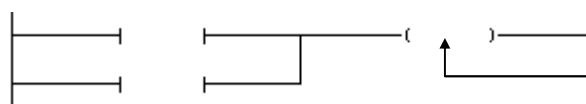
Action

⇒ flow chart language action rectangles,



Action

⇒ ladder language coils.



Action

## Assignment of a boolean variable

The « Assignment » action syntax is:

«boolean variable»

Operation:

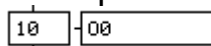
⇒ if the action rectangle or coil command is in a true state then the variable is put at 1 (true state),

⇒ if the action rectangle or coil command is in a false state then the variable is put at 0 (false state).

Truth table:

Command	Variable state (result)
0	0
1	1

Example:



If step 10 is active then 00 takes the value of 1, if not 00 takes the value 0.

Various « Assignment » actions can be used for the same variable in one program. In this case, the different commands are combined in « Or » logic.

Example:



State of X10	State of X50	State of O5
0	0	0
1	0	1
0	1	1
1	1	1

### Complement assignment of a boolean variable

The « Complement assignment » action syntax is:

«N boolean variable»

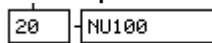
Operation:

- ⇒ if the action rectangle or coil command is in a true state then the variable is reset (false state),
- ⇒ if the action rectangle or coil command is in a false state then the variable is set at 1 (true state).

Truth table:

Command	Variable state (result)
0	1
1	0

Example:



If step 20 is active, then U100 takes the value 0, if not U100 takes the value 1.

Various « Complement assignment » actions can be used for the same variable in one program. In this case, the different commands are combined in « Or » logic.

Example:



State of X100	State of X110	State of O20
0	0	1
1	0	0
0	1	0
1	1	0

### Setting a boolean variable to one

The « Set to one » syntax is:

«S boolean variable»

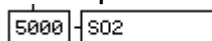
Operation:

- ⇒ if the action rectangle or coil command is in a true state then the variable is set to 1 (true state),
- ⇒ if the action rectangle or coil command is in a false state then the state of the variable is not modified.

Truth table:

Command	Variable state (result)
0	unchanged
1	1

Example:



If step 5000 is active then O2 takes the value of 1, if not O2 keeps the same state.

## Resetting a boolean variable

The « Reset » action syntax is:

«R boolean variable»

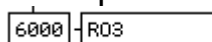
Operation:

- ⇒ if the action rectangle or coil command is in a true state then the variable is reset (false state),
- ⇒ if the action rectangle or coil command is in a false state then the variable state is not modified.

Truth table:

Command	Variable state (result)
0	unchanged
1	0

Example:



If step 6000 is active then O3 takes the value of 0, if not O3 keeps the same state.

## Inverting a boolean variable

The « Inversion » action syntax is:

«I boolean variable»

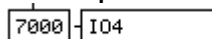
Operation:

- ⇒ if the action rectangle or coil command is in a true state then the variable state is inverted for each execution cycle,
- ⇒ if the action rectangle or coil command is in a false state then variable state is not modified.

Truth table:

Command	Variable state (result)
0	unchanged
1	inverted

Example:



If step 7000 is active then the state of O4 is inverted, if not O4 keeps the same state.

### Resetting a counter, a word or a long

The « Reset a counter, word or long» syntax is:

«R counter or word»

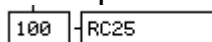
Operation:

- ⇒ if the action rectangle or coil command is in a true state then the counter, word or long is reset,
- ⇒ if the action rectangle or coil command is in a false state then the counter, word or long is not modified.

Truth table:

Command	Value of counter, word or long (result)
0	unchanged
1	0

Example:



If step 100 is active then counter 25 is reset, if not C25 keeps the same value.



### Incrementing a counter, a word or a long

The «Increment a counter » action syntax is:

«+ counter, word or long»

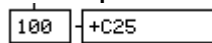
Operation:

- ⇒ if the action rectangle or coil command is in a true state then the counter, word or long is incremented for each execution cycle,
- ⇒ if the action rectangle or coil command is in a false state then the counter, word or long is not modified.

Truth table:

Command	Counter, word or long value (result)
0	Unchanged
1	current value +1

Example:



If step 100 is active then counter 25 is incremented, if not then C25 keeps the same value.

### Decrementing a counter, word or long

The « Decrement a counter » action syntax is:

«- counter, word or long»

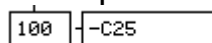
Operation:

- ⇒ if the action rectangle or coil command is in a true state then the counter, word or long is decremented for each execution cycle,
- ⇒ if the action rectangle or coil command is in a false state then the counter, word or long is not modified..

Truth table:

Command	Value or counter, word or long (result)
0	unchanged
1	current value -1

Example:



If step 100 is active then counter 25 is decreased, if not C25 keeps the same value.

### Time delays

Time delays are considered as boolean variables and can be used with « Assignment », « Complement assignment », « Set to one », « Reset », and « Invert ». The time delay order can be written after the action. The syntax is::

« time delay(duration) »

By default the duration is expressed in tenths of seconds. The letter « S » at the end of the duration indicates that it is expressed in seconds.

Examples:



Step 10 launches a time delay of 2 seconds which remains active as long as the step is. Step 20 sets a time delay of 6 seconds which remains active while step 20 is deactivated.

The same time delay can be used by different places with the same procedure and at different instants. In this case the time delay procedure must only be indicated once.

Note: other syntaxes exist for time delays.

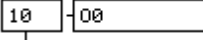
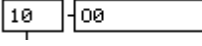
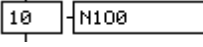
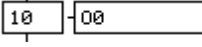
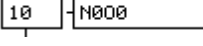
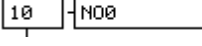
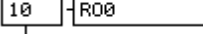
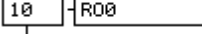
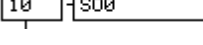
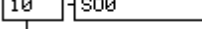
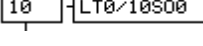
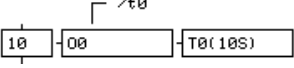
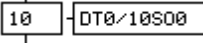
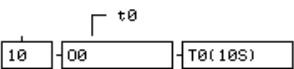
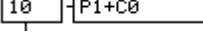

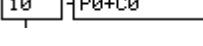

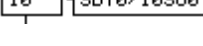
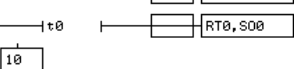
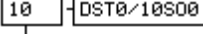
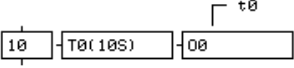
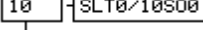

### Interferences among the actions

Certain types of actions cannot be used at the same time on a variable. The table below shows the combinations which cannot be used:

	Assignment	Complement assignment	Set to one	Reset	Inversion
<b>Assignment</b>	YES	NO	NO	NO	NO
<b>Complement assignment</b>	NO	YES	NO	NO	NO
<b>Set to one</b>	NO	NO	YES	YES	YES
<b>Reset</b>	NO	NO	YES	YES	YES
<b>Inversion</b>	NO	NO	YES	YES	YES

## IEC1131-3 standard actions

The table below provides the IEC 1131-3 standard actions which can be used with AUTOSIM V>=6 based on the AUTOSIM V5 standard syntax.

Name	AUTOSIM V>=6 Syntax	AUTOSIM V5 Syntax	AUTOSIM V>=6 Example	Equivalent example AUTOSIM V5
Not memorized	No value	No value		
Not memorized	N1	No value		
Complement not memorized	N0	N		
Reset	R	R		
Set to 1	S	S		
Limited in time	LTn/duration	Non-existent		
Time delay	DTn/duration	Non-existent		
Pulse on rising edge	P1	Non-existent		
Pulse on falling edge	P0	Non-existent		
Memorized and time delay	SDTn/duration	Non-existent		
Time delay and memorized	DSTn/duration	Non-existent		
Memorized limited in time	SLTn/duration	Non-existent		

## Multiple actions

Within the same action rectangle or coil, multiple actions can be written by separating them with « , » (comma).

Example:

50 - 00,N01,S02,R03,RC0,+C1,-C2

Multiple action rectangles (Grafcet and flow chart) or coils (ladder) can be combined. See the chapters on the relative languages for more information.

## Literal code

Literal code can be entered in an action rectangle or coil.

The syntax is:

« { literal code } »

Multiple lines of literal language can be written in braces. A « , » (comma) is also used here to separate them.

Example:

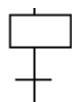
10 - {M200=[M200+10]}

For more information see the chapters « Low level literal language », «Extended literal language » and «ST literal language».

## Tests

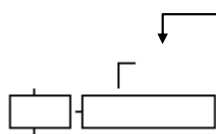
Tests are used in:

⇒ Grafcet language transitions,

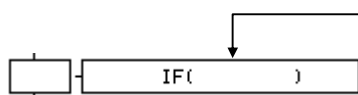


Test

⇒ conditions based on Grafcet language action,

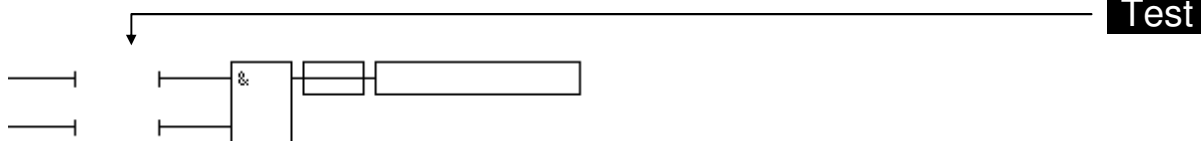


Test



Test

⇒ flow chart language tests,



⇒ ladder language tests.



## General form

A test is a boolean equation composed of one or n variables separated by the operators « + » (or) or « . » (and).

### Example of a test:

```
i0 (test input 0)
i0+i2 (test input 0 « or » input 2)
i10.i11 (test input 10 « and » input 11)
```

## Test modifier

By default if only the name of one variable is specified, the test is « equal to one» (true). Modifiers make it possible to test the complement state, the rising edge and the falling edge.

⇒ the character « / » placed before a variable tests the complement state,

⇒ the character « u » or the character « ↑\* » placed before a variable tests the rising edge

⇒ the character « d » or the character « ↓\*\* » placed before a variable tests the falling edge

Text modifiers can be applied to one variable or to an expression between parentheses.

### Examples:

```
↑ i0
/i1
/(i2+i3)
↓(i2+(i4./i5))
```

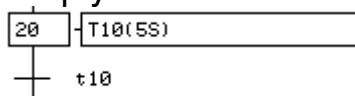
\* To obtain this character when editing a test press the [↑] key.

\*\* To obtain this character when editing a test press the [↓] key.

## Time delays

Four syntaxes are available for time delays.

In the first the time delay is activated in the action and the time delay is simply mentioned in a test to check the end state:



For the others, everything is written in the test. The general form is:

« time delay / launch variable / duration »

or

« duration / launch variable / time delay »

or

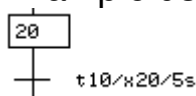
« duration / launch variable »

In this case, a time delay is automatically attributed. The attribution range is that of the automatic symbols.

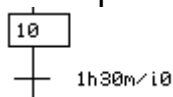
By default the duration is expressed in tenths of seconds.

The duration can be expressed in days, hours, minutes, seconds, milliseconds with the operators « d », « h », « m », « s » and « ms ». For example: 1d30s = 1 day and 30 seconds.

Example using the second syntax:



Example using the normalized syntax:



## Priority of boolean operators

By default the boolean operator «. » (AND) has a greater priority than the operator «+» (OR). Parentheses can be used to set a different priority.

Examples:

i0.(i1+i2)

((i0+i1).i2)+i5

## Always true test

The syntax of an always true test is:

« » (no value) or « =1 »

## Numeric variable test

Numeric variable tests must use the following syntax:

« numeric variable » « test type » « constant or numeric variable »

The test type can be:

- ⇒ « = » equal,
- ⇒ « ! » or « <> » different,
- ⇒ « < » less than (not signed),
- ⇒ « > » greater than (not signed),
- ⇒ « << » less than (signed),
- ⇒ « >> » greater than (signed),
- ⇒ « <= » less than or equal to (not signed),
- ⇒ « >= » greater than or equal to (not signed),
- ⇒ « <=< » less than or equal to (signed),
- ⇒ « >=> » greater than or equal to (signed).

A float can only appear with another float or a real constant.

A long can only appear with another long or a long constant.

A word or a counter can only appear with a word, a counter or a 16 bit constant.

Real constants must be followed by the letter « R ».

Long constants (32 bits) must be followed by the letter « L ».

16 or 32 bit integer constants are written in decimal by default. They can be written in hexadecimal (suffix « \$ » or « 16# ») or in binary (suffix « % » or « 2# »).

Numeric variable tests are used in equations like boolean variable tests. They can be used with test modifiers as long as they are in parentheses.

### Examples:

```
m200=100
```

```
%mw1000=16#abcd
```

```
c10>20.c10<100
```

```
f200=f201
```

```
m200=m203
```

```
%md100=%md102
```

```
f200=3.14r
```

```
l200=$12345678L
```

```
m200<<-100
```

```
m200>>1000
```

```
%mw500<=12
```

```

/ (m200=4)
↓ (m200=100)
/ (1200=100000+1200=-100000)

```

## Transitions on multiple lines

Transition text can be extended to multiple lines. The end of a transition line must be an operator « . » or « + ». A combination of key [CTRL] + [↓] and [CTRL] + [↑] makes it possible to move the cursor from one line to another.

## Use of symbols

Symbols make it possible to associate a text to a variable.  
 Symbols can be used with all the languages.  
 A symbol must be associated to one and only one variable.

## Symbol syntax

The symbols are composed of:

- ⇒ an optional character « \_ » (low dash, generally associated with key [8] on the keyboard) which indicates the beginning of the symbol,
- ⇒ the name of the symbol,
- ⇒ an optional character « \_ » (low dash) which indicates the end of the symbol.

The characters « \_ » enclosing the symbol names are optional. They must be used if the symbol starts with a digit or an operator (+, -, etc...).

## Automatic symbols

It can be a nuisance to have to set the attribution in each symbol and a variable, particularly if the precise attribution of a variable number is not very important. Automatic symbols are a solution to this problem, they are used to let the compiler automatically generate the attribution of a symbol to a variable number. The type of variable to use is provided in the name of the symbol.

## Automatic symbol syntax

The syntax of automatic symbols is as follows:

```
_« symbol name » %« variable type »_
```

« variable type » can be:

I , O or Q, U or M, T, C, M or MW, L or MD, F or MF.

It is possible to reserve multiple variables for a symbol. This is useful for setting tables. In this case the syntax is:

```
_« symbol name » %« variable »« length »_
```



«length » represents the number of variables to be reserved.

### How does the compiler manage the automatic symbols ?

When starting to compile an application, the compiler cancels all the automatic symbols located in the « .SYM » file of the application. Each time the compiler finds an automatic symbol it creates a unique attribution for the symbol based of the variable type specified in the symbol name. The symbol that is then generated is written in the « .SYM » file. If the same automatic symbol is present more than once in an application, it refers to the same variable.

### Range of variable attribution

By default, an attribution range is set for each type of variable:

Type	Start	End
I or %I	0	9999
O or %Q	0	9999
U or %M	100	9999
T or %T	0	9999
C or %C	0	9999
M or %MW	200	9999
L or %MD	100	4998
F or %MF	100	4998

The attribution range can be changed for each type of variable by using the compilation command #SR« type »=« start », « end »  
« type » designates the type of variable, start and end and the new limits to be used.

This command modifies the attribution of automatic variables for the entire sheet where it is written and up to the next « #SR » command.

### Fixed-address symbols

The syntax of the automatic symbols is:

\_« symbol name » %« variable name »\_

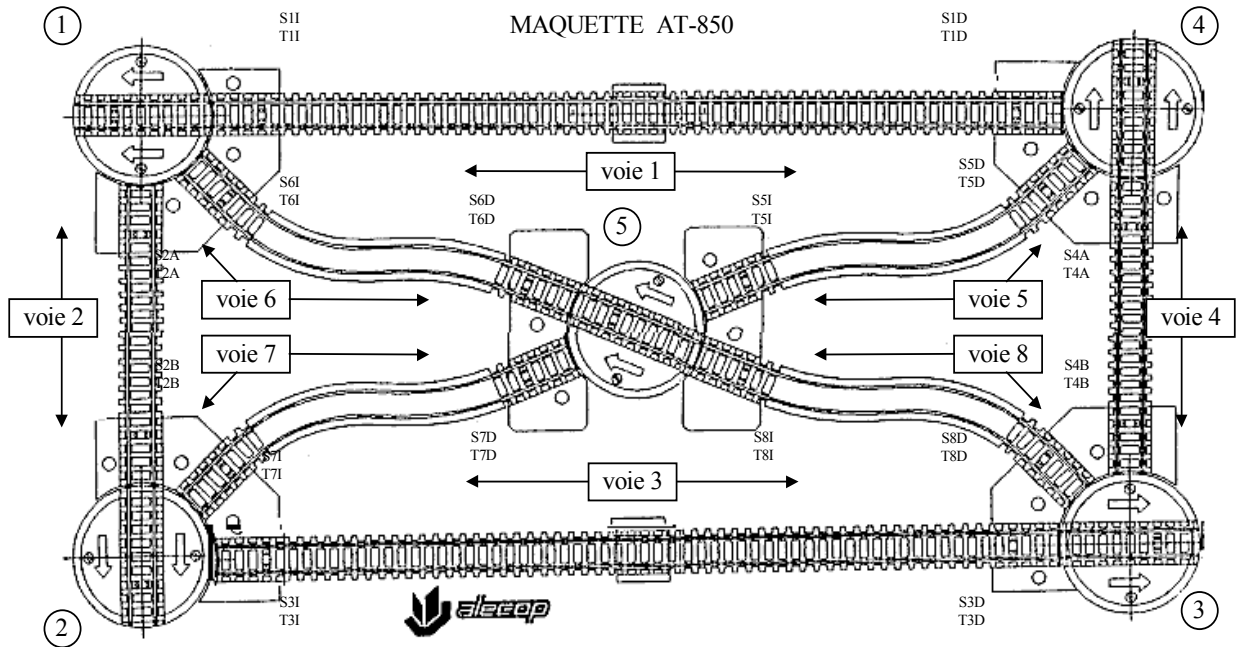
For example:

open valve%%q3

Designates a symbol that will be linked to the variable %Q3.

## Examples

To better illustrate this manual we have developed some functional examples with a model of a train as in the diagram below



We have used I/O cards on a PC to pilot this model. The symbols set by the constructor of the model have been saved.

The following symbol file was created:

AV1	O0	alimentation voie 1
AV2	O1	alimentation voie 2
AV3	O2	alimentation voie 3
AV4	O3	alimentation voie 4
AV5	O4	alimentation voie 5
AV6	O5	alimentation voie 6
AV7	O6	alimentation voie 7
AV8	O7	alimentation voie 8
AP1	O8	alimentation plateforme 1
AP2	O9	alimentation plateforme 2
AP3	O10	alimentation plateforme 3
AP4	O11	alimentation plateforme 4
AP5	O12	alimentation plateforme 5
IP1	O13	rotation plateforme 1
IP2	O14	rotation plateforme 2
IP3	O15	rotation plateforme 3
IP4	O16	rotation plateforme 4
IP5	O17	rotation plateforme 5
ZP1	O18	initialisation plateforme 1
ZP2	O19	initialisation plateforme 2
ZP3	O20	initialisation plateforme 3
ZP4	O21	initialisation plateforme 4
ZP5	O22	initialisation plateforme 5
DV1	O23	direction voie 1
DV2	O24	direction voie 2
DV3	O25	direction voie 3
DV4	O26	direction voie 4
DV5	O27	direction voie 5
DV6	O28	direction voie 6
DV7	O29	direction voie 7
DV8	O30	direction voie 8
S1D	O31	feu droit voie 1
S1I	O32	feu gauche voie 1
S2A	O33	feu haut voie 2
S2B	O34	feu bas voie 2
S3D	O35	feu droit voie 3
S3I	O36	feu gauche voie 3
S4A	O37	feu haut voie 4
S4B	O38	feu bas voie 4
S5D	O39	feu droit voie 5
S5I	O40	feu gauche voie 5
S6D	O41	feu droit voie 6
S6I	O42	feu gauche voie 6
S7D	O43	feu droit voie 7
S7I	O44	feu gauche voie 7
S8D	O45	feu droit voie 8
S8I	O46	feu gauche voie 8
T1D	i0	train droit voie 1
T1I	i1	train gauche voie 1
T2A	i2	train haut voie 2
T2B	i3	train bas voie 2
T3D	i4	train droit voie 3
T3I	i5	train gauche voie 3
T4A	i6	train haut voie 4
T4B	i7	train bas voie 4
T5D	i8	train droit voie 5

T5I	i9	train gauche voie 5
T6D	i10	train droit voie 6
T6I	i11	train gauche voie 6
T7D	i12	train droit voie 7
T7I	i13	train gauche voie 7
T8D	i14	train droit voie 8
T8I	i15	train gauche voie 8
TP1	i16	train plateforme 1
TP2	i17	train plateforme 2
TP3	i18	train plateforme 3
TP4	i19	train plateforme 4
TP5	i20	train plateforme 5
P1P	i21	index plateforme 1
P2P	i22	index plateforme 2
P3P	i23	index plateforme 3
P4P	i24	index plateforme 4
P5P	i25	index plateforme 5
P1Z	i26	init plateforme 1
P2Z	i27	init plateforme 2
P3Z	i28	init plateforme 3
P4Z	i29	init plateforme 4
P5Z	i30	init plateforme 5
ERR	i31	court-circuit

## Grafcet

AUTOSIM supports the following elements:

- ⇒ divergences and convergences in « And » and in « Or »,
- ⇒ destination and source steps,
- ⇒ destination and source transitions,
- ⇒ synchronization,
  - ⇒ setting Grafkets,
  - ⇒ memorization of Grafkets,
  - ⇒ fixing,
  - ⇒ macro-steps.

## Simple Grafcet

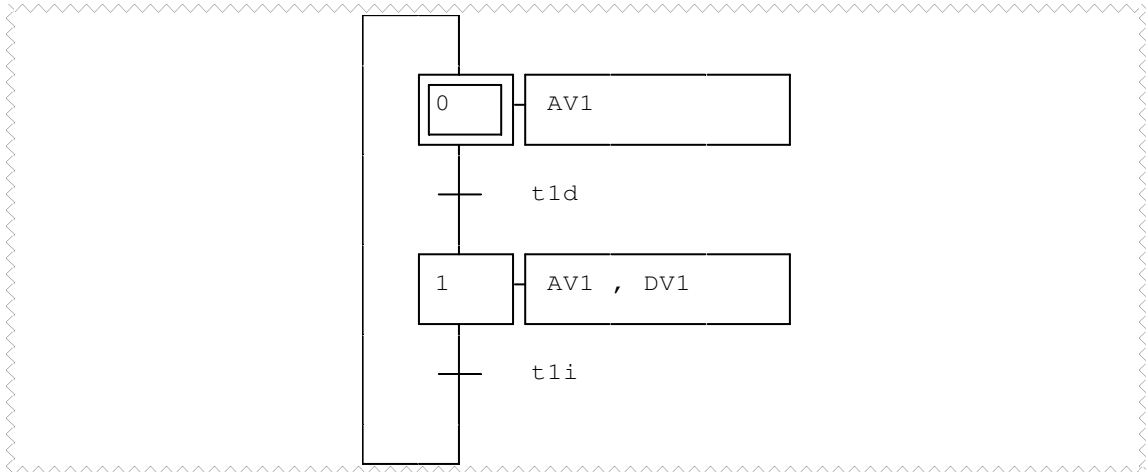
Grafcet line writing involves combined steps and transitions.

The example below illustrates a Grafcet line:

Conditions:

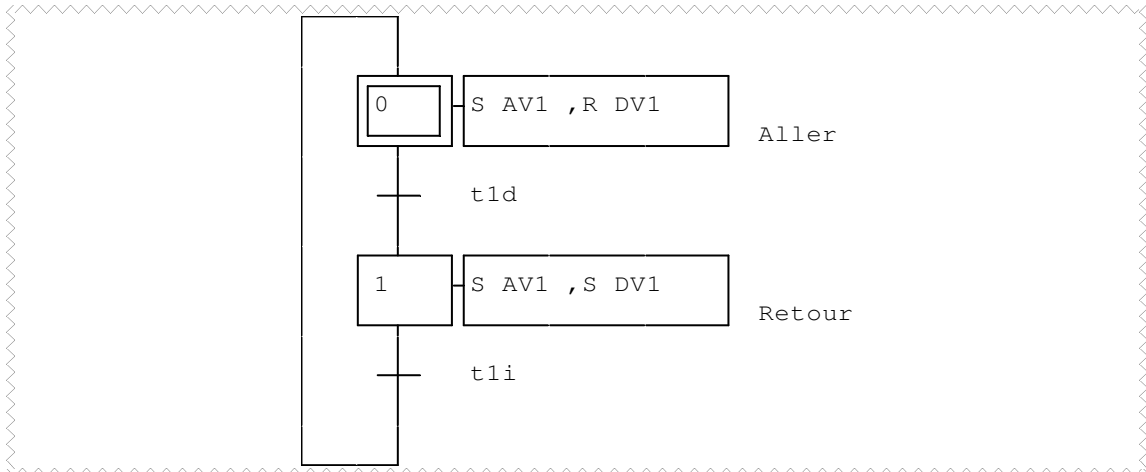
The locomotive needs to leave on track 1 towards the right, up to the end of the track. It returns in the opposite direction to the other end and starts again.

### Solution 1:



 examples\grafcet\simple1.agn

### Solution 2:

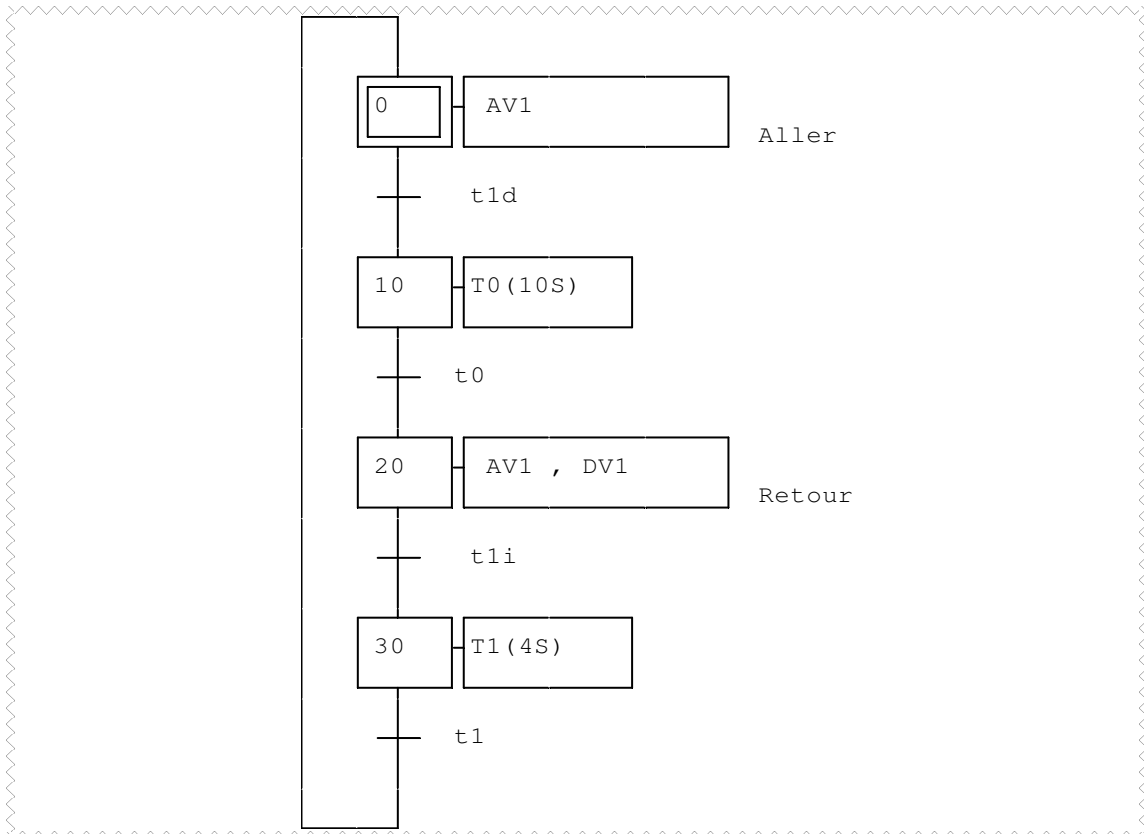


 example\grafcet\simple2.agn

The difference between the two solutions is that the first example uses « Assignment » actions and the second uses « Set to one » and «Reset ».

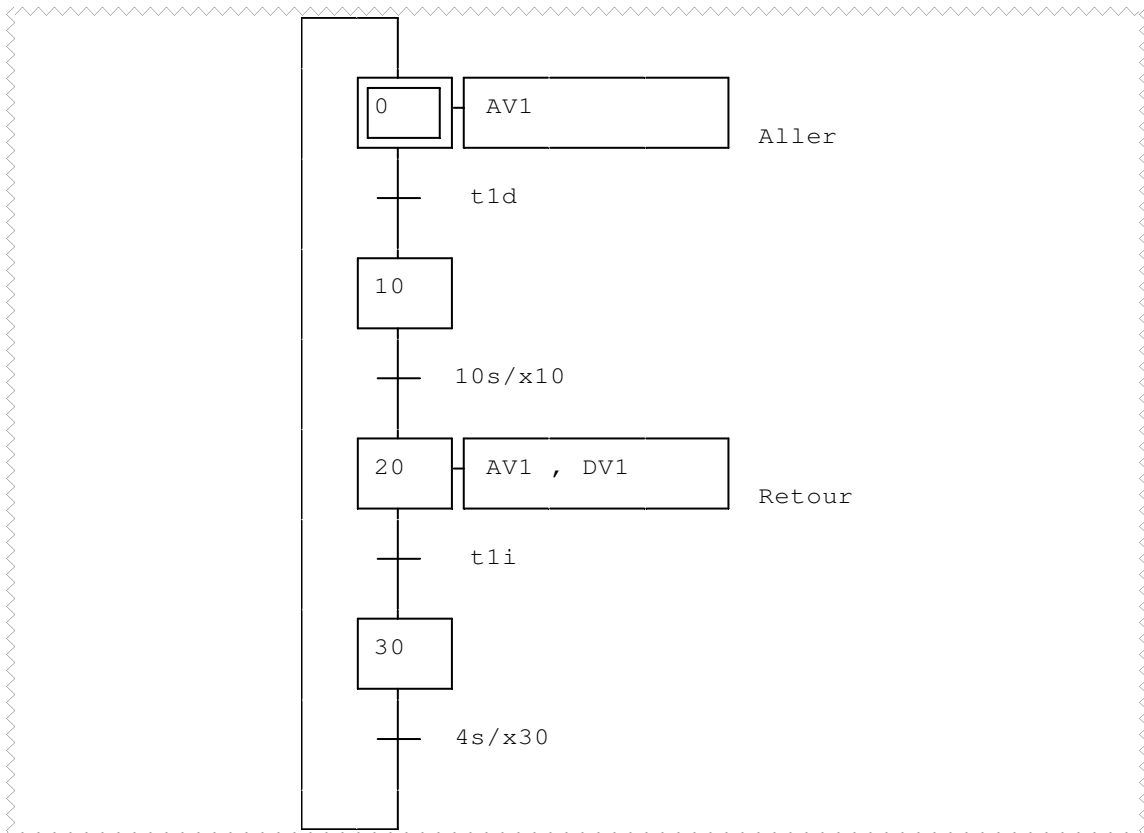
We change the conditions by setting a delay of 10 seconds when the locomotive arrives to the right of track 1 and a delay of 4 seconds when the locomotive arrives to the left of track 1.

### Solution 1:



 example\grafcet\simple3.agn

### Solution 2:

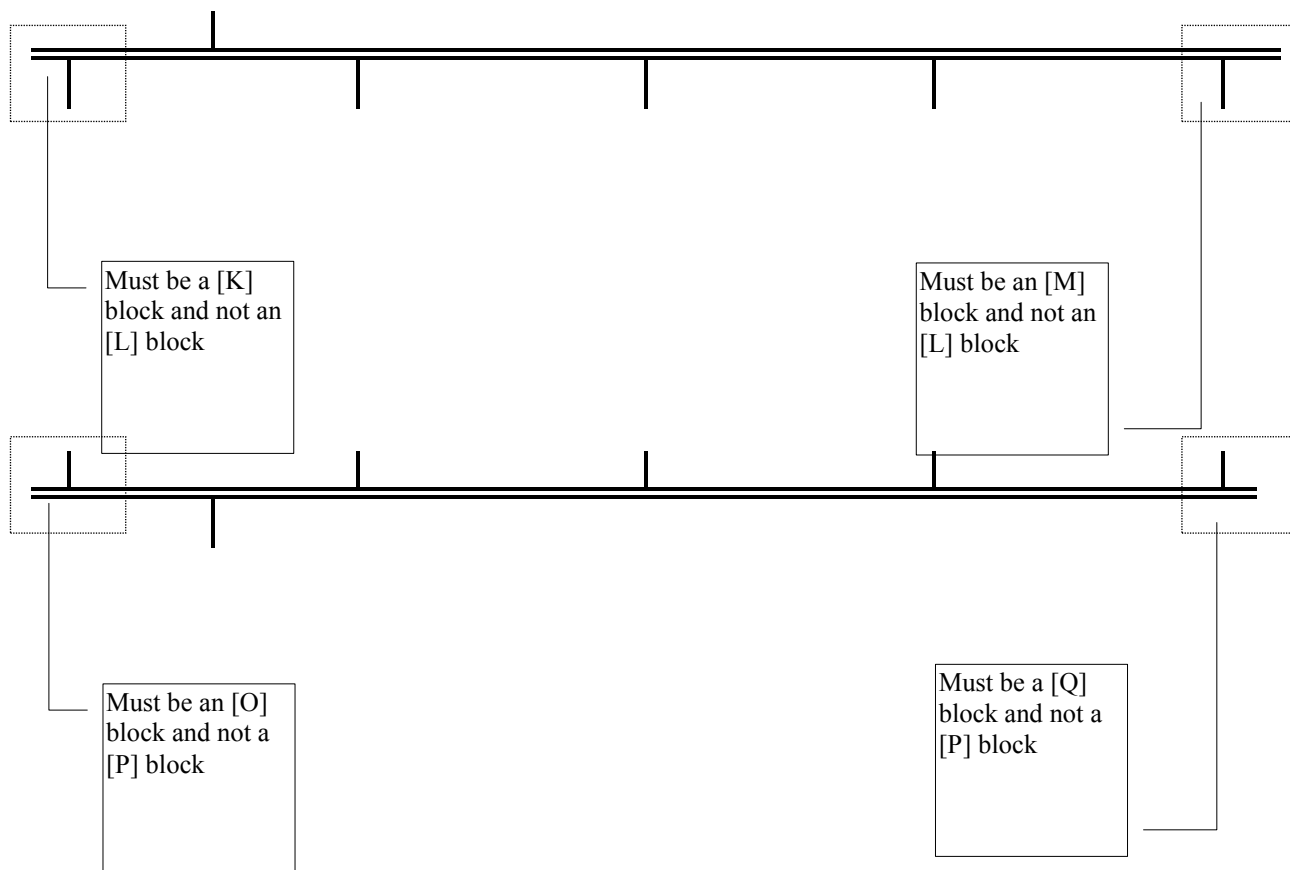


 example\grafcet\simple4.agn

The difference between example 3 and 4 is in the choice of syntax used to set the time delays. In terms of operation the result is identical.

### Divergence and convergence in « And »

Divergences in « And » can have n points. It is important to observe the use of the function blocks:

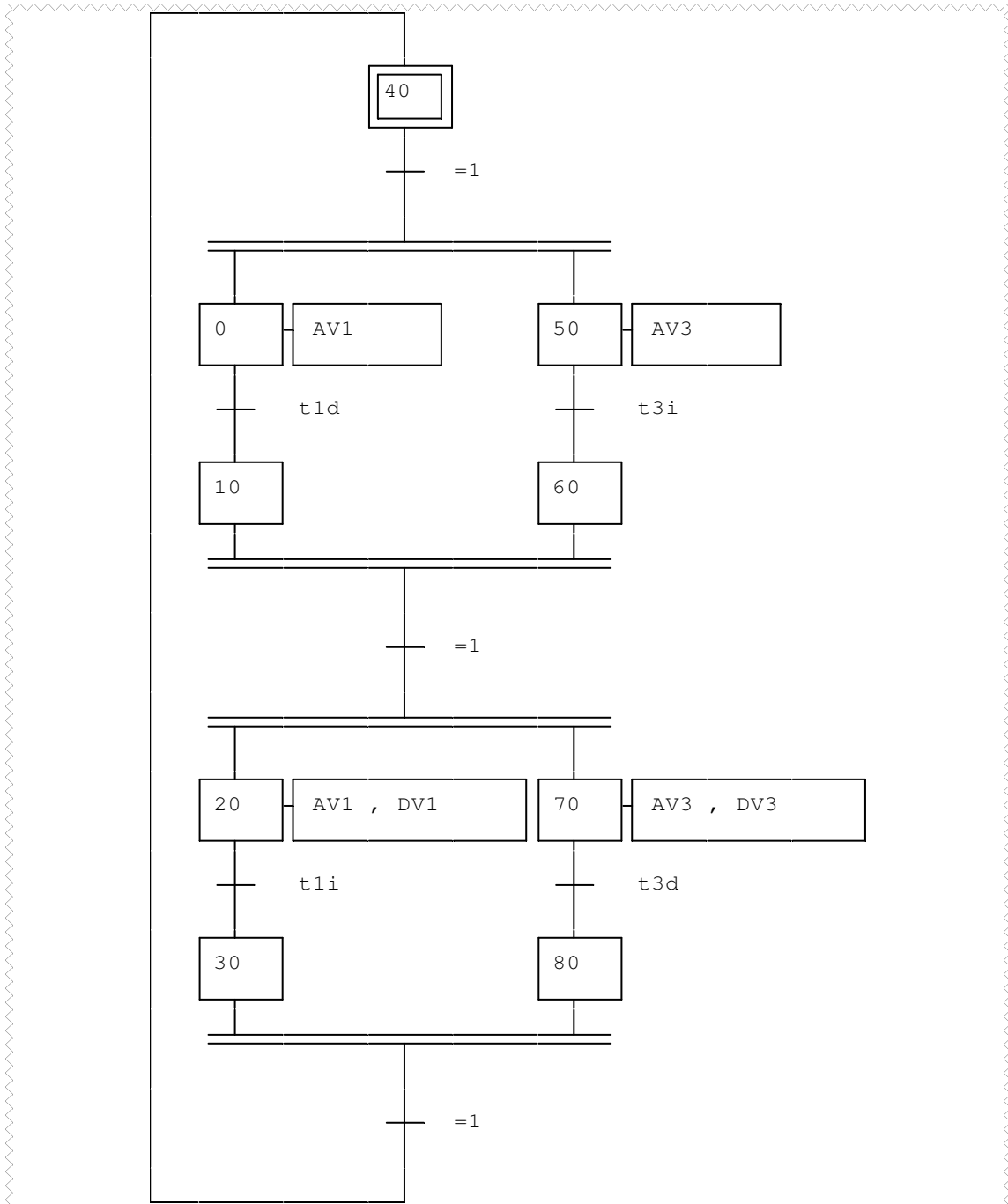


A description of the use of divergences and convergences in « And » follows.

Conditions:

We are going to use two locomotives: the first makes round trip journeys on track 1, the second on track 3. The two locomotives are synchronized (they wait at the end of the track).

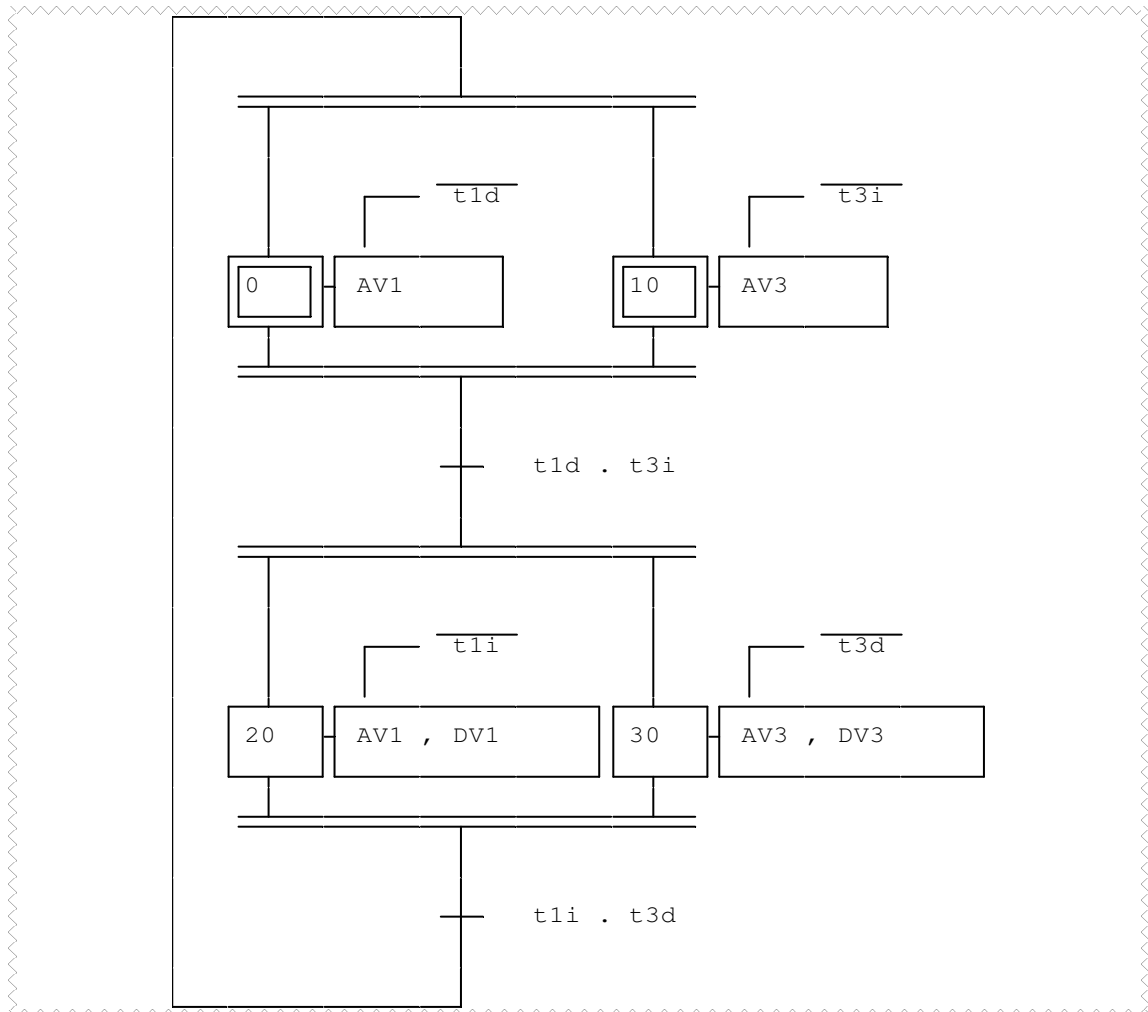
# Solution 1:



 example\grafcet\divergence et 1.agn



## Solution 2:

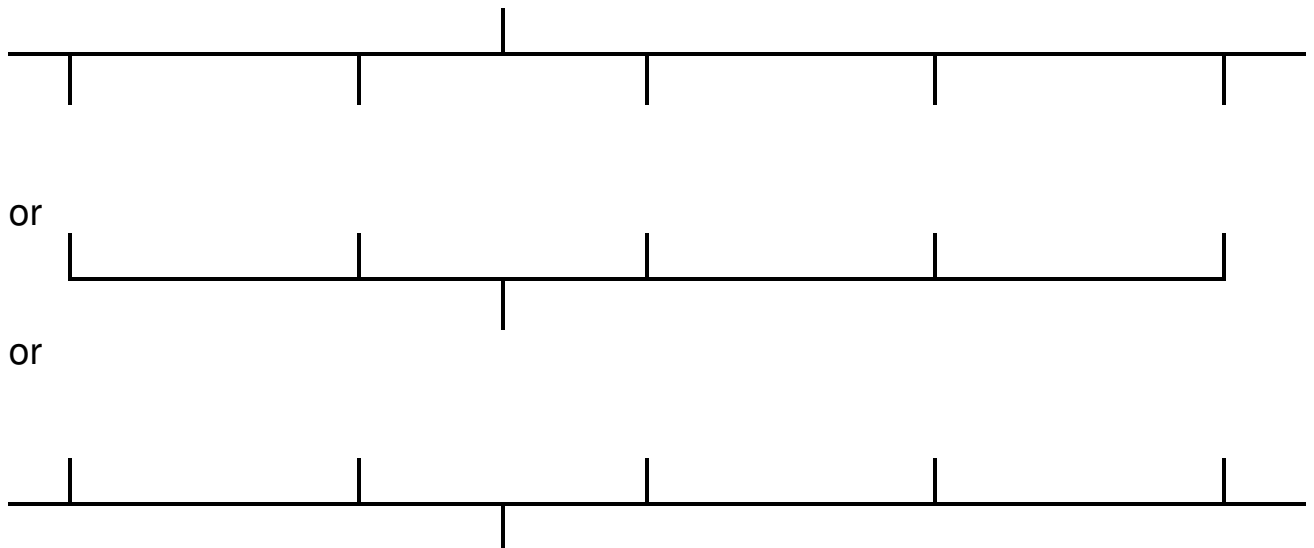


 example\grafcet\divergence and 2.agn

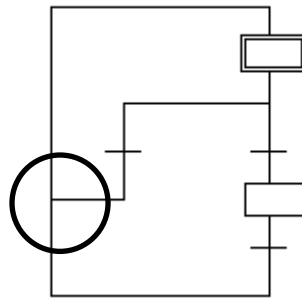
The two solutions are equivalent in terms of operation. The second is a more compact version which uses conditioned actions.

### Divergence and convergence in « Or »

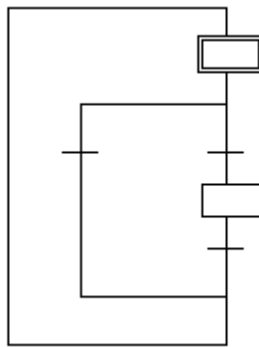
Divergences in « Or » can have n points. It is important to observe the use of the function blocks:



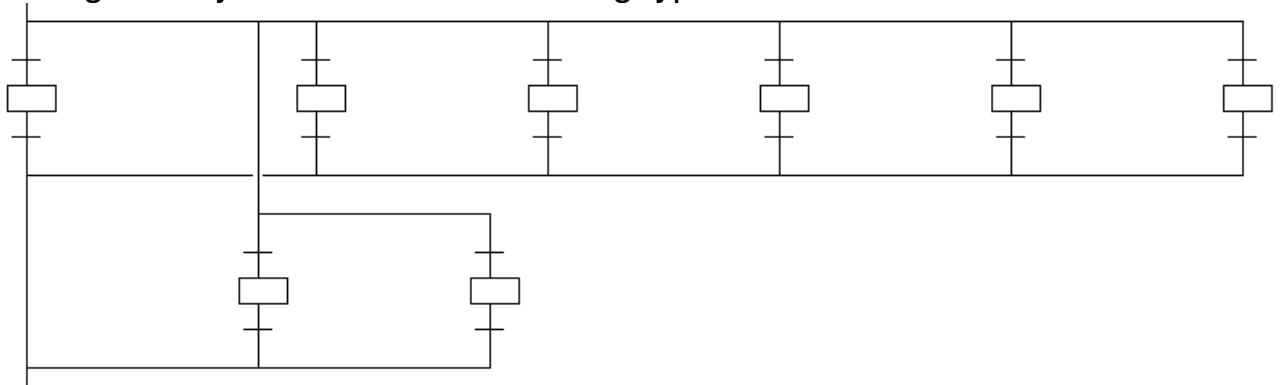
Divergences in « Or » must connect on descending links. For example:



incorrect, the right drawing is:



If the width of the page prevents you from writing a large number of divergences you can use the following type of structure:

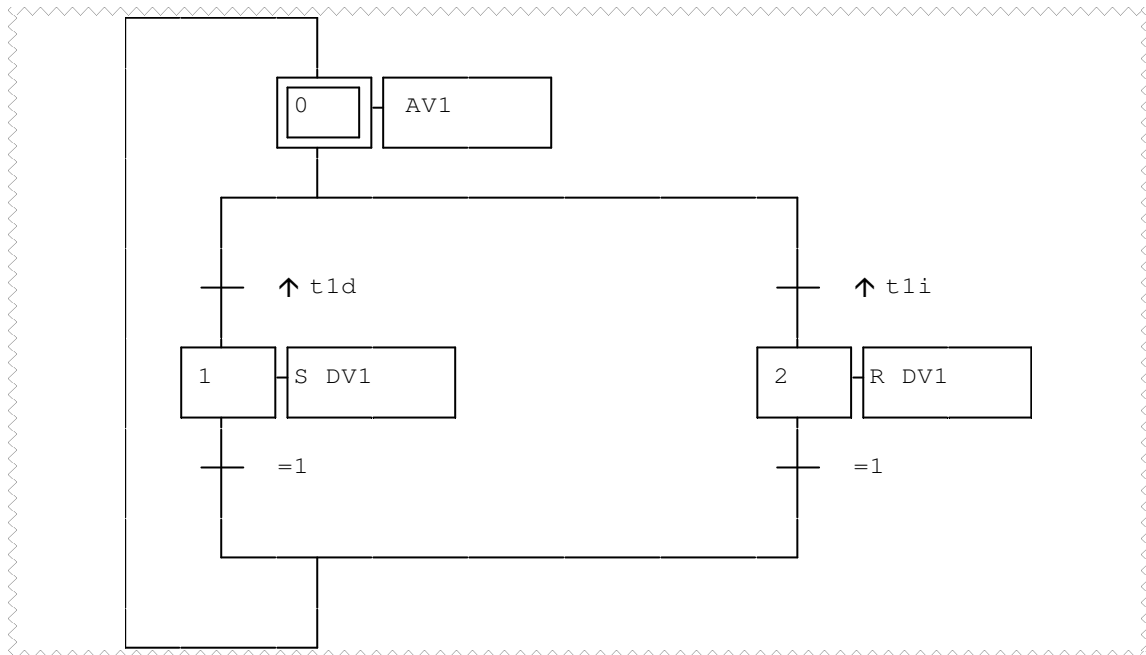


The following is an example to illustrate the use of divergences and convergences in « Or »:

Conditions:

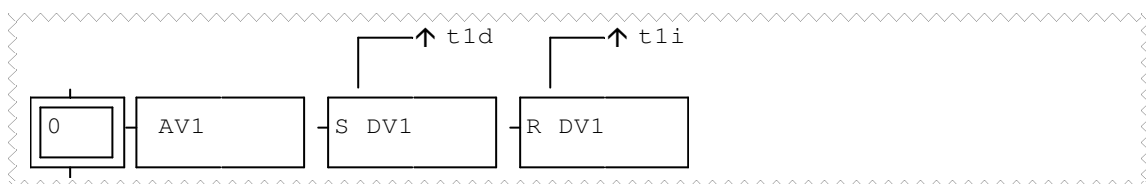
Let's look at the conditions for the first example in the chapter: roundtrip of a locomotive on track 1.

Solution:



 example\grafcet\divergence or.agn

This Grafcet could be restated in a step using conditioned actions as in this example:



 example\grafcet\conditional action.agn

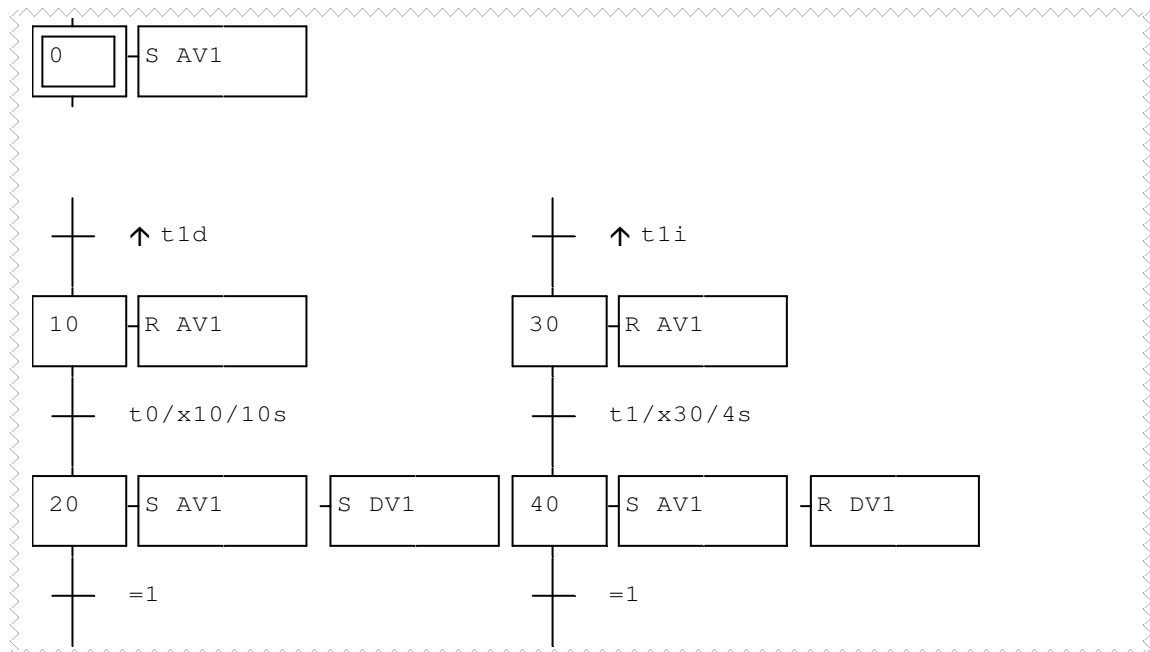
### Destination and source steps, destination and source transitions


The principles are illustrated in the examples below:

Conditions:

Let's look at the second example in this chapter: round trip of a locomotive on track 1 with a delay at the end of the track.

Solution:



 example\grafcet\destination and source steps.agn

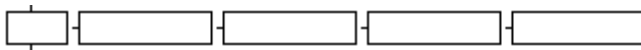
### Multiple actions, conditioned actions

We have already used multiple and conditioned actions in this chapter. The two principles are described in detail below.

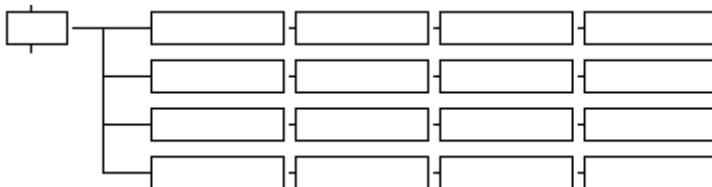
As described in the chapter dedicated to the compiler, multiple actions can be written in the same rectangle, by entering the character « , » (comma) as a separator.

When a condition is added to an action rectangle, all of the actions which continue in the rectangle are conditioned.

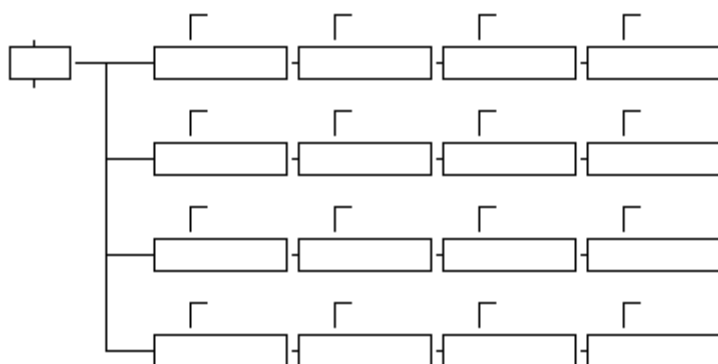
Multiple actions rectangles can be associated to a step.






another possibility:



Each of the rectangles can receive a different condition:

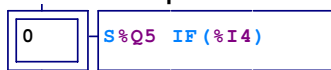


### Conditional actions, event-based actions



To design a conditional or event-based action, place the cursor over the action rectangle, right-click with the mouse and select “Conditional action” or event-based action from the menu. To document the condition on action, click on the element  or  or .

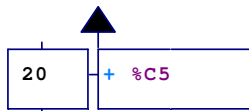
The IF (condition) syntax allows a condition on action to be written in the action rectangle.

For example:





### Actions on activation or deactivation of a step

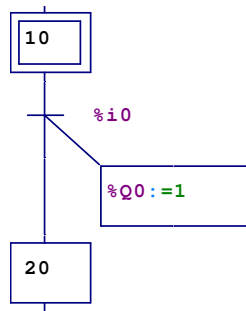
The  and  symbols make it possible to specify that the actions contained in a rectangle have to be performed once at the activation or deactivation of the step respectively. For example:



Increment counter 5 once at the activation of step 20.

### Actions on transition crossing

The  and  symbols make it possible to define actions on transition crossing. For example:



%Q0 will be activated on crossing the transition between steps 10 and 20.

## Synchronization

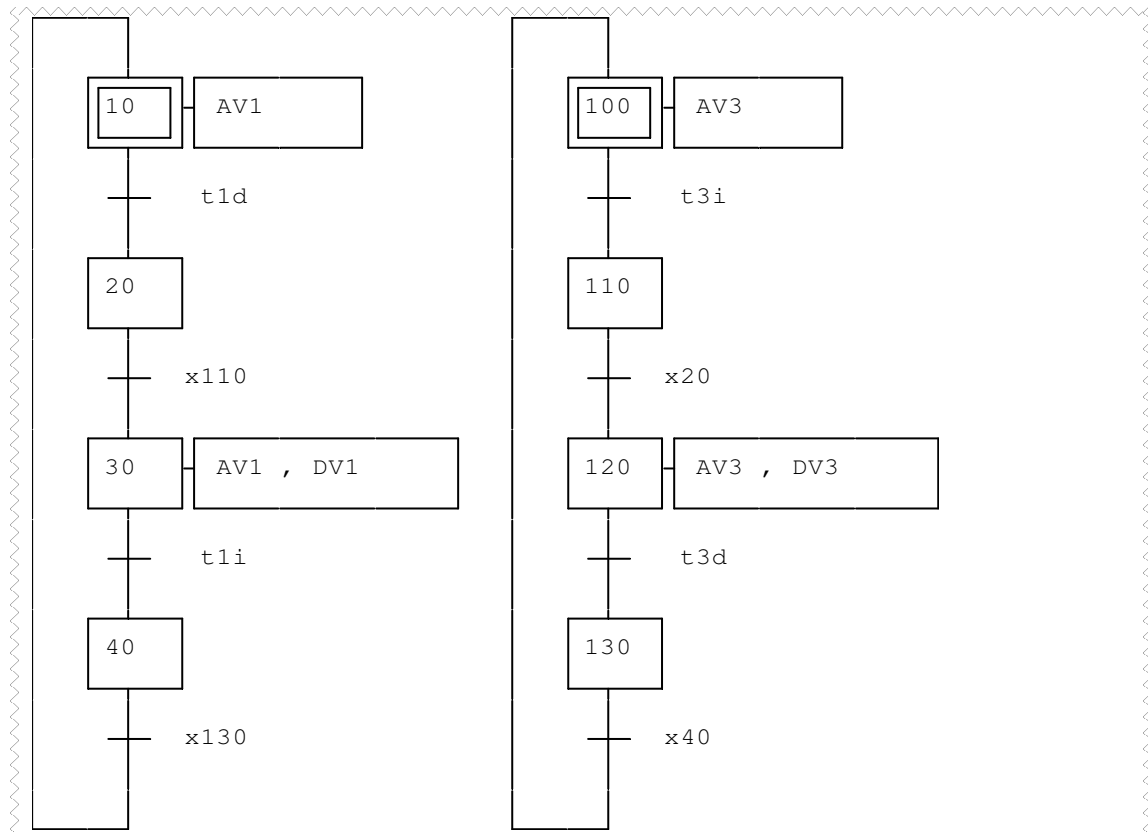
Let's return to a previous example to illustrate Grafsets synchronization.

Conditions:

Round trip of two locomotives on tracks 1 and 3 with a delay at the end of the track.

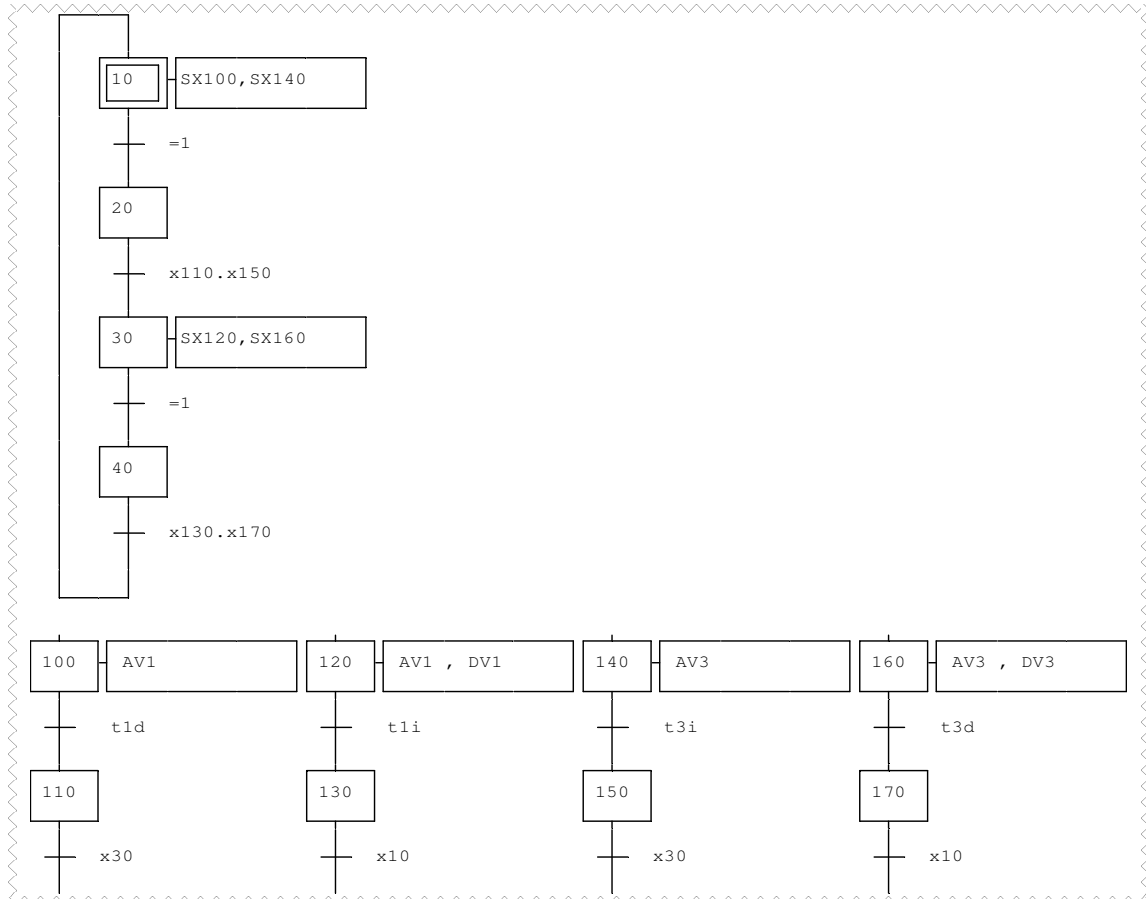
This example was used with a divergence in « And ».


Solution 1:



 example\grafcet\synchro1.agn

## Solution 2:



 example\grafcet\synchro2.agn

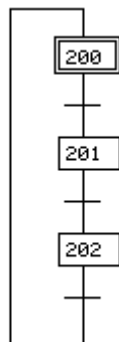
The second is an excellent example of how to complicate the simplest things for teaching purposes.

### Grafcet setting

The compiler regroupes the steps based on the links established within them. To draw a Grafcet, just refer to one of the steps making up that Grafcet.

It is also possible to draw all of the Grafcets present on a sheet by mentioning the name of the sheet.

For example:

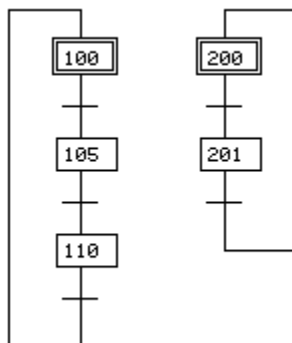


To draw a Grafcet we use Grafcet 200, Grafcet 201 or Grafcet 202. Thus the Grafcet of all the steps becomes a structured type variable, made up of n steps, each of these steps, being either active or idle. As we have seen, AUTOSIM divides the steps into independent groups. These groups can be regrouped, making it possible to consider them as a single Grafcet. To regroup multiple Grafcets, the compilation command « #G:g1,g2 » (command to be included in a comment) must be used. This command regroupes the Grafcets g1 and g2. Remember that the designation of a Grafcet is affected by mentioning the number of one of its steps.

Here is an example:

#G:105,200

this compilation command regroupes the two Grafcets:



Note: multiple « #G » commands can be used to regroup more than two Grafcets.

We are now going to describe the useable setting orders. They are simply written inside the action rectangles as normal assignments. They also support the operator S(set to one), R(reset), N(complement assignment) and I(Inversion) as well as conditional actions.

### Grafcet setting according to a list of active steps

Syntax:

« F<Grafcet>:{<list of active steps>} »

or

« F/<sheet name>:{<list of active steps>} »



The Grafcet/s thus designated will be set to the state established for the list of active steps if they are within braces. If multiple steps need to be active they need to be separated with a « , » (comma). If the Grafcet/s need to be set to an idle state (not active step) then no step should be present within the braces.

The number of steps may be preceded by an « X ». It is also possible to associate a symbol to the name of a step.

Examples:

« F10:{0} »

set all the steps of Grafcet 10 to 0 except step 0 which will be set to 1.

« F0:{4,8,9,15} »

sets all the steps of Grafcet 0 to 0 except steps 4,8,9 and 15 which will be set to 1.

« F/normal run:{} »

sets all the Grafcets on the « normal run » sheet to an idle state.

### Memorization of a Grafcet state

Current state of a Grafcet:

Syntax:

« G<Grafcet>:<bit N<sup>o</sup>> »

or

« G/<sheet name>:<bit N<sup>o</sup>> »

This command memorizes the state of one or more Grafcets in a series of bits. It is necessary to reserve a storage space for the state of the Grafcet/s (one bit per step). These storage bits must be consecutive. You must use the #B command to reserve a linear bit space.

The step number designating the Grafcet can be preceded by an « X ». It is also possible to associate a symbol to a step name. The bit number can be preceded by « U » or « B ». A symbol can be associated to the first bit of the state storage area.

Particular Grafcet state:

Syntax:

« G<Grafcet>:<Bit N<sup>o</sup>> {list of active steps} »

or

« G/<sheet name>:<Bit N<sup>o</sup>> {list of active steps} »

This command memorizes the state set for the list of active steps applied to the specified Grafcets starting with the indicated bit. Also here it is necessary to reserve a sufficient number of bits. If an idle situation needs to be memorized then no steps should appear between the braces.

The step number can be preceded by an « X ». It is also possible to associate a symbol to a step name. The bit number can be preceded by « U » or « B ». A symbol can be associated to the first bit of the state storage area.

Examples:

« G0:100 »

memorizes the current state of Grafcet 0 starting from U100.

« G0:U200 »

memorizes the idle state of Grafcet 0 starting from U200.

« G10:150{1,2} »

memorizes the state of Grafcet 10, in which only steps 1 and 2 are active, starting from U150.

« G/PRODUCTION:\_SAVE PRODUCTION STATE\_ »

memorizes the state of the Grafcets on the « PRODUCTION » spreadsheet in the \_SAVE PRODUCTION STATE\_ variable.

## Setting a Grafcet from a memorized state

Syntax:

« F<Grafcet>:<Bit N°> »

or

« F/<sheet name>:<Bit N°> »

Sets the Grafcet/s with a memorized state to start from the specified bit.

The step number designated by the Grafcet can be preceded by an « X ». It is also possible to associate a symbol to a step name. The bit number can be preceded by « U » or « B ». A symbol can be associated to the first bit of the state storage area.

Example:

« G0:100 »

memorizes the current state of Grafcet 0

« F0:100 »

and resets that state

## Fixing a Grafcet

Syntax:

« F<Grafcet> »

or

« F/<sheet name> »

Fixes a Grafcet/s: no evolution of these is permitted.

Example:

« F100 »

fixes Grafcet 100

« F/production »

fixes the Graficets on the « production » sheet

An illustration of setting is shown in the example below.

Conditions:

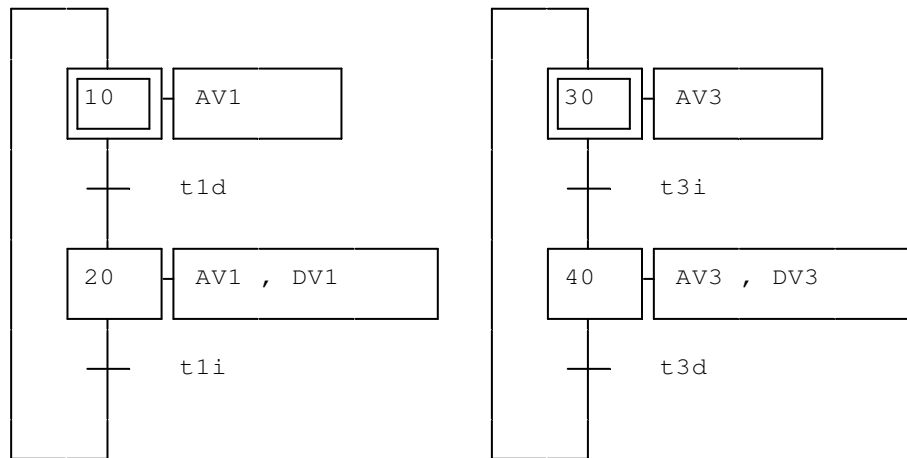
Let's look at a previous example: the round trip of two locomotives on tracks 1 and 3 (this time with no delay between the locomotives) and let's add an emergency stop. When the emergency stop is detected all the outputs are cleared. When the emergency stop disappears the program should start where it stopped.

## Solution 1:

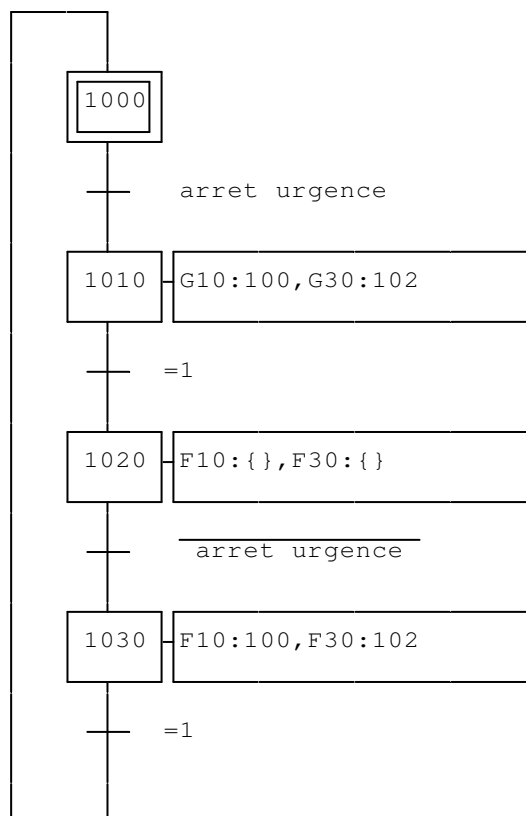
```
#B104      réserve 4 bits pour la mémorisation de l'état des Grafcet.
```

```
locomotive 1
```

```
locomotive 2
```



```
gestion de l'arrêt d'urgence
```

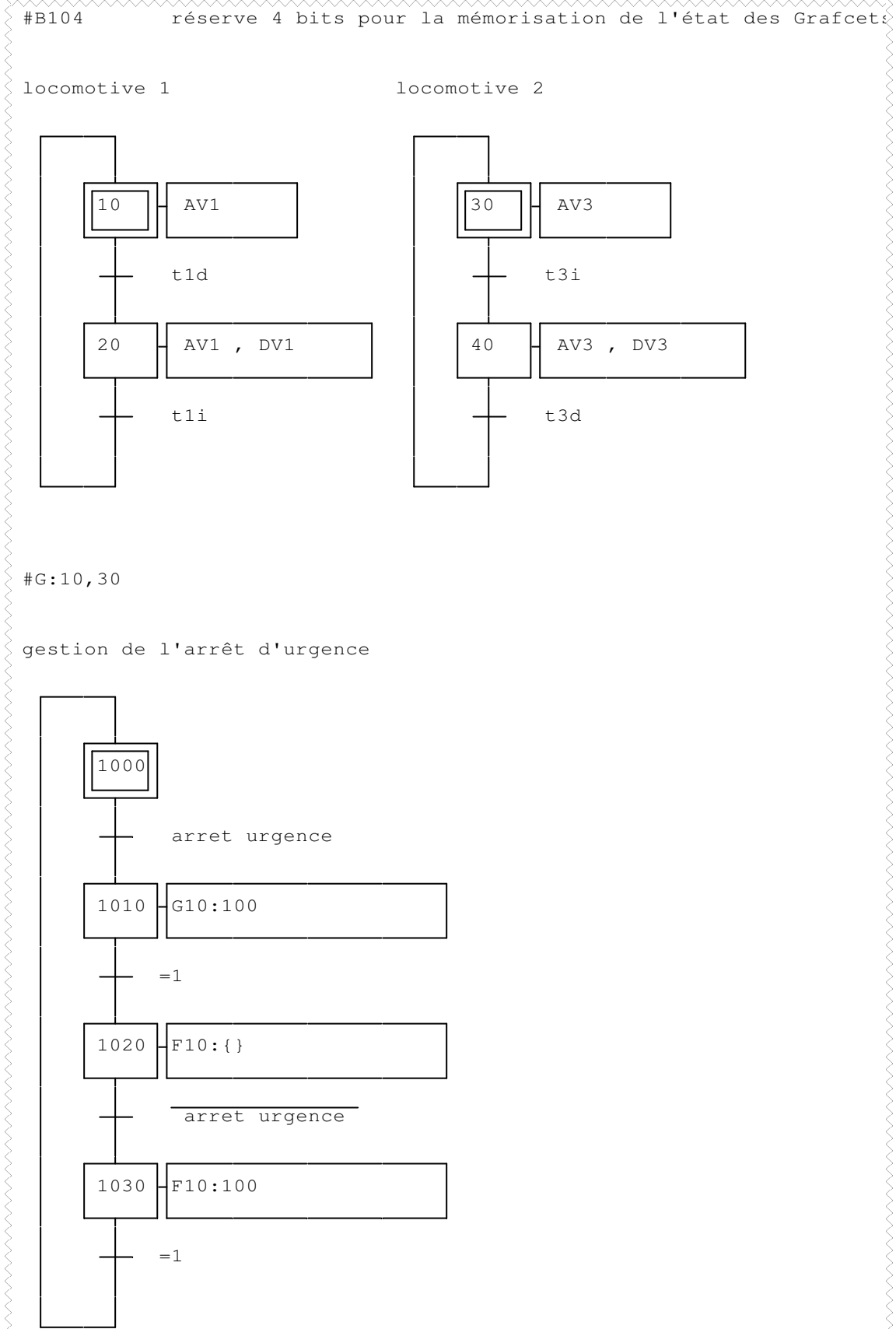


 example\grafcet\force1.agn

Note the use of command #B104 which makes it possible to reserve four consecutive bits (U100 to U103) to memorize the state of the two Grafkets. « \_emergency stop\_ » was associated to a bit (U1000). Its

state can thus be modified starting from the environment by clicking below when the dynamic display is active.

## Solution 2:



 example\grafcet\force2.agn

This second solution shows the use of the compilation command « #G » which makes it possible to regroup the Grafkets with setting command.

### Grafket forcings (60848 standard)

This standard defines the forcing orders in double action rectangles. Forcing actions are executed when the associated condition, step or logical diagram, is true. Conditions can be added on the double action rectangles: condition on action, event-based action, action on activation or deactivation.

### Forcing a Grafket according to a list of active steps

The syntax is:

G<grafket to be forced>{<list of steps to be forced when true>}

The step(s) included in the list are forced to true, the other to false. An empty list of steps causes all the steps to be forced to false.

Example:

```
G10 { 20, 30 }
```

Here 10 represents the Grafket to be forced: Grafket containing step 10.

Another example:

```
Gfolio à forcer { 100, 200, 300 }
```

Force all of the Grafkets on the sheet named “sheet to be forced”, with steps 100, 200 and 300 being set to true and the others set to 0.

### Forcing a Grafket to its initial state

The syntax is:

G<grafket to be forced>{INIT}

The Grafket(s) are forced to their initial state

Example:

```
G10 { INIT }
```

### Freezing a Grafket

The syntax is:

G<grafket to freeze>{\*}

Example:

```
G10 { * }
```

## Macro-steps

AUTOSIM implements macro-steps.

Additional information is given below:

A macro-step MS is the single representation of single group of steps and transitions called « MS expansion».


A macro-step obeys the following rules:

- ⇒ an MS expansion involves a special step called **input step** and a special step called **output step**.
- ⇒ the input step has the following property: complete clearing of a transition upstream from the macro-step, it activates the input step of its expansion.
- ⇒ the output step has the following property: it is involved in the validation of transitions downstream from the macro-step.
- ⇒ if outside the transitions upstream and downstream from the MS, there is no input structural connection, on one side with a step or transition of the MS expansion and on the other side, a step or a transition is not part of MS.

The use of a macro-step with AUTOSIM is set as follows:

- ⇒ the expansion of a macro-step is a Grafcet if it is on a distinct sheet,
- ⇒ the input step of the macro-step expansion must bear the number 0 or the reference Exxx, (with xxx = any number),
- ⇒ the output step of a macro-step expansion must bear the number 9999 or the reference Sxxx, with xxx = any number,
- ⇒ aside from these two requirements, a macro-step expansion can be any Grafcet and as such can contain macro-steps.

### 0.0.0.1. How can a macro-step be set ?

The symbol  must be used. To obtain this symbol, click on an empty space on the sheet and select « Add .../Macro-step » from the menu. To open the menu click on the bottom of the sheet with the right side of the mouse.

To set a macro-step expansion, create a sheet, designate the expansion and assign the sheet properties (by clicking with the right side of the mouse on the name of the sheet in the browser). Record the type of sheet on «Macro-step expansion » and the number of the macro-step.

In run mode it is possible to display a macro-step expansion. To do so place the cursor on the macro-step and click on the left side of the mouse.

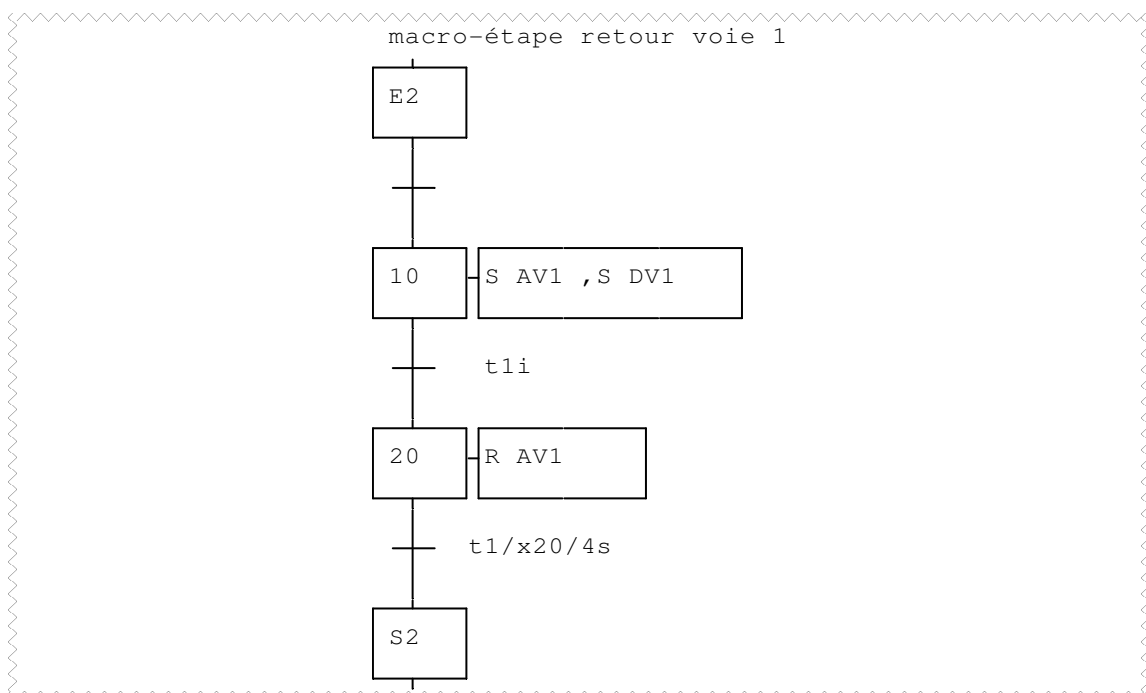
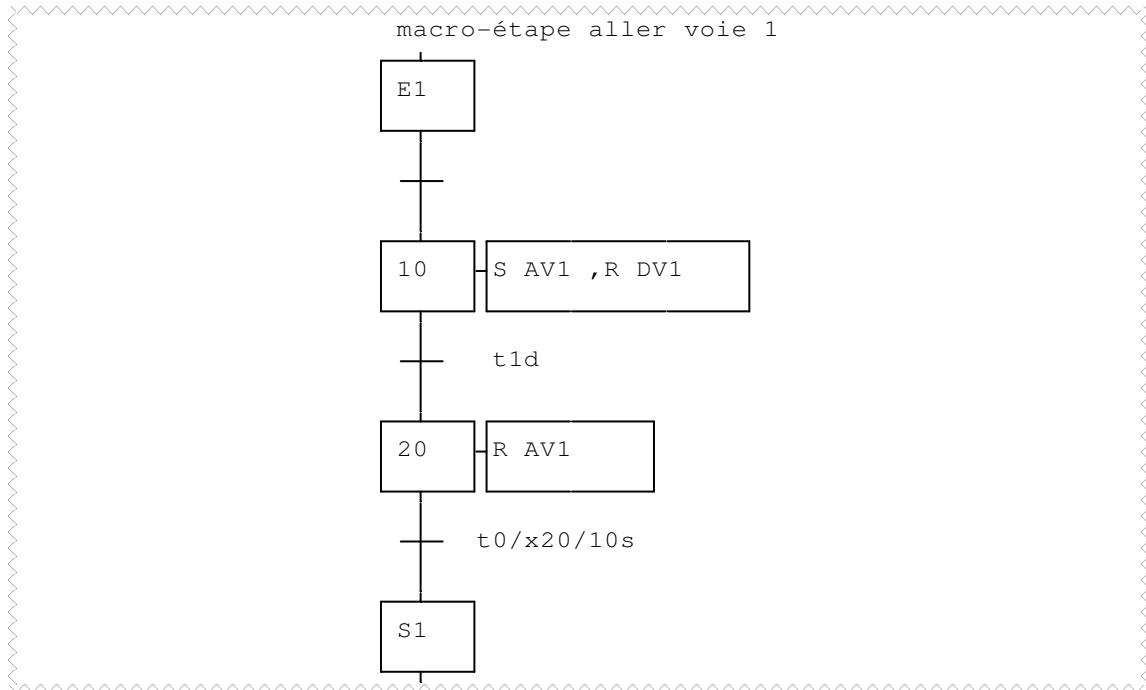
#### Notes:

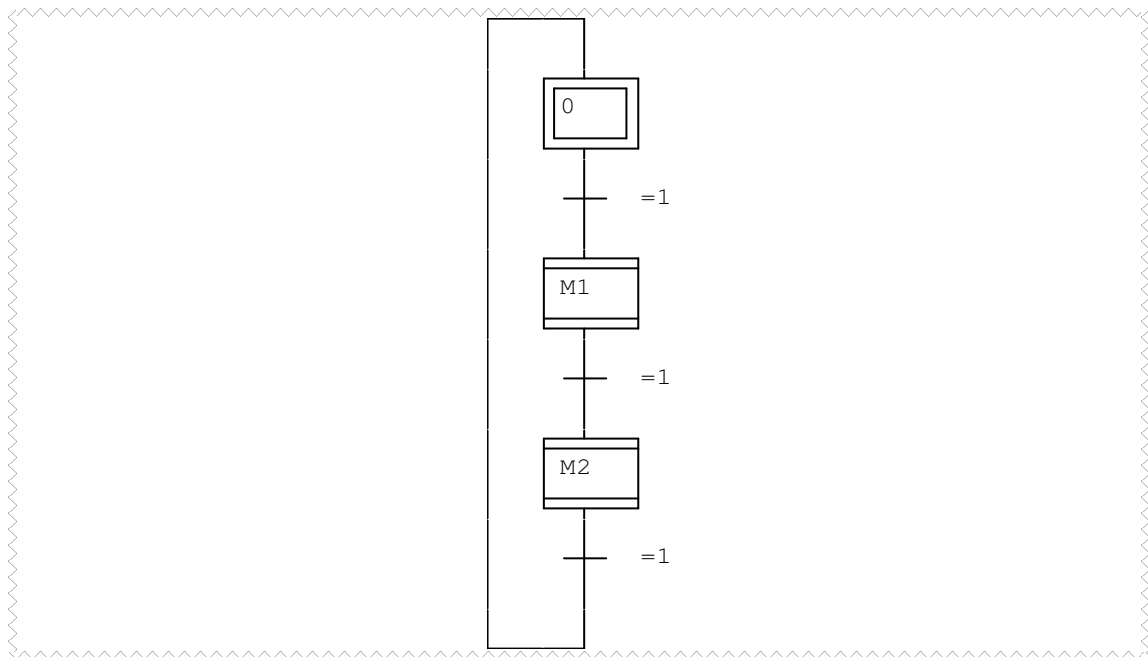
- ⇒ user steps and bits used in a macro-step expansion are local, this means that they have no connection with the steps and bits of other Graficets. All the other types of variables do not have this characteristic: they are common for all levels.
- ⇒ if an area of bits needs to be used in an overall method it is necessary to state this using the command « #B ».
- ⇒ assignment of non-local variables for different levels or different expansions is not managed by the system. In other words, it is necessary to use the assignments « S » « R » or « I » to ensure that the system operates correctly..

Let's use one of our previous examples to illustrate the use of macro-steps: a round trip voyage of a train on track 1 with a delay at the end of the track. We have broken down the legs of the trip into two separate macro-steps.



Solution:





 example\grafcet\macro steps.agn

## Encapsulating steps



Introduced in standard 60848, encapsulating steps are an evolution of the ideas proposed in macro-steps.

Using encapsulating steps under AUTOSIM is defined as follows:

⇒ the encapsulation is located in a separate sheet.

## How do you define an encapsulating step?




The  or  symbol have to be used. To place this symbol, right-click with the mouse on an empty part of the sheet and select “Plus.../Encapsulating step” in the contextual menu.

## How do you define an encapsulation?

To define the encapsulation, create a sheet, design the encapsulation and modify the properties of the sheet (by right-clicking with the mouse on the name of the sheet in the browser). Set the sheet type to “Encapsulation” as well as the encapsulating step number.

✱

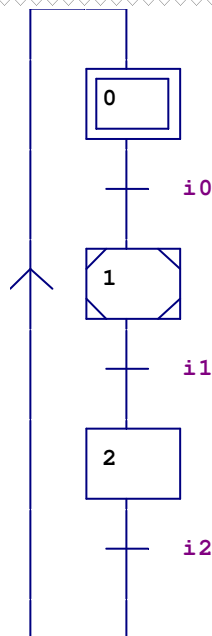
The  symbol allows the initial state to be defined for an encapsulation.

An encapsulation can be viewed in execution mode. To do this, you need to place the cursor over the encapsulating step and left-click with the mouse.

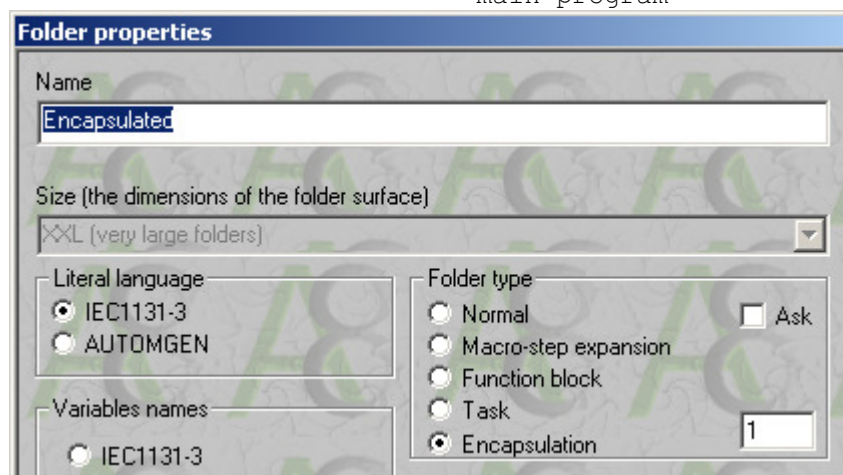
### Notes:

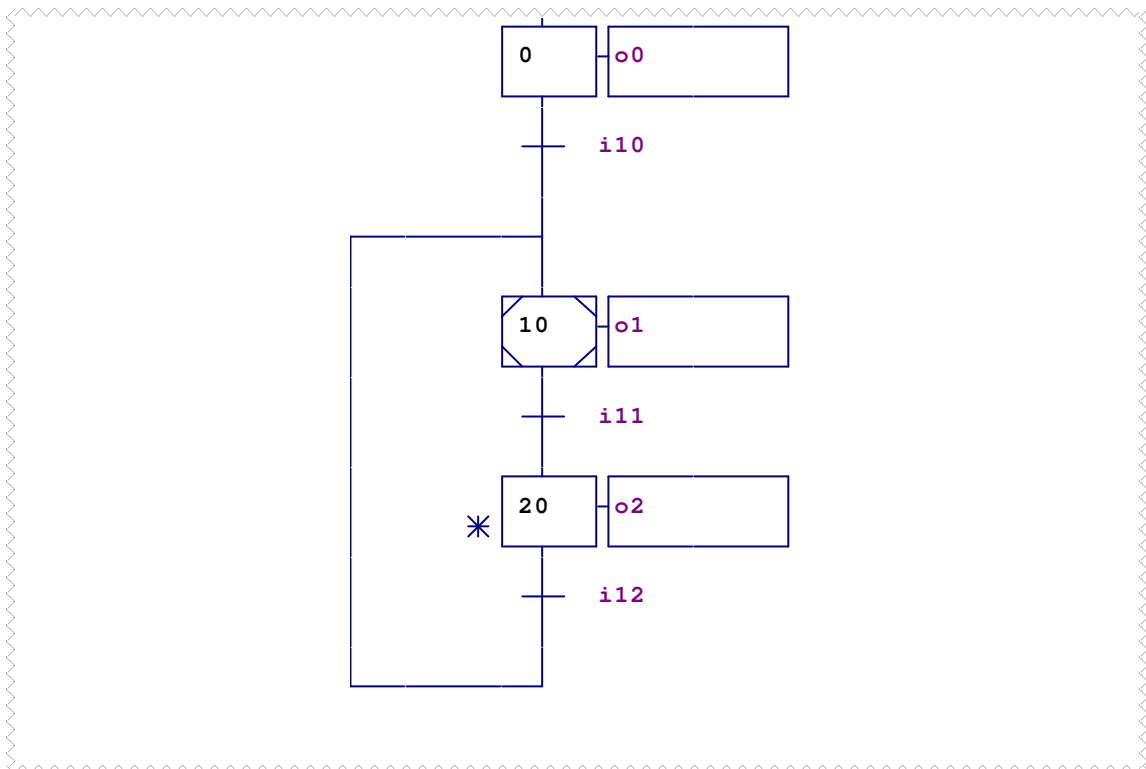
- ⇒ The User bits and steps used in an encapsulation are local, i.e. they are not related to the bits and steps of other Grafcet levels. This is not the case for all the other types of variable: they are common to all the levels. You can, for example, use word bits as global variables.
- ⇒ The encapsulated steps can be embedded.
- ⇒ The Xn/Xm or %Xn/%Xm syntax allows you to reference step m contained in the encapsulation associated to step n.

### Example :



main program

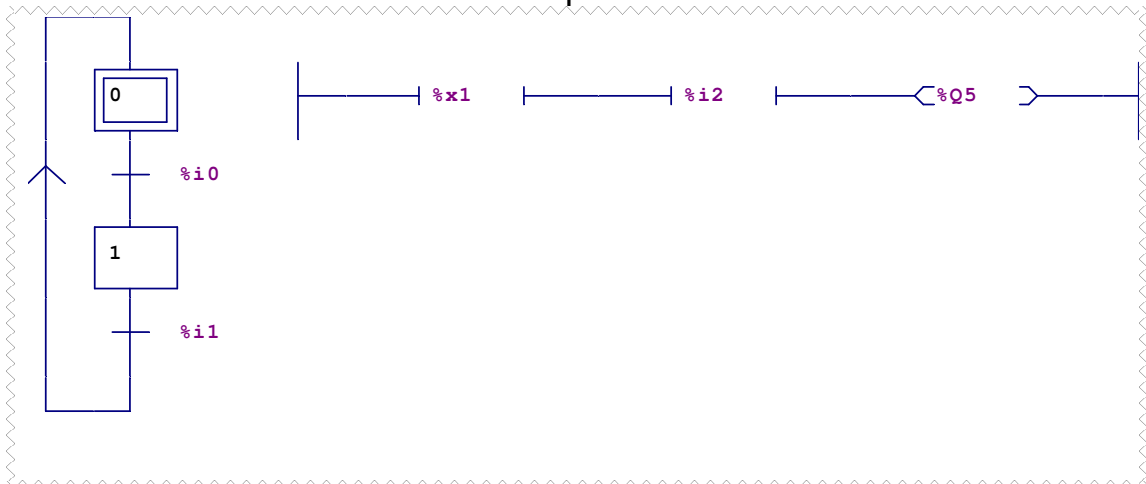





 examples\grafcet\encapsulation 2.agn

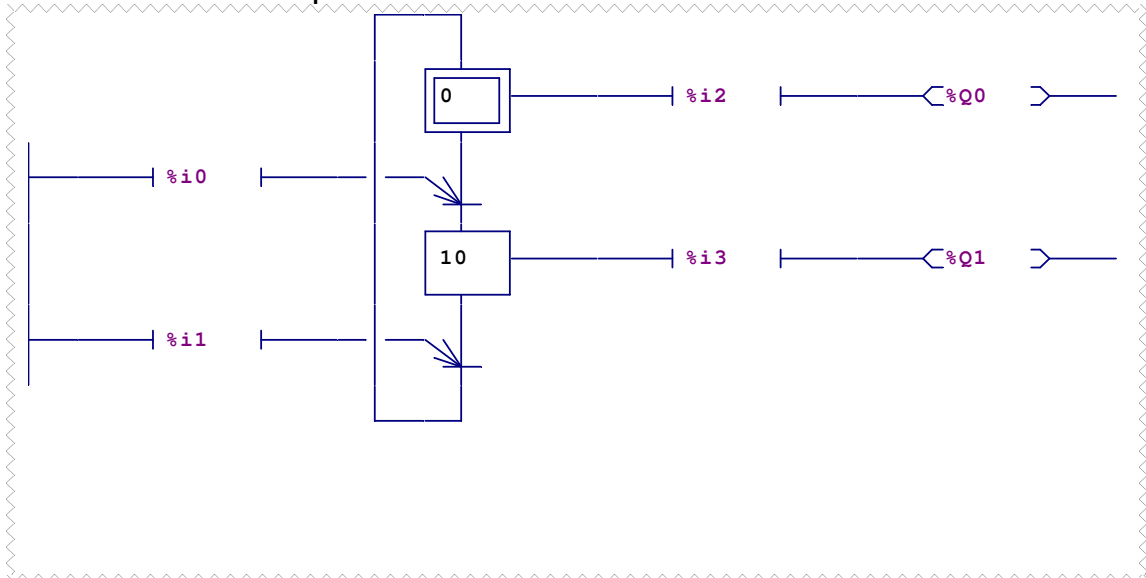
## Grafcet / Ladder and Grafcet / Flow chars links

Links can be defined with Grafcet step variables:



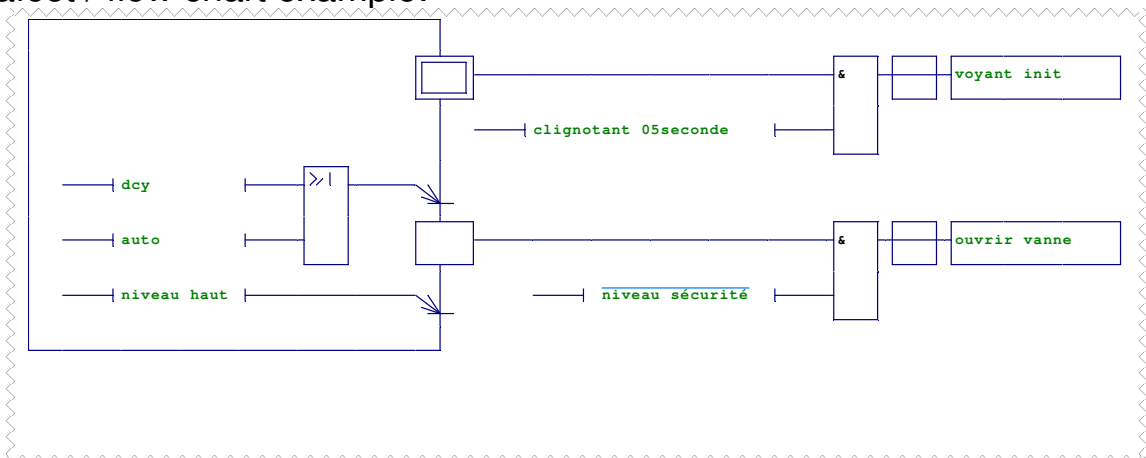
GRAFCET++: the  block can be used to wire a transition like a ladder circuit. The Grafcet steps can be wired as the start of a contact.

Grafcet / ladder example:



 examples\grafcet\grafcet++2.agn

Grafcet / flow chart example:



 examples\grafcet\grafcet++3.agn

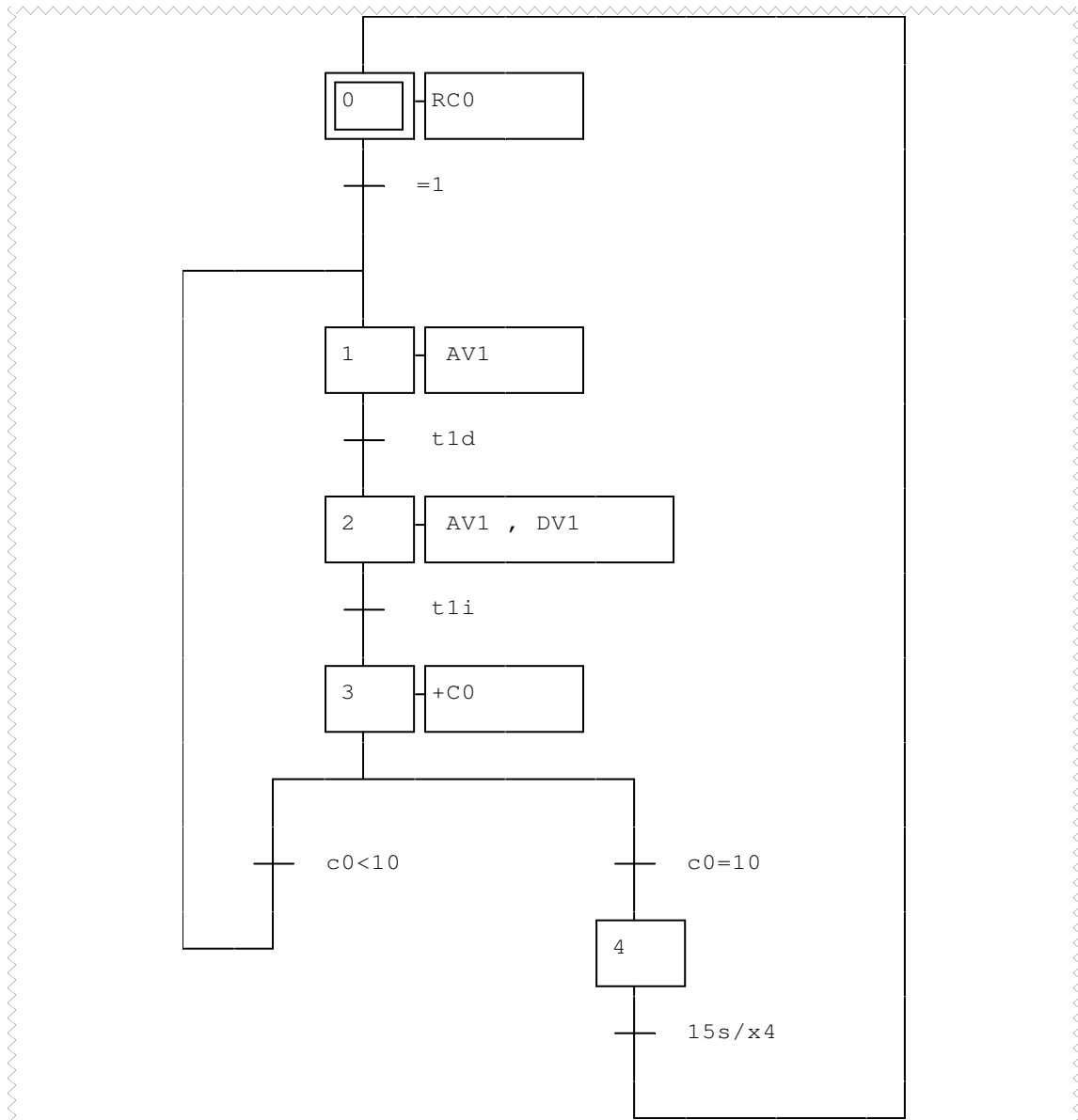
## Counters

We are going to use an example to describe the use of counters.

Conditions:

A locomotive must make 10 round trip journeys on track 1, stop for fifteen seconds and start again.

Solution:



 Example\grafcet\compteur.agn

## Gemma

AUTOSIM implements the Grafcet description of run mode management in a Gemma form. The main feature is an editing method open to the Grafcet mode. It is possible to go from the Grafcet editing mode to the Gemma editing mode. The translation of a Gemma into a Grafcet run mode management is therefore automatic and immediate..



## Creating a Gemma

To create gemma proceed as follows:

- ⇒ click on « Sheet » on the browser with the right side of the mouse and select the command « Add a new sheet»,
- ⇒ from the list of sizes select « Gemma »,
- ⇒ click on « OK »,
- ⇒ use the right side of the mouse to click on the sheet name created on the browser,
- ⇒ select properties « Proprieties » from the menu,
- ⇒ check « Display Gemma form ».

The window will contain a Gemma where all the links are gray. To validate a rectangle or a connection click on it with the right side of the mouse.

To edit the contents of a rectangle or the type of connection click on it with the left side of the mouse.

The contents of Gemma rectangles will be placed in the Grafcet action rectangles. The type of connection will be placed in the Grafcet transitions.

## Content of Gemma rectangles

Gemma rectangles can receive any action used by Grafcet. Because this involves setting a structure for managing run and stop modes, it is a good idea to use the lowest level setting orders for Grafcet, see chapter **¡Error! No se encuentra el origen de la referencia..**

## Obtaining a corresponding Grafcet

Check "Display Gemma form" again in sheet properties to call up a Grafcet representation. It is always possible to call up a Gemma representation because the Grafcet structure has not been changed. The transitions, the action rectangle contents and comments can be edited with automatic updating of Gemma.

### Deleting blank spaces in Grafcet

It is possible that the obtained Grafcet occupies more space than necessary on the page. The command « Change page layout » from the « Tools » menu makes it possible to eliminate all the unused spaces.



### Printing Gemma

When editing is in Gemma mode use the « Print » command to print the Gemma.

### Exporting Gemma

Use the « Copy to EMF format » in the « Editing » menu to export a Gemma to a vectorial form.

### Example of Gemma

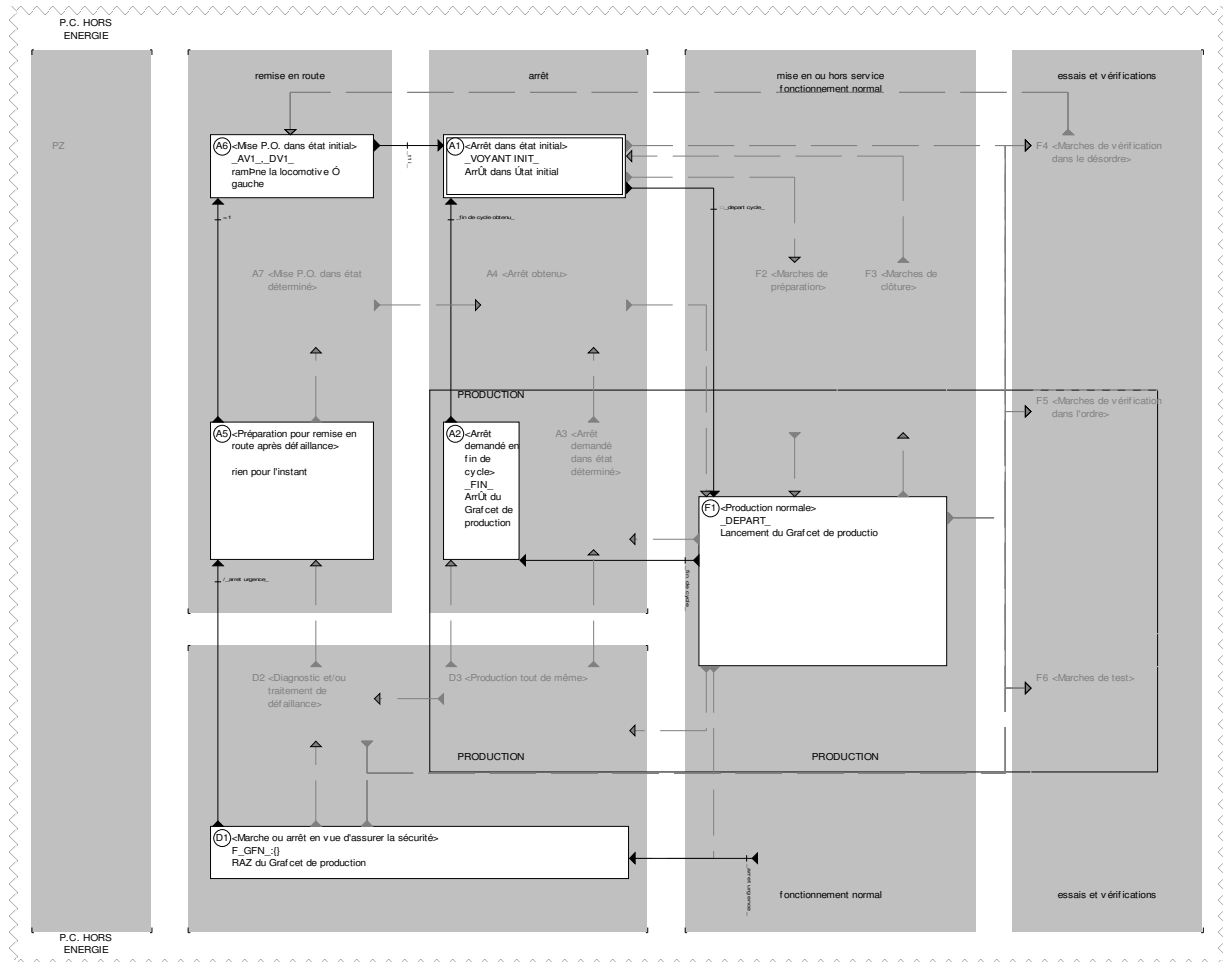
A description of how to use Gemma is below..

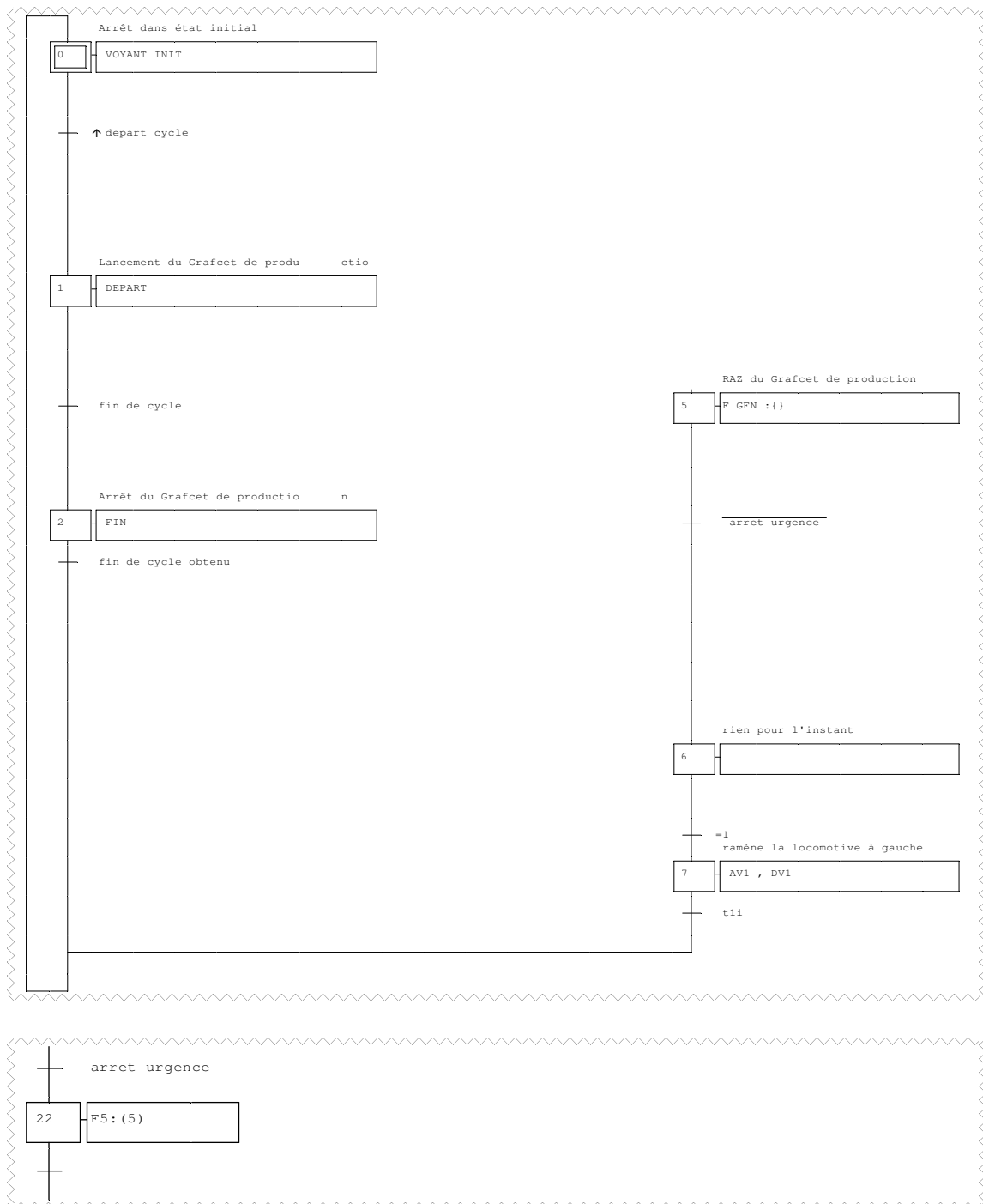
Conditions:

Imagine a panel with the following pushbuttons: « start cycle », « end cycle » and « emergency stop » a light « INIT ».

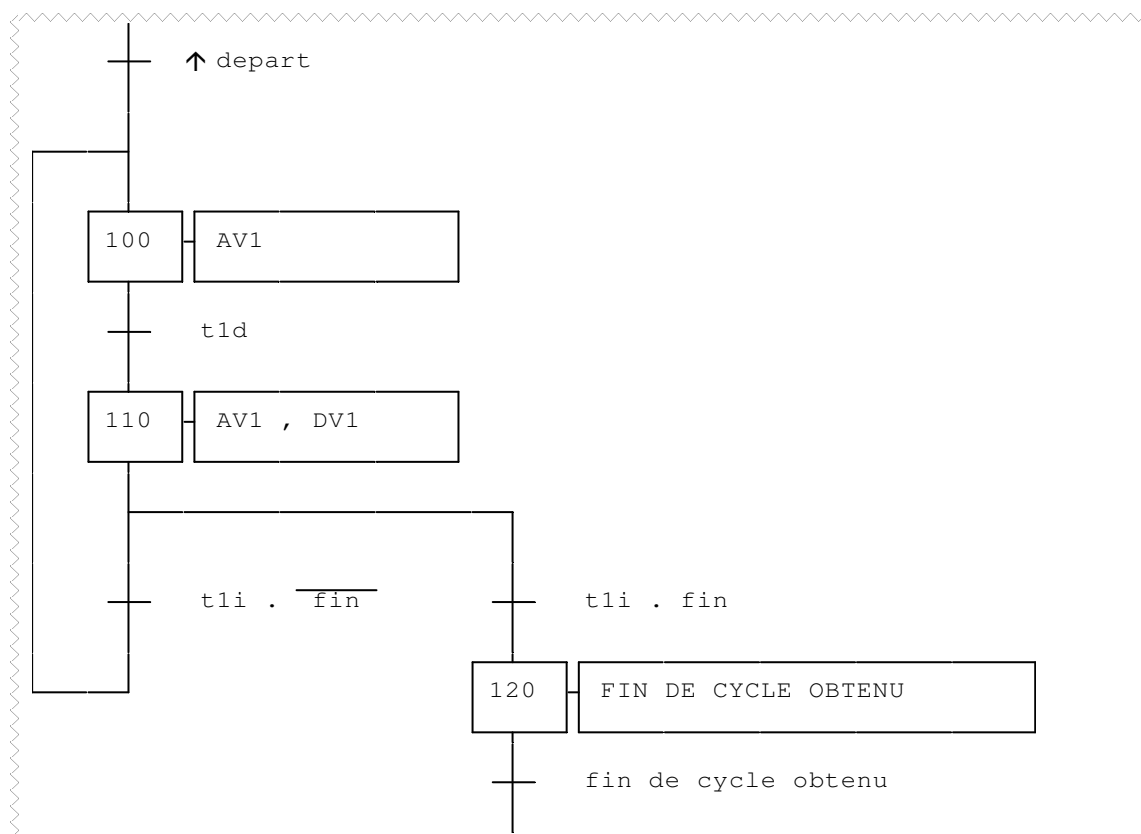
The main program will consist of a locomotive making round trip journeys on track 1.

# Solution:





(editing with Grafcet form)



 example\gemma\gemma.agn

## Ladder

Ladder language, also called contact model, is for graphically describing boolean equations. To create a logical function « And » it is necessary to write contacts in series. To write an « Or » function it is necessary to write contacts in parallel.



« And » function



« Or » function

The content of contacts must comply with the syntax established for the tests which is explained in the «Common elements» chapter of this manual.

The content of the coils must comply with the syntax established for the actions which is also explained in the «Common elements » chapter of this manual.

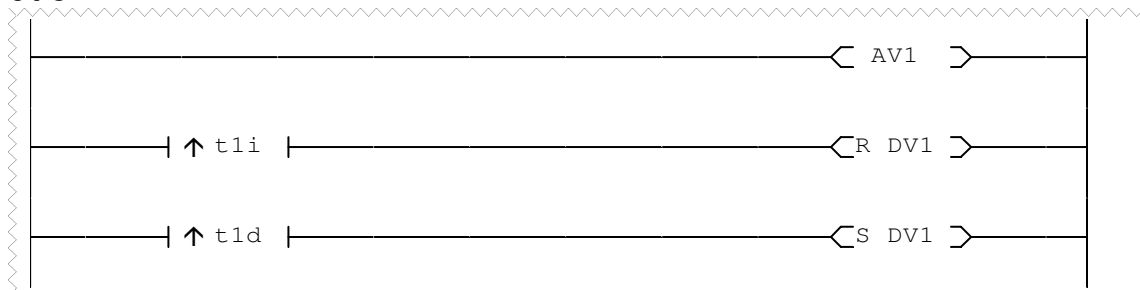
## Example of Ladder

Let's start with the simplest example.

Conditions:

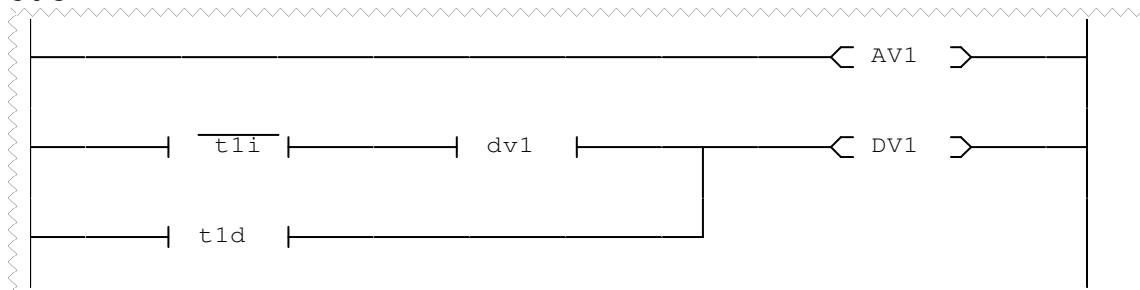
Round trip of a locomotive on track 1.

Solution 1:



 Example\ladder\ladder1.agn

Solution 2:



 Example\ladder\ladder2.agn

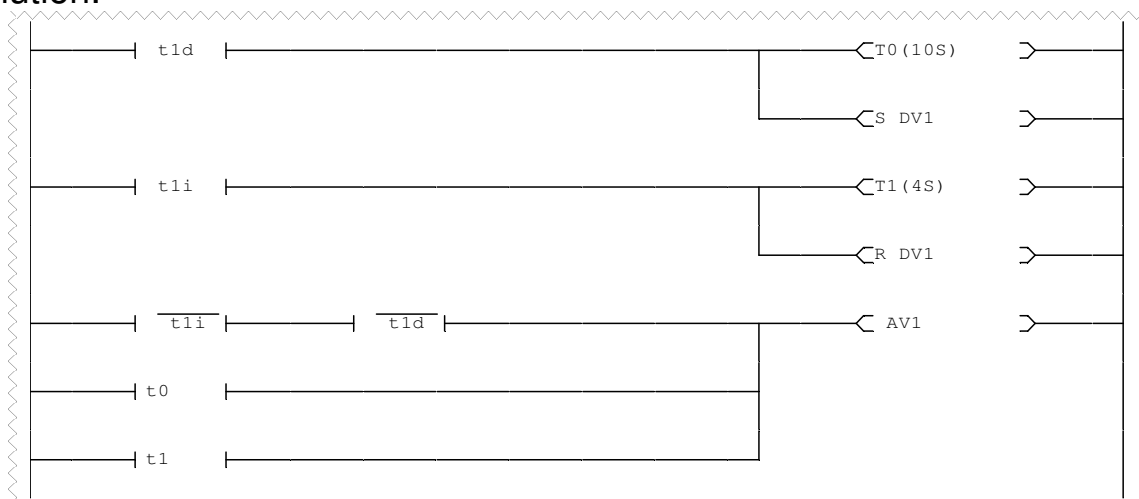
The second solution is identical from an operational point of view. It is used to display the use of a self-controlled variable.

Let's make our example more complex.

Conditions:

The locomotive must stop for 10 seconds to the right of track 1 and 4 seconds to the left.

Solution:



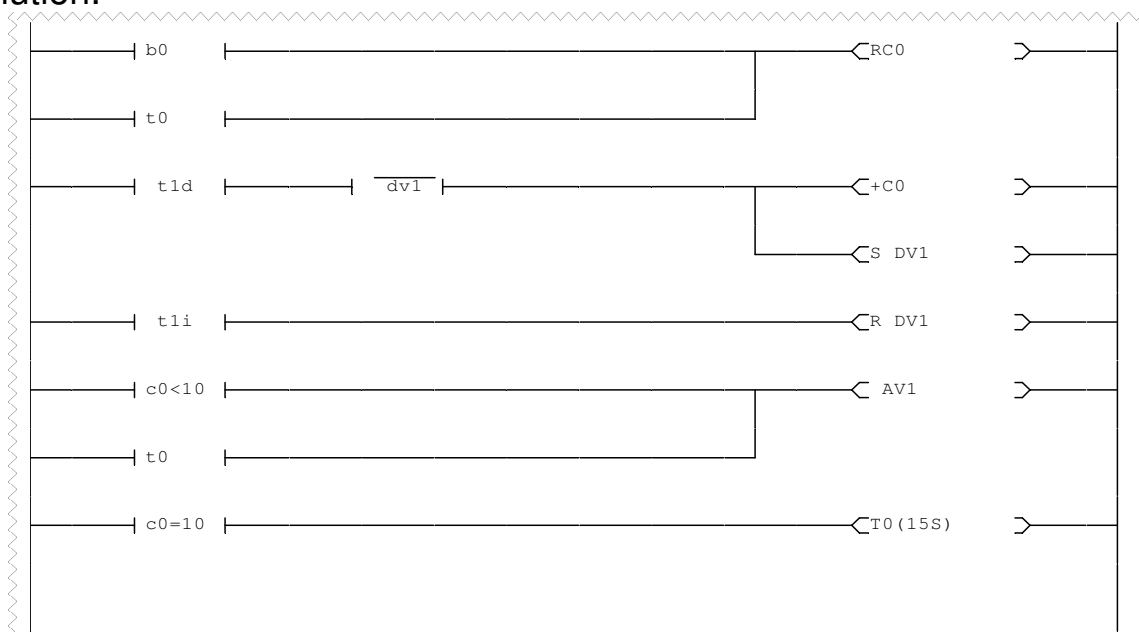
Example\ladder\ladder3.agn

A final example, even a little more complicated

Conditions:

Again a locomotive which makes round trips on track 1. For each 10 round trips it must stop for 15 seconds.


Solution:



Example\ladder\ladder4.agn

## Flow chart

AUTOSIM implements flow chart language in the following way:

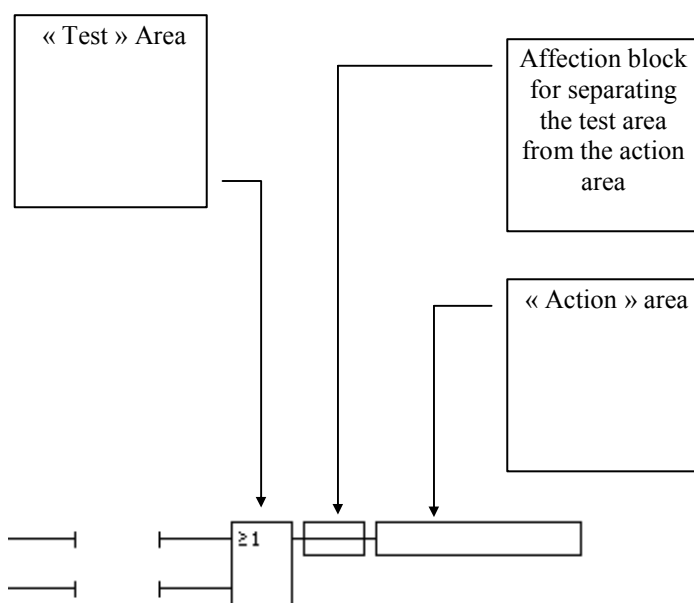
⇒ use of a special block called « assignment block », this block separates the action area and test area, it has the following form  and is associated with key [0] (zero),

⇒ it uses the functions « No », « And » and « Or »,

⇒ it uses action rectangles to the right of the action block.

Flow chart language is used for graphically writing boolean equations. The test content must comply with the syntax established in the « Common elements » chapter in this manual.

The content of action rectangles must comply with the syntax for actions, also described in the « Common elements » chapter of this manual.



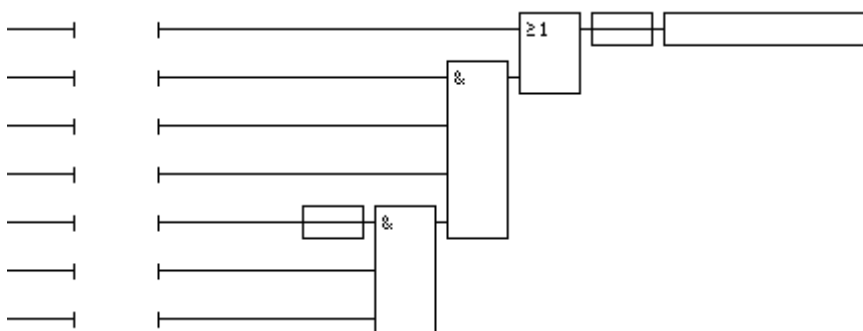
## Drawing flow charts

### Number of input of functions « And » and « Or »

The « And » and « Or » functions are respectively composed of a block  $\begin{array}{|c|} \hline \& \\ \hline \end{array}$  (key [2]) or a block  $\begin{array}{|c|} \hline \geq 1 \\ \hline \end{array}$  (key [3]), and possible blocks  $\begin{array}{|c|} \hline \\ \hline \end{array}$  (key [4]) for adding inputs to blocks and finally block  $\begin{array}{|c|} \hline \\ \hline \end{array}$  (key [5]). The functions « And » and « Or » thus involve a minimum of two inputs..

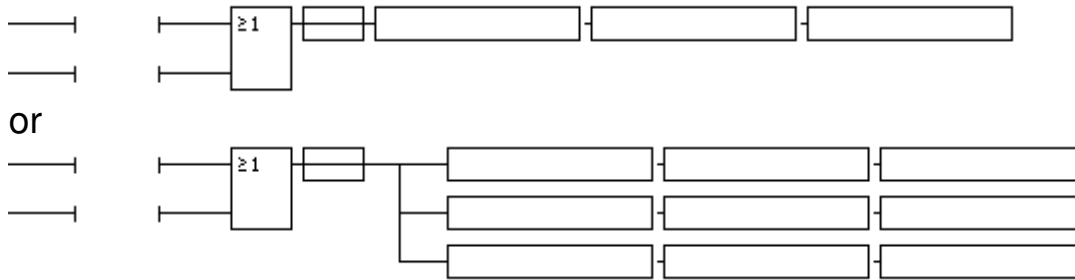
### Chaining the functions

The functions can be chained.



## Multiple actions

Multiple action rectangles can be associated to a flow chart after the assignment block..



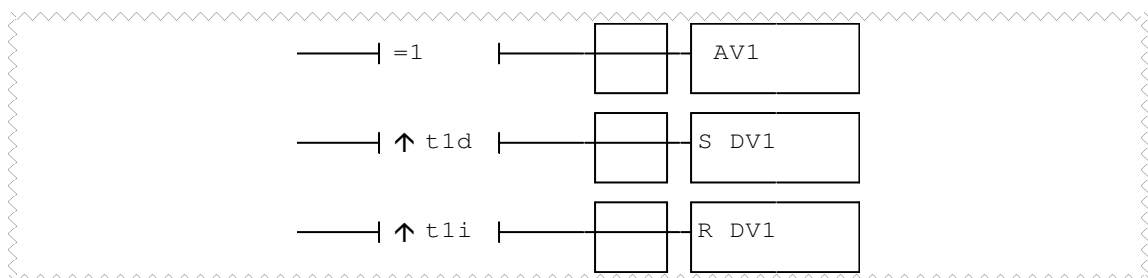
## Example of a flow chart

Let's start with the simplest example:

Conditions:

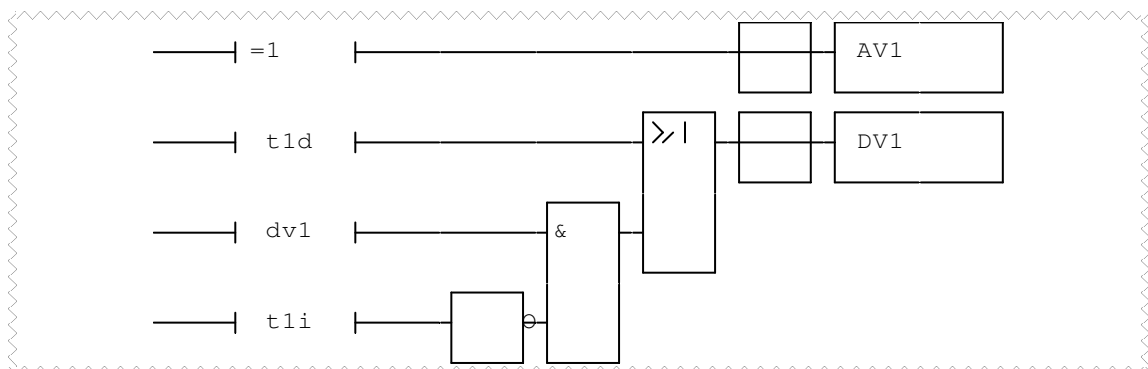
Roundtrip of a locomotive on track 1.

Solution 1:



 Example\flow chart\logigramme1.agn

Solution 2:



 Example\flow chart\logigramme2.agn

The second solution is identical from an operational point of view. It is used to display the use of a self-controlled variable.

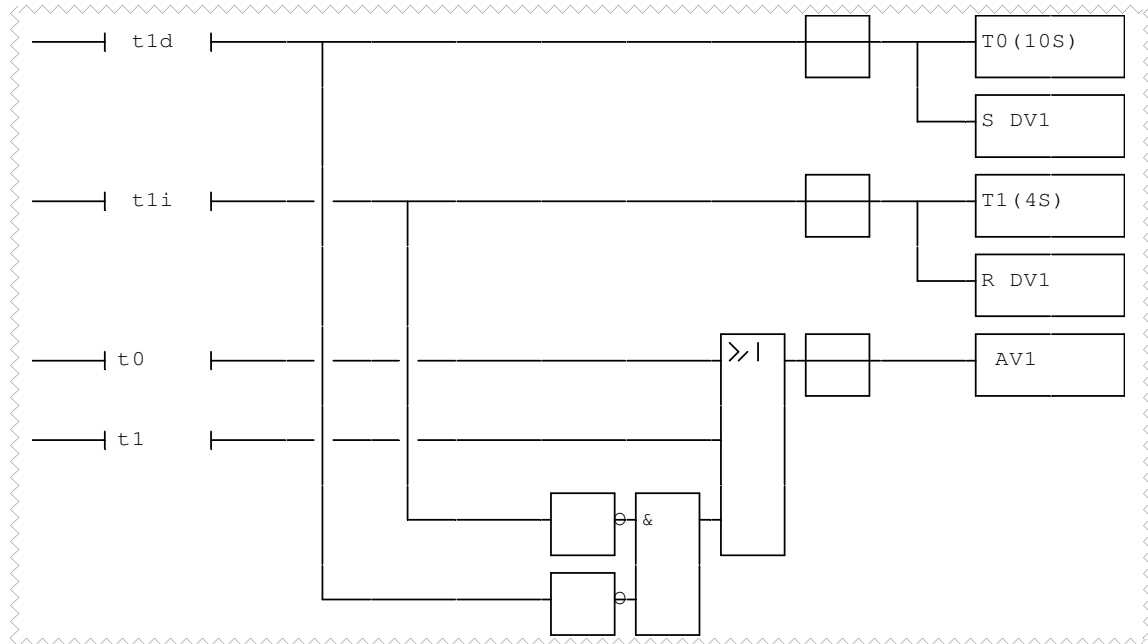
Let's make the example more complex.

Conditions:



The locomotive must stop for 10 seconds to the right of track 1 and 4 seconds to the left.

Solution:



Example\flow chart\logigramme3.agn

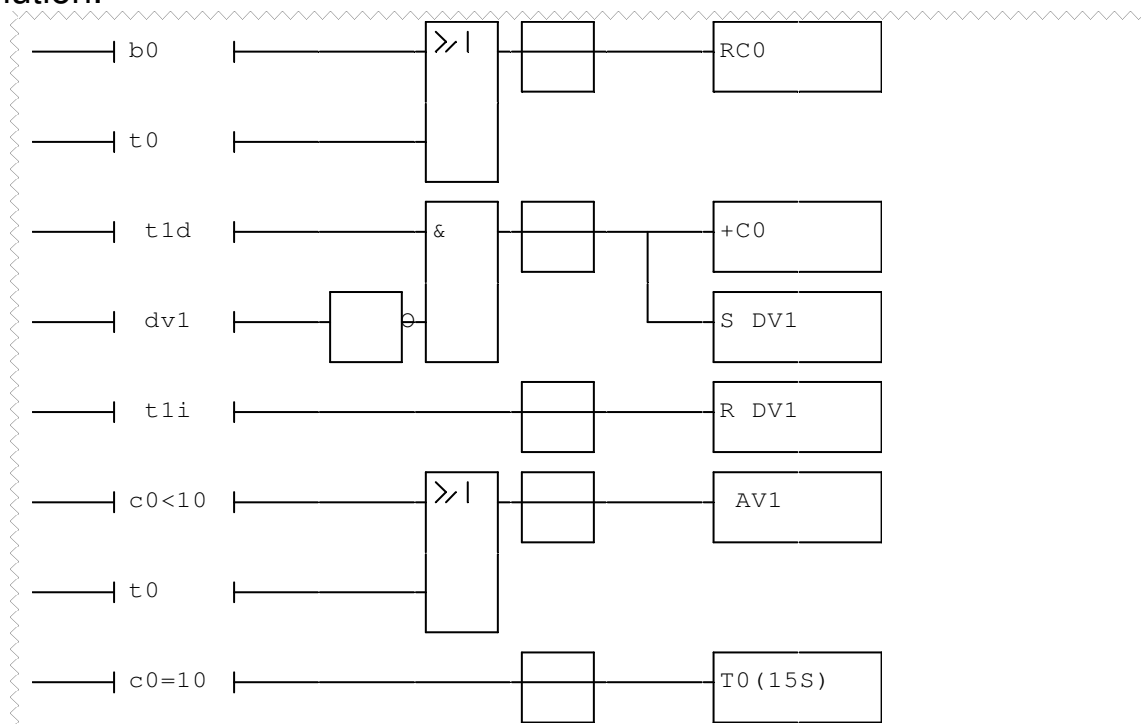
Note the reuse of the «And» block in the lower part of the example towards the inputs «`_t1d_`» and «`_t1i_`». This prevents having to write the two tests a second time.

A final example a bit more complicated.

Conditions:

Again a locomotive which makes round trips on track 1. Each 10 round trips it must stop for 15 seconds.

Solution:



Example\flow chart\logigramme4.agn

## Literal languages

This chapter describes the use of the three forms of literal language which are available in AUTOSIM:

- ⇒ low level literal language,
- ⇒ extended literal language,
- ⇒ IEC 1131-3 standard ST literal language

## How is a literal language used?

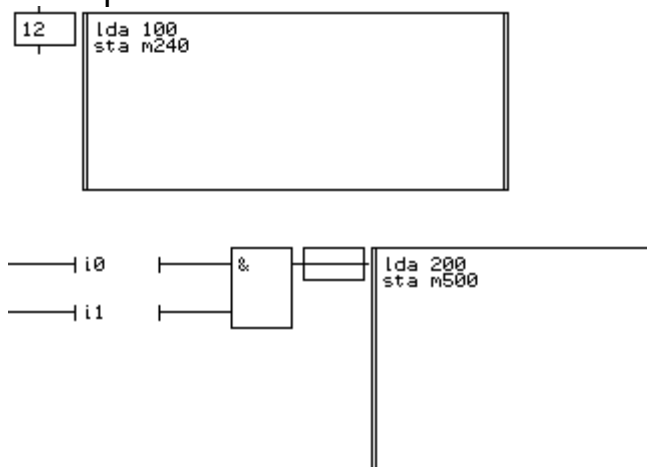
Literal language can be used in the following forms:

- ⇒ code files associated to an action (Grafcet, Ladder, flow chart),
- ⇒ code boxes associated to an action (Grafcet, flow chart),
- ⇒ literal code in action rectangle or coil (Grafcet, Ladder, flow chart),
- ⇒ code boxes used in the form of an organizational chart (see the «Organizational chart » chapter),
- ⇒ code files which support the function block functionality (see the « Function blocks » chapter),
- ⇒ code files which support a macro-instruction functionality see chapter Macro-instruction.

## Code box associated with a step or flow chart

A code box associated with an action is for being able to write lines of literal language on an application page.

Examples:



The code used above is scanned as long as the action is true. It is possible to use the action rectangles and code boxes together.

Example:

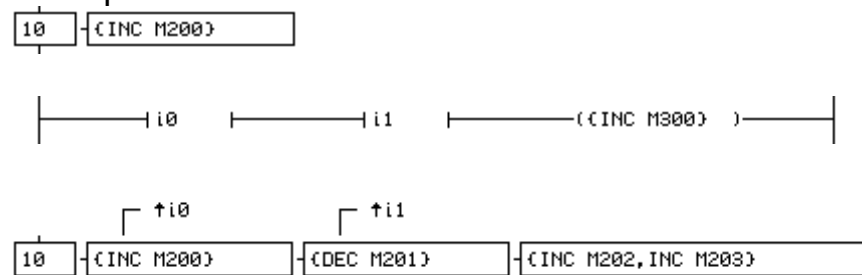


## Literal code in an action rectangle or coil

The characters « { » and « } » are used to directly enter instructions in literal language into an action rectangle (Grafcet and flow chart languages). The character « , » (comma) is used as a separator if multiple instructions are present in « { » and « } ».

This type of entry can be used with conditional orders.

Examples:



## Setting a code box

To create a code box, follow the steps below:

- ⇒ click on an empty space on the sheet with the right side of the mouse,
- ⇒ select « Add ... / Code box » from the menu,
- ⇒ click on the edge of the code box to edit its contents.

To exit the code box after editing click on [Enter] or click outside it.

## Low level literal language

This chapter describes the use of low level literal language. This language is an intermediate code between the evolved languages of Grafcet, flow chart, ladder, organizational chart, function block, extended literal, ST literal and executable languages. It is also known as pivot code. Post-processors translate low level literal language into executable code for the PC, automate or microprocessor card.

Literal language can also be used for an application in order to effect various boolean, numeric or algebraic operations.

Low level literal language is an assembler type language. It uses the idea of an accumulator for numeric treatment.

Extended literal language and ST literal language described in the following chapters, offer a simplified and higher level alternative for writing programs in literal language.

The general syntax for a line of low level literal language is:

«action » [[ [« Test »] « Test » ]...]

The actions and tests of low level literal language are represented by mnemonics formed of three letters. An instruction is always followed by an expression: variable, constant etc.

A line is composed of a single action and possibly a test. If a line only includes an action, then the instruction is always executed.

## Variables

The variables used are the same as those described in the « Common elements » chapter.

## Accumulators

Some instructions use an accumulator. The accumulators are internal registers which execute the final program and make it possible to temporarily store values.

There are three accumulators: a 16 bit accumulator known as AAA, a 32 bit accumulator known as AAL and a float accumulator known as AAF.

## Flags

Flags are boolean variables which are positioned based on the result of numeric operations.

There are four permanent flags to test the result of a calculation. These four indicators are:

- ⇒ carry indicator C: it indicates if an operations has generated a carry figure (1) or not (0),
- ⇒ zero indicator Z: it indicates if an operations has generated a nil result (1) or not nil (0),
- ⇒ sign indicator S: it indicates if an operation has generated a negative result (1) or positive one (0),
- ⇒ overflow indicator O: it indicates if an operation has generated an overflow (1).

## Addressing modes

Low level literal language has 5 addressing modes. An addressing mode is a characteristic associated to each literal language instruction. Addressing modes used appear below:

TYPE	SYNTAX	EXAMPLE
Immediate 16 bits	{constant}	100
Immediate 32 bits	{constant}L	100000L
Immediate float	{constant}R	3.14R
Absolute	{variable} {variable reference}	O540
16 bit accumulator	AAA	AAA
32 bit accumulator	AAL	AAL
Float accumulator	AAF	AAF
Indirect	{variable} {(word reference)}	O(220)
Label	:{label name}:	:loop

Thus an instruction has two characteristics: the type of variable and the addressing mode. Certain instructions support or do not support certain addressing modes and certain variable types. For example, an instruction may only apply to two words and not to other types of variables.

Note: Variables X and U can not be associated to an indirect address due to the non-linear nature of their assignments. If it is necessary to access a U variable table then a command #B must be used to make a table of linear bits.

## Tests

Tests that can be associated to instructions are composed of a mnemonic, a type of test and a variable.

Test mnemonics are used to set combination tests on multiple variables (and, or). If a test is composed of a single variable, an AND operator needs to be associated to it.

There are only three test mnemonics:

AND and

ORR or

EOR end or

Here are some examples of equivalencies in boolean equations and low level literal language:

```
o0=i1                : and i1
o0=i1.i2             : and i1 and i2
o0=i1+i2             : orr i1 eor i2
o0=i1+i2+i3+i4       : orr i1 orr i2 orr i3 eor i4
o0=(i1+i2).(i3+i4)    : orr i1 eor i2 orr i3 eor i4
o0=i1.(i2+i3+i4)      : and i1 orr i2 orr i3 eor i4
o0=(i1.i2)+(i3.i4)    ; impossible to translate directly,
                      ; intermediate variables
                      ; must be used:
```

```
equ u100 and i1 and i2
equ u101 and i3 and i4
equ o0 orr u100 eor u101
```

Test modifiers make it possible to test things other than the truth of a variable:

- ⇒ / no
- ⇒ # rising edge
- ⇒ \* falling edge
- ⇒ @ immediate state

## Notes:

- ⇒ boolean variables are updated after each execution cycle. In other words, if a binary variable is positioned at a state during a cycle, then its new state will be detected during the following cycle. The text modifier @ makes it possible to obtain the real state of a boolean variable without waiting for the following cycle.

- ⇒ test modifiers cannot be used with numeric tests.

### Examples:

```
set o100
equ o0 and @o100          ; true test of the first cycle
equ o1 and o100           ; true test at the second cycle
```

Only two addressing modes are available for tests: absolute and indirect

A test for counters, words, longs and floating points is available:

### Syntax:

```
« {variable} {=, !, <, >, <<, >>} {constant or variable} »
```

= equal,  
! different,  
< less than not signed,  
> greater than not signed,  
<< less than signed,  
>> greater than signed,

By default, constants are written in decimals. The suffixes « \$ » and « % » are used for writing in hexadecimal or binary. The quotation marks are for writing in ASCII.

32 bit constants must be followed by the letter « L ».

Real constants must be followed by the letter « R ».

A word or a counter can be compared to a word, a counter of a 16 bit constant..

A long can be compared to a long or a 32 bit constant.

A float can be compared to a float or a real constant.

### Examples:

```
and c0>100 and m225=10
orr m200=m201 eor m202=m203 and f100=f101 and f200<f203
orr m200<<-100 eor m200>>200
and f200=3.14r
and l200=$12345678L
and m200=%1111111100000000
```

### Comments

Comments need to start with the character « ; » (semi-colon), all the characters after it are ignored.

## Numbering base

The values (variable references or constants) can be written in decimal, hexadecimal, binary or ASCII.

The following syntax must be applied for 16 bit constants:

- ⇒ decimal: possibly the character « - » plus 1 to 5 digits « 0123456789 »,
- ⇒ hexadecimal: the prefix « \$ » or « 16# » followed by 1 to 4 digits « 0123456789ABCDEF »,
- ⇒ binary: the prefix « % » or « 2# » followed by 1 to 16 digits « 01 »,
- ⇒ ASCII: the character « " » followed by 1 or 2 characters followed by « " ».

The following syntax must be applied for 32 bit constants:

- ⇒ Decimal: possibly the character « - » plus 1 to 10 digits « 0123456789 »,
- ⇒ Hexadecimal: the prefix « \$ » or « 16# » followed by 1 to 8 digits « 0123456789ABCDEF »,
- ⇒ Binary: the prefix « % » or « 2# » followed by 1 to 32 digits « 01 »,
- ⇒ ASCII: the character « " » followed by 1 to 4 characters followed by « " ».

The following syntax must be applied for real constants:

`[ - ] i [ [ . d ] E s x ]`

i is the whole part

of a decimal part

s possible sign of an exponent

x possible exponent

## Presettings

A presetting is used to fix the value of a variable before starting the application.

The variables T or %T, M or %MW, L or %MD and F or %F can be preset.

The syntax is as follows:

`« $(variable)=constant{,constant{,constant...}} »`

For time delays the variable must be written in decimal and be included between 0 and 65535.

For words the following syntax must be used:



- ⇒ Decimal: possibly the character « - » plus 1 to 5 digits « 0123456789 »,
- ⇒ Hexadecimal: the prefix « \$ » or « 16# » followed by 1 to 4 digits « 0123456789ABCDEF »,
- ⇒ Binary: the prefix « % » or « 2# » followed by 1 to 16 digits « 01 »,
- ⇒ ASCII: (two characters per word) the character « " » followed by n characters followed by « " »,
- ⇒ ASCII: (one character per word) the character « ' » followed by n characters followed by « ' ».

For longs the following syntax must be used:

- ⇒ Decimal: possible the character « - » plus 1 to 10 digits « 0123456789 »,
- ⇒ Hexadecimal: the prefix « \$ » or « 16# » followed by 1 to 8 digits « 0123456789ABCDEF »,
- ⇒ Binary: the character « % » or « 2# » followed by 1 to 32 digits « 01 »,
- ⇒ ASCII: (four characters per long) the character « " » followed by n characters followed by « " »,
- ⇒ ASCII: (one character per long) the character « ' » followed by n characters followed by « ' »

For floats the value must be written in the following form:

`[-] i [[.d] Esx]`

i is the whole part

d a possible decimal part

s a possible exponent sign

x a possible exponent

Examples:

`$t25=100`

fixes the time delay order 25 at 10 s

`$MW200=100,200,300,400`

places the values 100,200,300,400 in the words 200, 201, 202, 203

`$m200="ABCDEF"`

places the string « ABCDEF » starting from m200 (« AB » in m200, « CD » in m201, « EF » in m202)

`$m200='ABCDEF'`

places the string « ABCDEF » starting from m200, each word receives a character

```
$f1000=3.14
```

places the value 3,14 in f1000

```
$%mf100=5.1E-15
```

places the value 5,1 \* 10 exponent -15 in %mf100

```
$l200=16#12345678
```

places the value 12345678 (hexa) in the long l200

It is easier to write text in the presettings.

Example:

```
$m200=" Stop the gate N°10 "
```

Places the message starting from word 200 by placing two characters in each word.

```
$m400=' Motor fault '
```

Places the message starting from word 400 by placing a character in the byte of lower weights of each word, the byte of higher weights contains 0.

The syntax « \$...= » is used to continue a table of presettings after the previous one.

For example:

```
#$m200=1,2,3,4,5
```

```
#$...=6,7,8,9
```

Place the variables 1 to 9 in the words m200 à m208.

Presettings can be written in the same manner as low level literal language or in a command on a sheet. In this case, the presetting starts with the character « # ».

Example of a presetting written in a code box:



Example of a presetting written in a command:

```
##m200=12,13
```

## Indirect addressing

Indirect addressing is used to effect an operation on a variable with an index..

These are M variables (words) which are used as an index

Syntax:

« variable ( index ) »

Example:

```
lda 10          ; load 10 in the accumulator
sta m200        ; enter in the word 200
set o(200)      ; set to one the output indicated by the word 200 (o10)
```

## Address of a variable

The character « ? » is used to specify the address of a variable.

Example:

```
lda ?o10        ; enters the value 10 in the accumulator
```

This syntax is primarily of interest if symbols are used.

Example:

```
lda ?_gate_     ; enters the variable number in the accumulator
                ; associated to symbol « _gate_ »
```

This syntax can also be used in presettings to create variable address tables..

Example:

```
$m200=?_gate1_, ?_gate2_, ?_gate3_
```

## Jumps and labels

Jumps must be referred to a label. Label syntax is:

«:label name: »

Example:

```
jmp:next:
...
:next:
```

## Function list by type

### Boolean functions

SET	set to one
RES	reset
INV	inversion
EQU	equivalence
NEQ	non-equivalence

**Loading and storage functions on integers and floats**

LDA	load
STA	storage

**Arithmetic functions on integers and floats**

ADA	addition
SBA	subtraction
MLA	multiplication
DVA	division
CPA	comparison

**Arithmetic functions on floats**

ABS	absolute value
SQR	square root

**Access functions for PC input/output ports**

AIN	access input
AOU	access output

**Access functions for PC memory**

ATM	input address memory
MTA	output address memory

**Binary functions on integers**

ANA	and bit to bit
ORA	or bit to bit
XRA	exclusive or bit to bit
TSA	test bit to bit
SET	set all bits to one
RES	reset all bits
RRA	shift to the right
RLA	shift to the left

**Other functions on integers**

INC	incrementation
DEC	decrementation

**Conversion functions**

ATB	integers to booleans
BTA	booleans to integers
FTI	float to integer
ITF	integer to float
LTI	32 bit integer to 16 bit integer

ITL            16 bit integer to 32 bit integer

### Trigonometric functions

SIN	sine
COS	cosine
TAN	tangent
ASI	arc sine
ACO	arc cosine
ATA	arc tangent

### Connection functions

JMP	jump
JSR	jump to sub routine
RET	return from sub routine

### Test functions

RFZ	zero result flag
RFS	sign flag
RFO	overflow flag
RFC	carry flag

### Asynchronous access functions to inputs outputs

RIN	read inputs
WOU	write outputs

### Information contained in the function list

The following are provided for each instruction:

- ⇒ Name: mnemonic.
- ⇒ Function: a description of the function created by the instruction.
- ⇒ Variables: the types of variables used with the instruction
- ⇒ Addressing: the types of addressing used
- ⇒ Also see: the other instructions related to the mnemonic.
- ⇒ Example: a example of the use.

The post-processors which generate construction language are subject to certain limitations. See the information on these post-processors for details on these limitations.

# ***ABS***

Name : ABS - abs accumulator  
Function : calculate the absolute value of the floating accumulator  
Variables : none  
Addressing : accumulator  
Also see : SQR  
Example :

```
lda f200  
abs aaf  
sta f201  
; leaves f201 in the absolute value of f200
```

# ACO

Name : ACO – accumulator arc cosine  
Function : calculate the arc cosine value of the floating-point accumulator  
Variables : none  
Addressing : accumulator  
Also see : COS, SIN, TAN, ASI, ATA  
Example: :

```
lda f200  
aco aaf  
sta f201  
; leave the value of the arc cosine of f200 in f201
```

# ***ADA***

Name	:	ADA - adds accumulator
Function	:	adds a value to the accumulator
Variables	:	M or %MW, L or %MD, F or %MF
Addressing	:	absolute, indirect, immediate
Also see	:	<u>SBA</u>
Example	:	
		ada 200
		; adds 200 to the 16 bit accumulator
		ada f124
		; adds the content of f124 to the float accumulator
		ada l200
		; adds the content of l200 to the 32 bit accumulator
		ada 200L
		; adds 200 to the 32 bit accumulator
		ada 3.14R
		; adds 3.14 to the float accumulator



# ***AIN***

Name	:	AIN - accumulator input
Function	:	reads an input port (8 bits) and stores in the lower part of the 16 bit accumulator ; reads a 16 bit input port and stores in the 16 bit accumulator (in this case the port address must be written in the form of a 32 bit constant) only useable with PC compiler
Variables	:	M or %MW
Addressing	:	indirect, immediate
Also see	:	<u>AOU</u>
Example	:	 ain \$3f8 ; reads port \$3f8 (8 bits)  ain \$3f8l ; reads port \$3f8 (16 bits)

# ANA

Name	:	ANA - and accumulator
Function	:	effects an AND logic in the 16 bit accumulator and a word or a constant or the 32 bit accumulator and a long or a constant
Variables	:	M or %MW, L or %MD
Addressing	:	absolute, indirect, immediate
Also see	:	<u>ORA, XRA</u>
Example	:	 ana %1111111100000000 ; masks the 8 bits of lower weight of ; the 16 bit accumulator  ana \$ffff0000L ; masks the 16 bits of lower weight of the 32 bit accumulator

# ***AOU***

Name : AOU - accumulator output

Function : transfers the lower part (8 bits) of the 16 bit accumulator on an output port ;  
transfers the 16 bits of the 16 bit accumulator on an output port (in this case the port address must be written in the form of a 32 bit constant)  
only useable with PC compiler

Variables : M or %MW

Addressing : indirect, immediate

Also see : AIN

Example :

```
lda "A"
aou $3f8
; places the character« A » on output port $3f8
```

```
lda $3f8
sta m200
lda "z"
aou m(200)
; places character « z » on output port $3f8
```

```
lda $1234
aou $3001
; places the 16 bit value 1234 on output port $300
```

# *ASI*

Name : ASI – accumulator arc sine  
Function : calculate the arc sine value of the floating-point accumulator  
Variables : none  
Addressing : accumulator  
Also see : COS, SIN, TAN, ACO, ATA  
Example: :

```
lda f200  
asi aaf  
sta f201  
; leave the value of the arc sine of f200 in f201
```

# ***ATA***

Name : ATA – accumulator arc tangent  
Function : calculate the arc tangent value of the floating-point accumulator  
Variables : none  
Addressing : accumulator  
Also see : COS, SIN, TAN, ACO, ASI  
Example: :

```
lda f200  
ata aaf  
sta f201  
; leave the value of the arc tangent of f200 in f201
```

# ***ATB***

Name	:	ATB - accumulator to bit
Function	:	transfers the 16 bits of the 16 bit accumulator towards the subsequent 16 boolean variables ; the the lower weight bit correspond to the first boolean variable
Variables	:	I or %I, O or %Q, B or %M, T or %T, U*
Addressing	:	absolute
Also see	:	<u>BTA</u>
Example	:	<pre>lda m200 atb o0 ; recopies the 16 bits of m200 in variables ; o0 to o15</pre>

---

\* Note: to be able to use the U bits with this function it is necessary to create a linear table of bits using command #B.

# *ATM*

Name	:	ATM - accumulator to memory
Function	:	transfers the 16 bit accumulator to a memory address; the word or specified constant defines the memory address offset to reach, the word m0 must be loaded with the segment value of the memory address to reach only useable with PC compiler
Variables	:	M or %MW
Addressing	:	indirect, immediate
Also see	:	<u>MTA</u>
Example	:	<pre>lda \$b800 sta m0 lda 64258 atm \$10 ; places the value 64258 at address \$b800:\$0010</pre>

# ***BTA***

Name	:	BTA - bit to accumulator
Function	:	transfers the subsequent 16 boolean variables towards the 16 bits of the 16 bit accumulator ; the lower weight bit corresponds to the first boolean variable
Variables	:	I or %I, O or %Q, B or %M, T or %T, U*
Addressing	:	absolute
Also see	:	<u>ATB</u>
Example	:	 bta i0 sta m200 ; recopies the 16 inputs i0 to i15 in the word m200

---

\* Note: to be able to use the U bits with this function it is necessary to create a linear table of bits using command #B.



# ***COS***

Name : COS – accumulator cosine  
Function : calculate the cosine value of the floating-point accumulator  
Variables : none  
Addressing : accumulator  
Also see : SIN, TAN, ACO, ASI, ATA  
Example: :

```
lda f200
```

```
cos aaf
```

```
sta f201
```

```
; leave the value of the cosine of f200 in f201
```

# *CPA*

Name	:	CPA - compares accumulator
Function	:	compares a value at the 16 bit or 32 bit or floating accumulator, effects the same operation as SBA but without changing the content of the accumulator
Variables	:	M or %MW, L or %MD, F or %MF
Addressing	:	absolute, indirect, immediate
Also see	:	<u>SBA</u>
Example	:	

```
lda m200
```

```
cpa 4
```

```
rfz o0
```

```
; sets o0 to 1 if m200 is equal to 4, otherwise o0
```

```
; is reset to 0
```

```
lda f200
```

```
cpa f201
```

```
rfz o1
```

```
; sets o1 to 1 if f200 is equal to f201, otherwise o1
```

```
; is reset to 0
```

# ***DEC***

Name	:	DEC – decrement
Function	:	decrements a word, a counter, a long, the 16 bit or 32 bit accumulator
Variables	:	M or %MW, C or %C, L or %MD
Addressing	:	absolute, indirect, accumulator
Also see	:	<u>INC</u>
Example	:	

dec m200  
; decrements m200

dec aal  
; decrements the 32 bit accumulator

dec m200  
dec m201 and m200=-1  
; decrements a 32 bit value composed of  
; m200 (lower weights)  
; et m201 (higher weights)

# *DVA*

Name	:	DVA - divides accumulator
Function	:	division of the 16 bit accumulator by a word or a constant; division of the float accumulator by a float or a constant; division of the 32 bit by a long or a constant, for the 16 bit accumulator the remainder is placed in word m0, if the division is by 0 system bit 56 passes to 1
Variables	:	M or %MW, L or %MD, F or %MF
Addressing	:	absolute, indirect, immediate
Also see	:	<u>MLA</u>
Example	:	<pre>lda m200 dva 10 sta m201 ; m201 is equal to m200 divided by 10, m0 contains the ; remainder  lda l200 dva \$10000L sta l201</pre>

# ***EQU***

Name	:	EQU - equal
Function	:	sets a variable to 1 if the test is true, if not the variable is set to 0
Variables	:	I or %I, O or %Q, B or %M, T or %T, X or %X, U
Addressing	:	absolute, indirect (except for X variables)
Also see	:	<u>NEQ</u> , <u>SET</u> , <u>RES</u> , <u>INV</u>
Example	:	

```
equ o0 and i10  
; sets the output of o0 to the same state as input i10
```

```
lda 10  
sta m200  
equ o(200) and i0  
; sets o10 to the same state as input i0
```

```
$t0=100  
equ t0 and i0  
equ o0 and t0  
; sets o0 to the state of i0 with an activation delay  
; of 10 seconds
```

# ***FTI***

Name : FTI - float to integer

Function : transfers the float accumulator to the 16 bit accumulator

Variables : none

Addressing : accumulator

Also see : ITF

Example :

```
lda f200
fti aaa
sta m1000
; leaves the integer part of f200 in m1000
```

# ***INC***

Name	:	INC - increment
Function	:	increments a word, a counter, a long the 16 or 32 bit accumulator
Variables	:	M or %MW, C or %C, L or %MD
Addressing	:	absolute, indirect, accumulator
Also see	:	<u>DEC</u>
Example	:	

```
inc m200  
; adds 1 to m200
```

```
inc m200  
inc m201 and m201=0  
; increments a value on 32 bits, m200  
; represents the  
; lower weights, and m201 the higher weights
```

```
inc l200  
; increments long l200
```

# *INV*

Name	:	INV - inverse
Function	:	inverts the state of a boolean variable or inverts all the bits of a word, a long or the 16 bit or 32 bit accumulator
Variables	:	I or %I, O or %Q, B or %M, T or %T, X or %X, U, M or %MW, L or %MD
Addressing	:	absolute, indirect, accumulator
Also see	:	<u>EQU</u> , <u>NEQ</u> , <u>SET</u> , <u>RES</u>
Example	:	 inv o0 ; inverts the state of output 0  inv aaa ; inverts all the bits of the 16 bit accumulator  inv m200 and i0 ; inverts all m200 bits if i0 is at state 1



# ***ITF***

Name : ITF - integer to float  
Function : transfers the 16 bit accumulator to the float accumulator  
Variables : none  
Addressing : accumulator  
Also see : FTI  
Example :

```
lda 1000  
itf aaa  
sta f200  
; leaves the constant 1000 in f200
```

# ***ITL***

Name	:	ITL - integer to long
Function	:	transfers the 16 bit accumulator to the 32 bit accumulator
Variables	:	none
Addressing	:	accumulator
Also see	:	<u>LTI</u>
Example	:	<pre>lda 1000 itl aaa sta f200 ; leaves the constant 1000 in l200</pre>

# ***JMP***

Name	:	JMP - jump
Function	:	jump to a label
Variables	:	label
Addressing	:	label
Also see	:	<u>JSR</u>
Example	:	

jmp:end of program:

; unconditional connection to end of  
; program label:

jmp:string: and i0

set o0

set o1

:string:

; conditional connection to a label:string:  
; following the state of i0

# ***JSR***

Name : JSR - jump sub routine  
Function : effects a connection to a sub routine  
Variables : label  
Addressing : label  
Also see : RET  
Example :

```
lda m200  
jsr:square:  
sta m201  
jmp end:
```

```
:square:  
sta m53  
mla m53  
sta m53  
ret m53
```

```
:end:
```

; the sub routine « square » raises the content  
; of the accumulator to the square

# ***LDA***

Name	:	LDA - load accumulator
Function	:	loads a constant, word or counter in the 16 bit accumulator; loads a long or constant in the 32 bit accumulator; loads a float or a constant in the float accumulator; loads a counter or a time delay in the 16 bit accumulator
Variables	:	M or %MW, C or %C, L or %MD, F or %MF, T or %T
Addressing	:	absolute, indirect, immediate
Also see	:	<u>STA</u>
Example	:	

lda 200

; loads the constant 200 in the 16 bit accumulator

lda 0.01R

; loads the real constant 0.01 in the float accumulator

lda t10

; loads the counter of time delay 10 in the  
; accumulator

# ***LTI***

Name	:	LTI - long to integer
Function	:	transfers the 32 bit accumulator to the 16 bit accumulator
Variables	:	none
Addressing	:	accumulator
Also see	:	<u>ITL</u>
Example	:	<pre>lda l200 lti aaa sta m1000 ; leaves the 16 bits of lower weight of l200 in m1000</pre>

# ***MLA***

Name	:	MLA - multiples accumulator
Function	:	multiplies the 16 bit accumulator by a word or a constant; multiplies the 32 bit accumulator by a long or a constant; multiplies the float accumulator by a float or a constant; for the 16 bit accumulator the 16 bits of higher weight result of the multiplication will be transferred in m0
Variables	:	M or %MW, L or %MD, F or %MF
Addressing	:	absolute, indirect, immediate
Also see	:	<u>DVA</u>
Example	:	lda m200 mla 10 sta m201 ; multiplies m200 by 10, m201 is loaded with the ; 16 bits of lower weight, and m0 with the 16 bits of ; higher weight

# ***MTA***

Name	:	MTA - memory to accumulator
Function	:	transfers the contents of a memory address to the 16 bit accumulator, the specified word or constant defines the offset of the memory address to reach; the word m0 must be loaded with the segment value of the memory address to be reached; only useable with a PC compiler
Variables	:	M or %MW
Addressing	:	indirect, immediate
Also see	:	<u>ATM</u>
Example	:	<pre>lda \$b800 sta m0 mta \$10 ; places the value contained at address \$b800:\$0010 ; in the 16 bit accumulator</pre>



# NEQ

Name	:	NEQ - not equal
Function	:	sets a variable to 0 if the test is true, if not the variable is set to 1
Variables	:	I or %I, O or %Q, B or %M, T or %T, X or %X, U
Addressing	:	absolute, indirect (except for X variables)
Also see	:	<u>EQU</u> , <u>SET</u> , <u>RES</u> , <u>INV</u>
Example	:	<pre>neq o0 and i00 ; sets the output of o0 to a complement state of input ; i0  lda 10 sta m200 neq o(200) and i0 ; sets o10 to a complement state of input i0  \$t0=100 neq t0 and i0 neq o0 and t0 ; sets o0 to the state of i0 with a deactivation ; delay of 10 seconds</pre>

# ***ORA***

Name	:	ORA - or accumulator
Function	:	effects an OR logic on the 16 bit accumulator and a word or a constant, or on the 32 bit accumulator and a long or a constant
Variables	:	M or %M, L or %MD
Addressing	:	absolute, indirect, immediate
Also see	:	<u>ANA</u> , <u>XRA</u>
Example	:	 ora %1111111100000000 ; sets the 8 bits of lower weight of ; the 16 bit accumulator to 1  ora \$ffff0000L ; sets the 16 bits of higher weight of the 32 bit accumulator ; to 1

# ***RES***

Name	:	RES - reset
Function	:	sets a boolean variable, a word a counter, a long, the 16 bit accumulator or the 32 bit accumulator to 0
Variables	:	I or %I, O or %Q, B or %M, T or %T, X or %X, U, M or %MW, C or %C, L or %MD
Addressing	:	absolute, indirect (except for X variables), accumulator
Also see	:	<u>NEQ</u> , <u>SET</u> , <u>EQU</u> , <u>INV</u>
Example	:	<pre>res o0 ; sets the output of o0 to 0  lda 10 sta m200 res o(200) and i0 ; sets o10 to 0 if input i0 is at 1  res c0 ; sets counter 0 to 0</pre>

# ***RET***

Name	:	RET - return
Function	:	indicates the return of a sub routine and places a word or a constant in the 16 bit accumulator; or places a long or a constant in the 32 bit accumulator; or places a float or a constant in the float accumulator
Variables	:	M or %MW, L or %MD, F or %MF
Addressing	:	absolute, indirect, immediate
Also see	:	<u>JSR</u>
Example	:	 ret 0 ; returns to a sub routine by placing 0 in ; the 16 bit accumulator  ret f200 ; returns to a sub routine by placing the content of ; f200 in the float accumulator

# ***RFC***

Name	:	RFC - read flag: carry
Function	:	transfers the carry indicator in a boolean variable
Variables	:	I or %I, O or %Q, B or %M, T or %T, X or %X, U
Addressing	:	absolute
Also see	:	<u>RFZ</u> , <u>RFS</u> , <u>RFO</u>
Example	:	

rfc o0

; transfers the carry indicator to o0

lda m200

ada m300

sta m400

rfc b99

lda m201

ada m301

sta m401

inc m401 and b99

; effects an addition on 32 bits

; (m400,401)=(m200,201)+(m300,301)

; m200, m300 and m400 are lower weights

; m201, m301 and m401 are higher weights

# ***RFO***

Name	:	RFO - read flag: overflow
Function	:	transfers the contents of the overflow indicator in a boolean variable
Variables	:	I or %I, O or %Q, B or %M, T or %T, X or %X, U
Addressing	:	absolute
Also see	:	<u>RFZ</u> , <u>RFS</u> , <u>RFC</u>
Example	:	 rfo o0 ; transfers the overflow indicator to o0

# ***RFS***

Name	:	RFS - read flag: sign
Function	:	transfers the sign indicator in a boolean variable
Variables	:	I or %I, O or %Q, B or %M, T or %T, X or %X, U
Addressing	:	absolute
Also see	:	<u>RFZ</u> , <u>RFC</u> , <u>RFO</u>
Example	:	rfs o0 ; transfers the sign indicator to o0

# ***RFZ***

Name	:	RFZ - read flag: zero
Function	:	transfers the content of a zero result indicator in a boolean variable
Variables	:	I or %I, O or %Q, B or %M, T or %T, X or %X, U
Addressing	:	absolute
Also see	:	<u>RFC</u> , <u>RFS</u> , <u>RFO</u>
Example	:	

rfz o0

; transfers the zero result indicator to o0

lda m200

cpa m201

rfz o0

; position o0 at 1 if m200 is equal to m201

; or 0 if not



# ***RIN***

Name	:	RIN - read input
Function	:	effects a reading of physical input. This function is only implemented on Z targets and varies following the target. See the documentation related to each executor for more information..
Variables	:	none
Addressing	:	immediate
Also see	:	<u>WOU</u>

# ***RLA***

Name	:	RLA - rotate left accumulator
Function	:	effects a left rotation of the bits of the 16 bit or 32 bit accumulator; the bits evacuated to the left enter on the right, the subject of this function is a constant which sets the number of shifts to be made, the size of the subject (16 or 32 bits) determines which of the accumulators will undergo rotation
Variables	:	none
Addressing	:	immediate
Also see	:	<u>RRA</u>
Example	:	<pre>ana \$f000 ; separates the digit of higher weight of the 16 bit accumulator  rla 4 ; and brings it to the right  rla 8L ; effects 8 rotations to the left of the bits of the 32 bit ; accumulator</pre>

# ***RRA***

Name : RRA - rotate right accumulator  
Function : effects a right rotation of the bits of the

16 bit or 32 bit accumulator; the bits evacuated to the right enter on the left, the subject of this function is a constant which sets the number of shifts to be made, the size of the subject (16 or 32 bits) determines which of the accumulators will undergo rotation

Variables : none  
Addressing : immediate  
Also see : RLA  
Example :

ana \$f000

; separates the digit of higher weight of the 16 bit

rra 12

; and brings it to the right

rra 1L

; effects a rotation of the bits of the 32 bit accumulator

; to a position towards the right

# ***SBA***

Name	:	SBA - subtracts accumulator
Function	:	removes the content of a word or constant from the 16 bit accumulator; removes the content of a long or a constant from the 32 bit accumulator; removes the content of a float or constant from the float accumulator
Variables	:	M or %MW, L or %MD, F or %MF
Addressing	:	absolute, indirect, immediate
Also see	:	<u>ADA</u>
Example	:	<p>sba 200 ; removes 200 from the 16 bit accumulator</p> <p>sba f(421) ; removes the float content if the number is contained ; in word 421 from the float accumulator</p>

# ***SET***

Name	:	SET - set
Function	:	sets a boolean variable to 1; sets all the bits of a word, a counter, a long, the 16 bit or the 32 bit accumulator to 1
Variables	:	I or %I, O or %Q, B or %M, T or %T, X or %X, U, M or %MW, C or %C, L or %MD
Addressing	:	absolute, indirect (except for X variables), accumulator
Also see	:	<u>NEQ</u> , <u>RES</u> , <u>EQU</u> , <u>INV</u>
Example	:	

```
set o0
; sets the output of o0 to 1
```

```
lda 10
sta m200
set o(200) and i0
```

; sets o10 to 1 if input i0 is at 1

```
set m200
; sets m200 to the value -1
```

```
set aal
; sets all the bits of the 32 bit accumulator to 1
```

# ***SIN***

Name : SIN – accumulator sine  
Function : calculate the sine value of the floating-point accumulator  
Variables : none  
Addressing : accumulator  
Also see : COS, TAN, ACO, ASI, ATA  
Example: :

```
lda f200  
sin aaf  
sta f201  
; leave the value of the sine of f200 in f201
```

# ***SQR***

Name : SQR - square root  
Function : calculates the square root of the float accumulator  
Variables : none  
Addressing : accumulator  
Also see : ABS  
Example :

```
lda 9  
itf aaa  
sqr aaf  
fti aaa  
sta m200  
; leaves value 3 in m200
```

# STA

Name	:	STA - store accumulator
Function	:	stores the 16 bit accumulator in a counter or a word; stores the 32 bit accumulator in a long; stores the float accumulator in a float, stores the 16 bit accumulator in a time delay order
Variables	:	M or %MW, C or %C, L or %MD, F or %MF, T or %T
Addressing	:	absolute, indirect
Also see	:	<u>LDA</u>
Example	:	 sta m200 ; transfers the content of the 16 bit accumulator ; to word 200  sta f200 ; transfers the content of the float accumulator ; to float 200  sta l200 ; transfers the 32 bit accumulator to long l200



# ***TAN***

Name : TAN – accumulator tangent  
Function : calculate the tangent value of the floating-point accumulator  
Variables : none  
Addressing : accumulator  
Also see : COS, SIN, ACO, ASI, ATA  
Example: :

```
lda f200  
tan aaf  
sta f201  
; leave the value of the tangent of f200 in f201
```

# *TSA*

Name	:	TSA - test accumulator
Function	:	effects AND logic on the 16 bit accumulator and a word or a constant, effects AND logic on the 32 bit accumulator and a long or a constant, operates in a similar manner to ANA instruction but without changing the accumulator content
Variables	:	M or %MW, L or %MD
Addressing	:	absolute, indirect, immediate
Also see	:	<u>ANA</u>
Example	:	<pre>tsa %10 rfz b99 jmp:follow: and b99 ; connection to label:follow: if bit 1 ; of the 16 bit accumulator is at 0</pre>

# ***WOU***

Name	:	WOU - write output
Function	:	effects a writing of the physical outputs. This function is only implemented on Z targets (and varies following the target) See the documentation related to each executor for more information
Variables	:	none
Addressing	:	immediate
Also see	:	<u>RIN</u>

# ***XRA***

Name	:	XRA - xor accumulator
Function	:	effects an EXCLUSIVE OR on the 16 bit accumulator and a word or a constant, effects an EXCLUSIVE OR on the 32 bit accumulator and a long or a constant
Variables	:	M or %MW, L or %MD
Addressing	:	absolute, indirect, immediate
Also see	:	<u>ORA</u> , <u>ANA</u> ,
Example	:	 xra %1111111100000000 ; inverts the 8 bits of higher weight of the 16 bit accumulator  xra 1L ; inverts the lower weight bit of the 32 bit accumulator

## Macro-instruction

Macro-instructions are new literal language instructions which hold a set of basic instructions.

Call up syntax for a macro-instruction:

« %<Macro-instruction name\*> {parameters ...} »

Statement syntax for a macro-instruction:

```
#MACRO  
<program>  
#ENDM
```

This statement is found in a file with the name of the macro-instruction and the extension « .M ».

The file M can be placed in a sub-directory « lib » of the AUTOSIM installation directory or in project resources.

Ten parameters can be passed to the macro-instruction. When called up these parameters are placed on the same line as the macro-instruction and are separated by a space

The syntax « {?n} » in the macro-instruction program refers to the n parameter.

Example:

We are going to create a « square » macro-instruction which raises the first parameter of the macro-instruction to its square and puts the results in the second parameter.

Call up of the macro-instruction:

```
lda 3  
sta m200  
%square m200 m201  
; m201 will contain 9 here
```

« SQUARE.M » file:

```
#MACRO  
lda {?0}  
mla {?0}  
sta {?1}  
#ENDM
```

---

\* The name of the macro-instruction can be a complete access path to the file « .M », it can contain a read and directory designation.

## Libraries

A library is used to define the resources which will be compiled one time in an application, no matter how many times those resources are called up.

### Syntax for defining a library:

```
#LIBRARY <Library name>  
  
<program>  
  
#ENDL
```

<library name > is the function name which will be called up for a `jsr:<library name>` instruction:

The first time the library code is called up by the compiler its code is compiled. The following times, the call up is simply directed to the existing routine..

This mechanism is especially suited to the use of function blocks and macro-instructions to limit the generation of codes in the event that there is multiple use of the same program resources.

Words m120 to m129 are reserved for libraries and can be used for passing parameters.

## Pre-defined macro-instructions

Inversion macro-instructions are in the sub-directory « LIB » of the AUTOSIM installation directory.

Functional block equivalents are also present.

## Description of pre-defined macro-instructions

### Conversions

```
%ASCTOBIN <first two digits> <last two digits> <binary result>
```

Effecting a hexadecimal ASCII conversion (first two parameters) to binary (third parameter), by exiting the accumulator containing \$FFFF if the first two parameters are not valid ASCII numbers, otherwise 0. All the parameters are 16 bit words.

```
%BCDTOBIN <value in BCD> <binary value>
```

Effecting a BCD conversion to binary. In the output of the accumulator containing \$FFFF if the first parameter is not a valid bcd number, otherwise 0. The two parameters are 16 bit words.

```
%BINTOASC <binary value> <upper part result> <lower part result>
```

Effecting a binary conversion (first parameter) to hexadecimal ASCII (second and third parameters). All parameters are 16 bit words.

```
%BINTOBCD <binary value> <BCI value>
```

Effecting a BCD (first parameter) conversion to binary (second parameter). In the accumulator containing \$FFFF if the binary number can be converted in BCD, otherwise 0.

`%GRAYTOB <GRAY code value> <binary value>`

Effecting a Gray code conversion (first parameter) to binary (second parameter).

### Treatment on word tables

`%COPY <first word table source> <first word table destination> <number of words>`

Copy a table of source words to a table of destination words. The length is given by the number of words.

`%COMP <first word table 1> <first word table 2> <number of words> <result>`

Compares two tables of words. The result is a binary variable which takes the value 1 if all the elements in table 1 are identical to those in table 2.

`%FILL <first word table> <value> <number of words>`

Fills a word table with a value.

### Treatment on strings

The coding of strings is as follows: one character per word, one word containing the value 0 indicates the end of the chain. In macro-instructions the strings are passed in parameters by designating by the first word they are composed of.

`%STRCPY <source string> <destination string>`

Copies a string to another.

`%STRCAT <source string> <destination string>`

Adds the source string to the end of the destination string.

`%STRCMP <string 1> <string 2> <result>`

Compares to strings. The result is a boolean variable which passes to 1 if the two strings are identical.

`%STRLEN <string> <result>`

Places the length of the string in the result word.

`%STRUPR <string>`

Transforms all the characters of the string into capital letters.

`%STRLWR <string>`

Transforms all the characters of the string into lower case letters.

Example:

Conversion of m200 (binary) to m202, m203 in 4 digits (ASCII bcd)

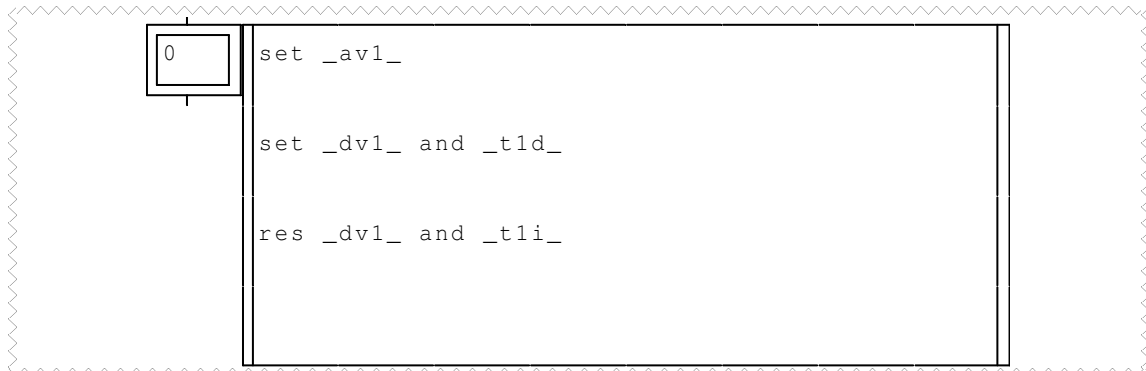
```
%bintobcd m200 m201
```

```
%bintoasc m201 m202 m203
```

### Example of low level literal language

Conditions: let's start with the simplest example: round trip of a locomotive on track 1.

Solution:



 Example\lit\low level literal1.agn

A more evolved example.

Conditions:

The locomotive must make a 10 second delay at the right end of the track and a 4 second delay at the left end.

Solution:



0	<pre> \$t0=100,40  equ u100 and  _t1i_ and  _t1d_  equ u101 orr t0 eor t1  equ _av1_ orr u100 eor u101  set _dv1_ and _t1d_  equ t0 and _t1d_  res _dv1_ and _t1i_  equ t1 and _t1i_ </pre>
---	---

 Example\lit\low level literal 2.agn

Another example:

Conditions: Make all of the model lights flash:

Solution:

0

```

; table contenant l'adresse de tous les feux
$_table_=123,?_s1d_,?_s1i_,?_s2a_,?_s2b_
$...=?_s3d_,?_s3i_,?_s4a_,?_s4b_
$...=?_s5i_,?_s5d_,?_s6d_,?_s6i_
$...=?_s7i_,?_s7d_,?_s8d_,?_s8i_
$...=-1

; initialise l'index sur le debut de la table
lda $_table_
sta _index_

:boucle:
; la valeur -1 marque la fin de la table
jmp :fin: and m(_index_)=-1

; inverser la sortie
lda m(_index_)

sta _index2_

inv o(_index2_)

inc _index_

jmp :boucle:

:fin:

```

 Example\lit\low level literal 3.agn

This example shows the use of presettings. They are used here to create a variable address table. The table contains the addresses of all the outputs which pilot the model lights.

For each execution cycle, the state of all the lights is inverted.

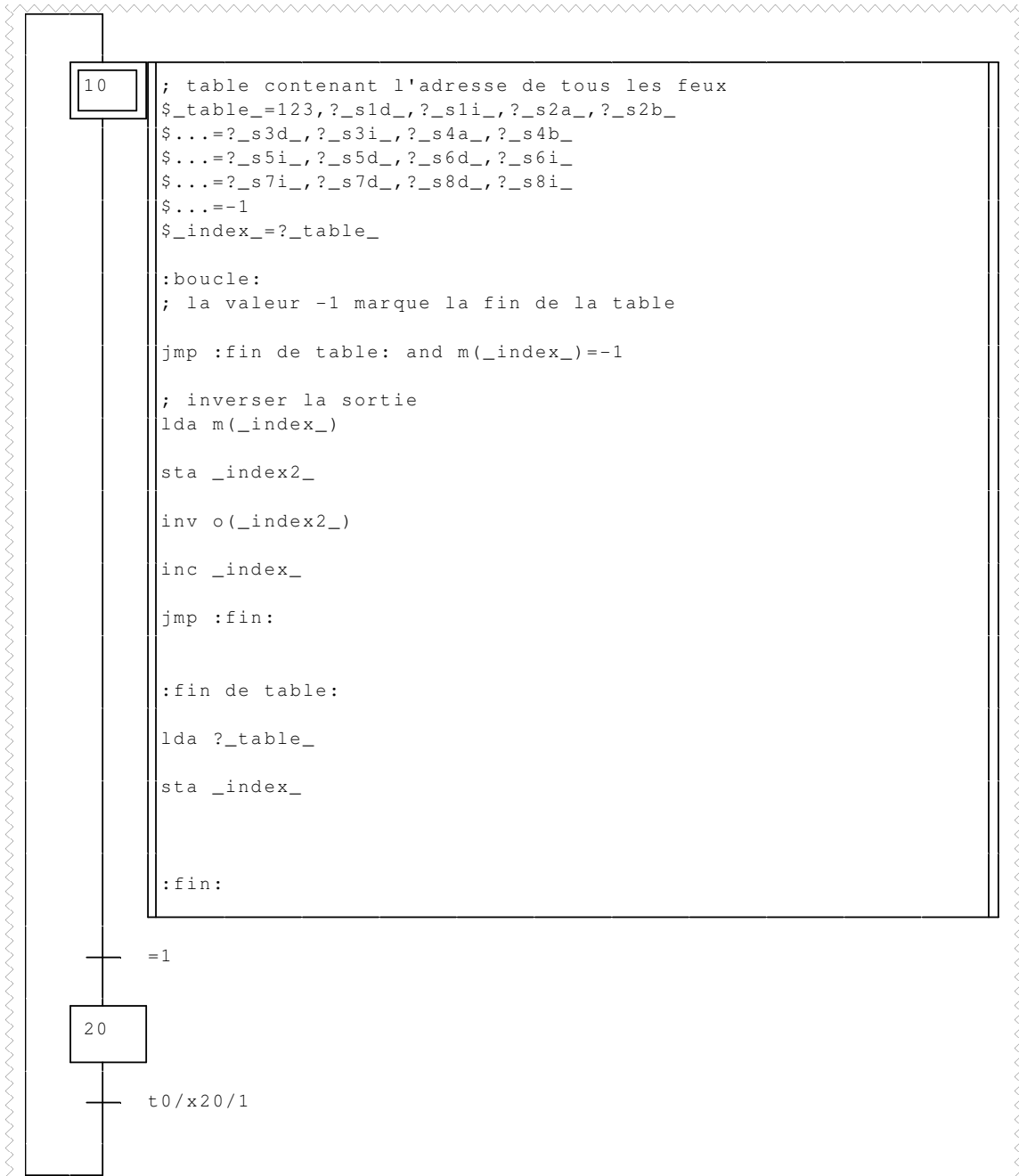
A problem occurs, all the lights flash very quickly and it is hard to see much.

Let's modify our example.

Conditions:

The state of all the lights must remain inverted every ten seconds one by one.

## Solution:



 Example\lit\low level literal 4.agn

## Extended literal language

Extended literal language is a subset of low level literal language. It is used for writing boolean and numeric equations more simply and concisely.

It is still possible to write structures like IF ... THEN ... ELSE and WHILE ... ENDWHILE (loop).

Use of extended literal language is subject to the same rules as low level literal language, it uses the same syntax for variables, mnemonics, the test types (fronts, complement state, immediate state) and addressing modes.

It is possible to mix low level literal language with extended literal language.

When the compiler of literal language detects a line written in extended literal language, it decomposes it into low level literal language instructions, then compiles it.

## Writing boolean equations

### General syntax:

```
« bool. variable=(assignment type) (bool. variable 2 operator 1 bool. variable
3... operator n -1 bool. variable n ) »
```

The type of assignment must be indicated if it is other than « Assignment »

It can be:

⇒ « (/) »: complement assignment,

⇒ « (0) »: reset,

⇒ « (1) »: set to one.

The operators can be:

⇒ « . »: and,

⇒ « + »: or.

The equations can contain various levels of parentheses to indicate the evaluation order. By default, the equations are evaluated from the left towards the right.

### Examples and equivalencies with low level literal language

<code>o0=(i0)</code>	<code>equ o0 and i0</code>
<code>o0=(i0.i1)</code>	<code>equ o0 and i0 and i1</code>
<code>o0=(i0+i1)</code>	<code>equ o0 orr i0 eor i1</code>
<code>o0=(1)</code>	<code>set o0</code>
<code>o0=(0)</code>	<code>res o0</code>
<code>o0=(1) (i0)</code>	<code>set o0 and i0</code>
<code>o0=(0) (i0)</code>	<code>res o0 and i0</code>
<code>o0=(1) (i0.i1)</code>	<code>set o0 and i0 and i1</code>
<code>o0=(0) (i0+i1)</code>	<code>res o0 orr o0 eor i1</code>
<code>o0=(/) (i0)</code>	<code>neq o0 and i0</code>
<code>o0=(/) (i0.i1)</code>	<code>neq o0 and i0 and i1</code>
<code>o0=(/i0)</code>	<code>equ o0 and /i0</code>
<code>o0=(/i0./i1)</code>	<code>equ o0 and /i0 and /i1</code>
<code>o0=(c0=10)</code>	<code>equ o0 and c0=10</code>
<code>o0=(m200&lt;100+m200&gt;200)</code>	<code>equ o0 orr m200&lt;100 eor m200&gt;200</code>

## Writing numeric equations

### General equations for integers:

« num. variable 1=[num. variable 2 operator 1 ... operator n-1 num. variable n] »

The equations can contain various levels of braces for indicating the evaluation order. By default, the equations are evaluated from left to right. Operators for 16 and 32 bit integers can be:

« + »: addition (equivalent to instruction ADA),  
 « - »: subtraction (equivalent to instruction SBA),  
 « \* »: multiplication (equivalent to instruction MLA),  
 « / »: division (equivalent to instruction DVA),  
 « < »: shift to left (equivalent to instruction RLA),  
 « > »: shift to right (equivalent to instruction RRA),  
 « & »: « And » binary (equivalent to instruction ANA),  
 « | »\*: « Or » binary (equivalent to instruction ORA),  
 « ^ »: « Exclusive or » binary (equivalent to instruction XRA).

### Operators for floats can be:

⇒ « + »: addition (equivalent to instruction ADA),  
 ⇒ « - »: subtraction (equivalent to instruction SBA),  
 ⇒ « \* »: multiplication (equivalent to instruction MLA),  
 ⇒ « / »: division (equivalent to instruction DVA).

It is possible to indicate the constant in float equations. If this is necessary use the presettings on floats.

Equations on floats can call up the « SQR » and « ABS » functions

Note: depending on the complexity the compiler may use intermediate variables. These variables are the words m53 to m59 for 16 bit integers, the longs l53 to l59 for 32 bit integers and the floats f53 à f59.

### Examples and equivalencies with low level literal language

M200=[10]	lda 10
	sta m200
M200=[m201]	lda m201
	sta m200
M200=[m201+100]	lda m201
	ada 100
	sta m200
M200=[m200+m201-m202]	lda m200
	ada m201

---

\* This character is normally associated to the [ALT] + [6] keys on keyboards

	sba m202
	sta m200
M200=[m200&\$ff00]	lda m200
	ana \$ff00
	sta m200
F200=[f201]	lda f201
	sta f200
F200=[f201+f202]	lda f201
	ada f202
	sta f200
F200=[sqr[f201]]	lda f201
	sqr aaa
	sta f200
F200=[sqr[abs[f201*100R]]]	lda f201
	m1a 100R
	abs aaa
	sqr aaa
	sta f200
L200=[l201+\$12345678L]	lda l201
	ada \$12345678L
	sta l200

## IF...THEN...ELSE...structure

### General syntax:

```

IF(test)
    THEN
        action if true test
    ENDIF
    ELSE
        action if false test
    ENDIF

```

The test must comply with the syntax described in the chapter dedicated to boolean equations.

Only if an action tests true or tests false can it appear.

It is possible to connect multiple structures of this type.

System bits u90 to u99 are used as temporary variables for managing this type of structure.

### Examples:

```

IF(i0)
    THEN
        inc m200                ; increments word 200 if i0
    ENDIF

```

```

IF (i1+i2)
    THEN
        m200=[m200+10]      ; adds 10 to word 200 if the or i2
    ENDIF
    ELSE
        res m200            ; else effect m200
    ENDIF

```

## WHILE ... ENDWHILE structure

### General syntax:

```

WHILE(test)
    action is repeated as long as the test is true
ENDWHILE

```

The test must comply with the syntax described in the chapter dedicated to boolean equations.

It is possible to connect multiple structures of this type.

System bits u90 to u99 are used as temporary variables for managing this type of structure.

### Examples:

```

m200=[0]
WHILE (m200<10)
    set o(200)
    inc m200      ; increments word 200
ENDWHILE

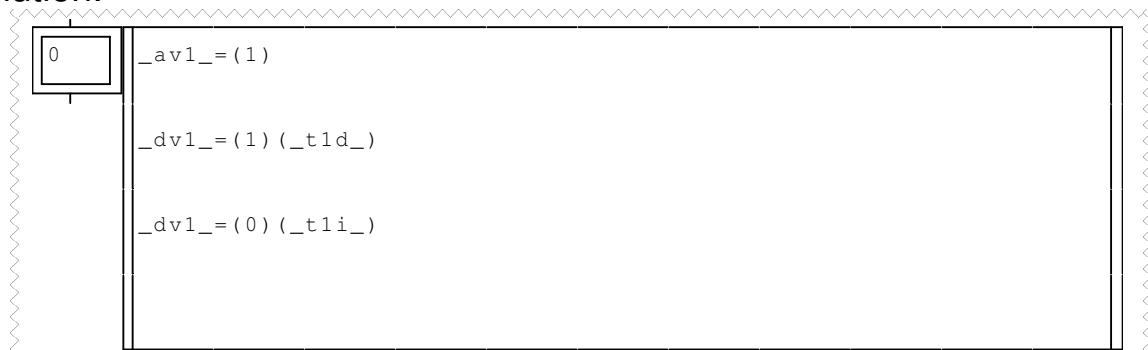
```

This example sets outputs o0 to o9 to one.

## Example of a program in extended literal language

Let's go back to the example from the previous chapter

### Solution:



 Example\lit\extended literal 1.agn

Let's complicate our example with some calculations

Conditions:

Calculate the speed in millimeters per second and meters per hour of the locomotive on the left to right trajectory.

## Solution:



 Example\lit\extended literal 2.agn

Word 32 is used to read the system time. The value is then transferred to the float to effect the calculations without compromising exactness.

## ST literal language

ST literal language is a structured literal language defined by IEC1131-3 standard. This language is used to write boolean and numeric equations as well as program structures.

## General Information

ST literal language is used in the same way as low level literal language and extended literal language.



Commands are used to establish the sections in ST literal language  
« #BEGIN\_ST » indicates the beginning of an ST language section.  
« #END\_ST » indicates the end of an ST language section.

Example:

```
m200=[50]           ; extended literal language
#BEGIN_ST
m201:=4;             (* ST language *)
#END_ST
```

It is also possible to choose to use ST language for an entire sheet.  
This selection is made in the properties dialogue box on each sheet.

On a sheet where ST language is the default language it is possible to enter low level and extended literal language by using the commands « #END\_ST » and « #BEGIN\_ST ».

Comments for ST language must start with « (\* » and end with « \*) ».

ST language instructions end with the character « ; ». Multiple instructions can be written on the same line.

Example:

```
o0:=1; m200:=m200+1;
```

## Boolean equations

The general syntax is:

```
variable:= boolean equation;
```

Boolean equations can be composed of a constant, a variable or multiple variables separated by operators.

Constants can be: 0, 1, FALSE or TRUE.

Examples:

```
o0:=1;
o1:=FALSE;
```

The operators used to separate multiple variables are: + (or), . (and), OR or AND.

« And » has priority over « Or ».

**Example:**

```
o0:=i0+i1.i2+i3;
```

**Will be treated as:**

```
o0:=i0+(i1.i2)+i3;
```

Parentheses can be used in the equations to indicate priorities.

**Example:**

```
o0:=(i0+i1).(i2+i3);
```

Numeric tests can be used.

**Example:**

```
o0:=m200>5.m200<100;
```

## Numeric equations

**The general syntax is:**

```
variable:= numeric equation;
```

Numeric equations can be composed of a constant, a variable or multiple variables separated by operators.

The constants can be expressed as decimal, hexadecimal (prefix #16) or binary (prefix #2) values.

**Examples:**

```
m200:=1234;
```

```
m201:=16#aa55;
```

```
m202:=2#100000011101;
```

Operators are used to separate multiple variables or constants in their order of priority.

\* (multiplication), / (division), + (addition), - (subtraction), & or AND (binary and), XOR (binary exclusive or), OR (binary or).

**Examples:**

```
m200:=1000*m201;
```

```
m200:=m202-m204*m203; (* equivalent to m200:=m202-(m204*m203) *)
```

Parentheses can be used in the equations to indicate priority.

**Example:**

```
m200:=(m202-m204)*m203;
```

## Programming structures

### IF THEN ELSE test

**Syntax:**

```
IF condition THEN action ENDIF;
```

**and**

```
IF condition THEN action ELSE action ENDIF;
```

**Example:**

```
if i0
    then o0:=TRUE;
    else
        o0:=FALSE;
        if i1 then m200:=4; endif;
endif ;
```

### WHILE loop

**Syntax:**

```
WHILE condition DO action ENDWHILE;
```

**Example:**

```
while m200<1000
    do
        m200:=m200+1;
    endwhile;
```

### REPEAT UNTIL loop

**Syntax:**

```
REPEAT action UNTIL condition; ENDREPEAT;
```

**Example:**

```
repeat
    m200:=m200+1;
until m200=500
endrepeat;
```

## FOR TO loop

### Syntax:

```
FOR variable:=start value TO end value DO action ENDFOR;
```

or

```
FOR variable:=start value TO end value BY no DO action ENDFOR;
```

### Example:

```
for m200:=0 to 100 by 2
    do
        m201:=m202*m201;
    endfor;
```

## Exiting a loop

The key word « EXIT » is used to exit a loop.

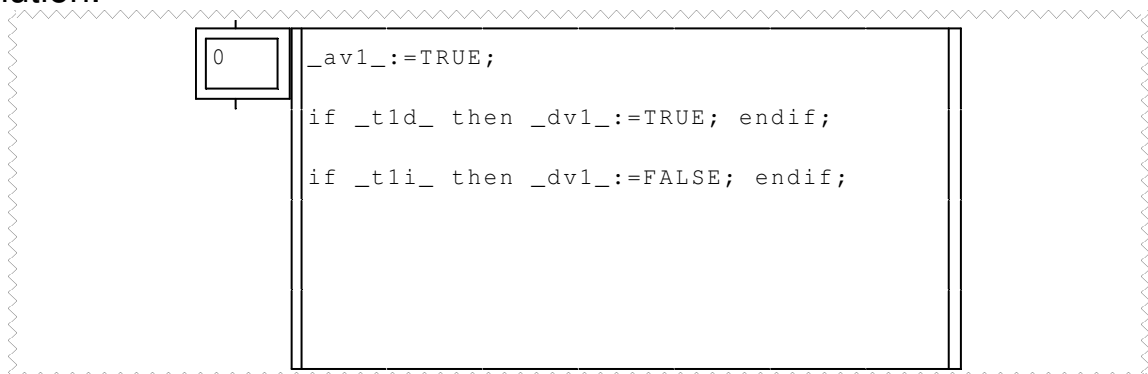
### Example:

```
while i0
    m200:=m200+1;
    if m200>1000 then exit; endif;
endwhile;
```

## Example of a program in extended literal language

Let's go back to our example in the previous chapter

### Solution:



 Example\lit\ST literal 1.agn

## Organization chart

AUTOSIM implements a « organization chart » type program. Literal languages must be used with this type of program. See the previous chapters to learn how to use these languages.

The basis of programming with an organizational chart form is the graphic representation of an algorithmic treatment.


Unlike Grafcet language, programming in the organizational chart form generates a code which will be executed one time per search cycle. This means that it is not possible to remain in an organizational chart rectangle it is mandatory for the execution to exit the organizational chart to continue to execute the rest of the program..


This is a very important point and must not be forgotten when this language is selected.

Only rectangles can be drawn. The contents of a rectangle and its connections determine if the rectangle is an action or a test.

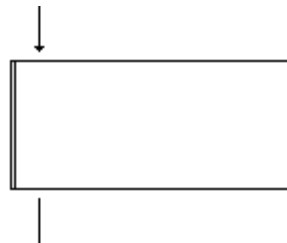
### Creating an organizational chart



The rectangles are drawn by selecting the command « Add ... / Code box» from the menu (click on the right side of the mouse on the bottom of the sheet to open the menu).

It is necessary to place a block  (key [<]) at the entry of each rectangle, this must be placed on the upper part of the rectangle.

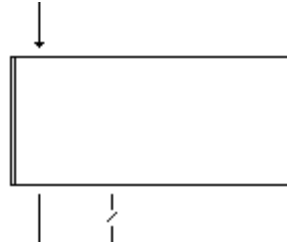
If the rectangle is an action it will have only one exit represented by a block  (key [E]) on the lower left side of the rectangle.

An action rectangle:



If the rectangle is a test it must have two outputs. The first is represented by a block  (key [E]) on the lower left side and is for a true test, the second represented by a block  (key [=]) is immediately to the right of the other output and is for a false test.

A test rectangle:



The branches of the organizational chart must always end with a rectangle without an output that could remain empty.

## Rectangle content

### Action rectangle content

Action rectangles can contain any kind of literal language instructions.

### Test rectangle content

Test rectangles must contain a test that complies with the test syntax of the IF...THEN...ELSE... type structure of extended literal language.

For example:

```
IF (i0)
```

It is possible to write actions before this test in the test rectangle.

This can be used to make certain calculations before the test

For example, if we want to test if the word 200 is equal to the word 201 plus 4:

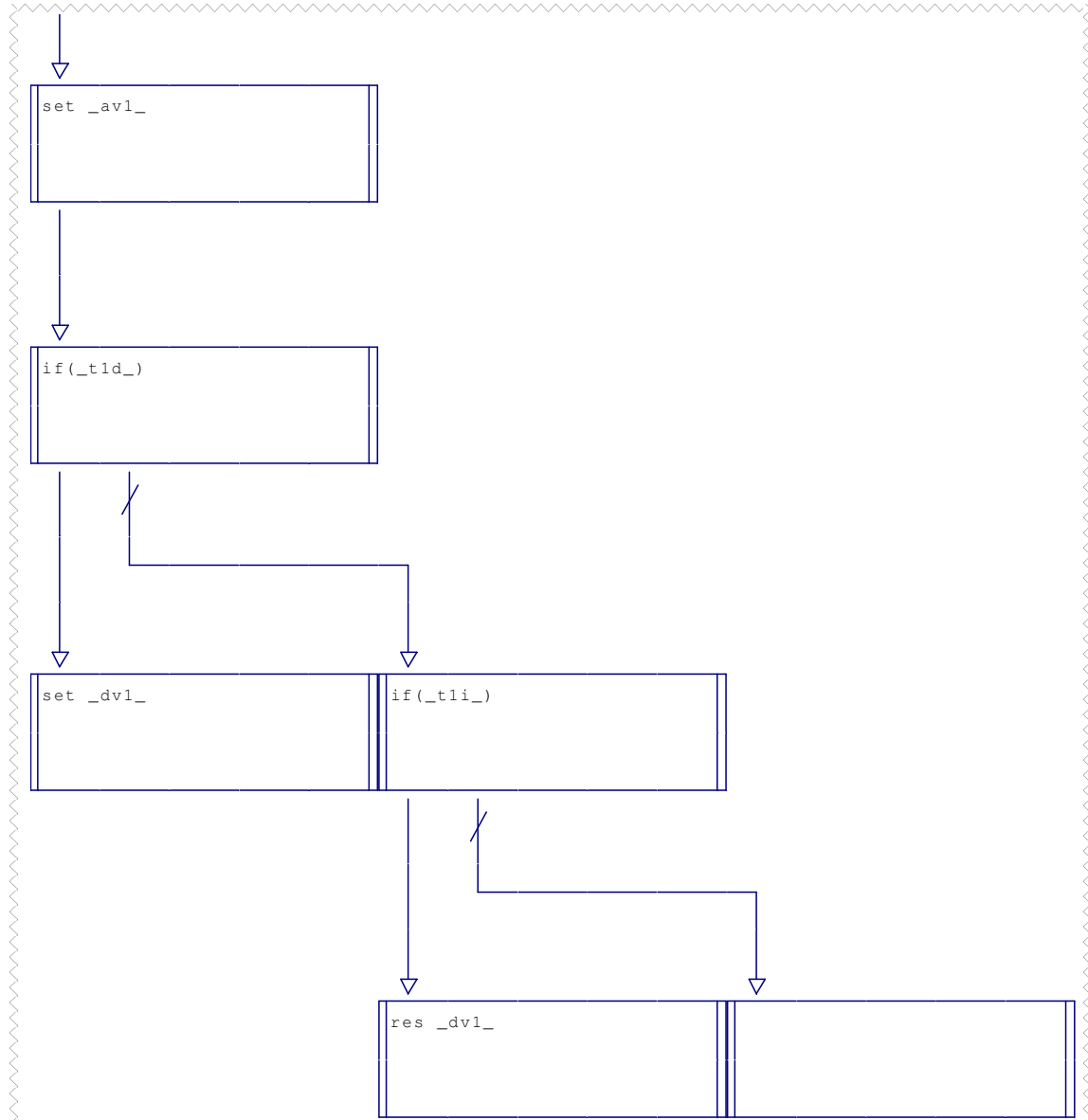
```
m202=[m201+4]
```

```
IF (m200=m202)
```

## Illustration

Our first, now typical, example is to make a locomotive make round trips on track 1 of the model.

Solution:



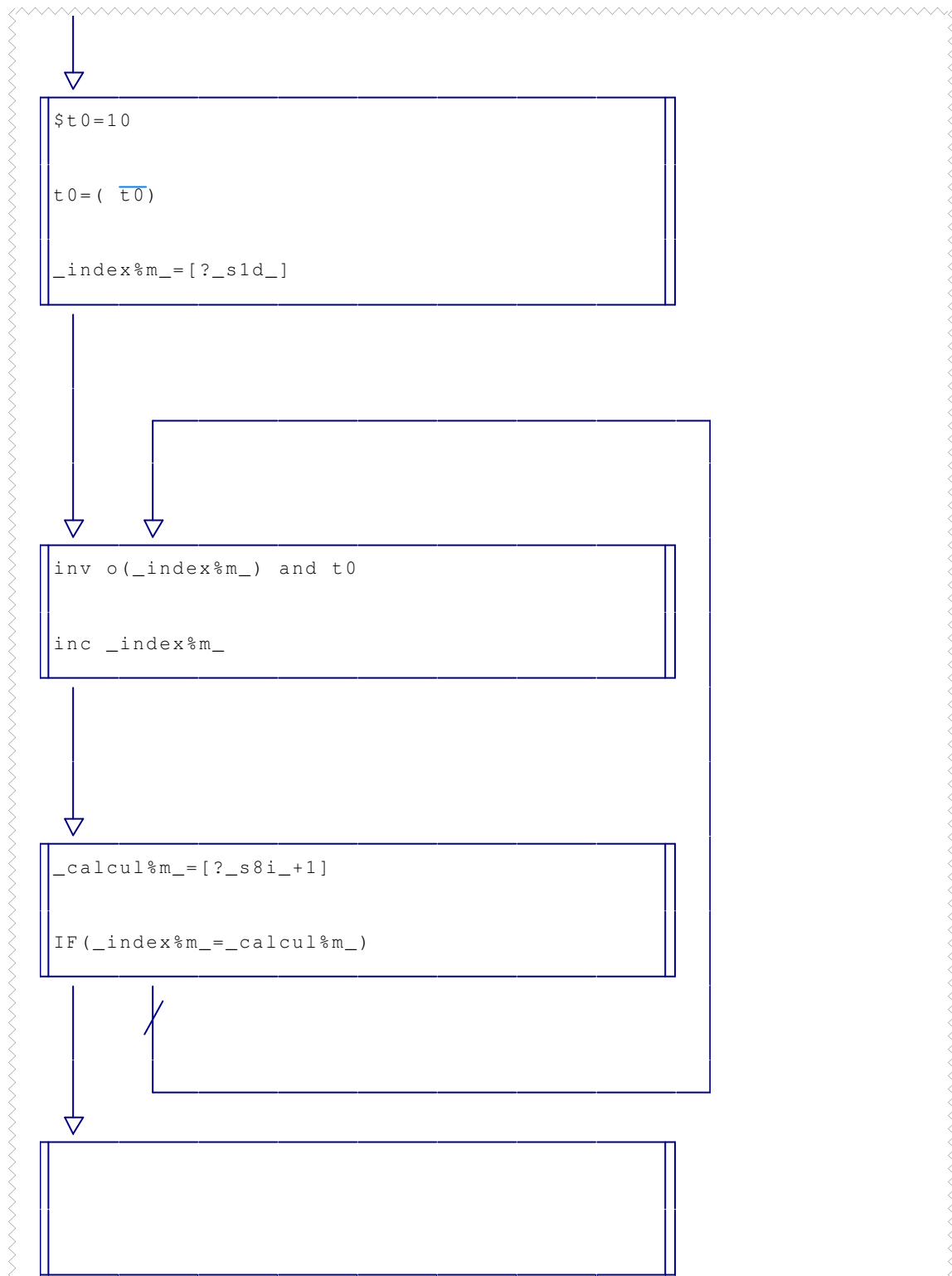
 Example\ Ornagization chart\ Ornagization chart 1.agn

Second example

Conditions:

Make all the model light flash. The light change states every second.

Solution:



 Example\ Ornagization chart\ Ornagization chart 2.agn

Note the use of automatic symbols in this example.



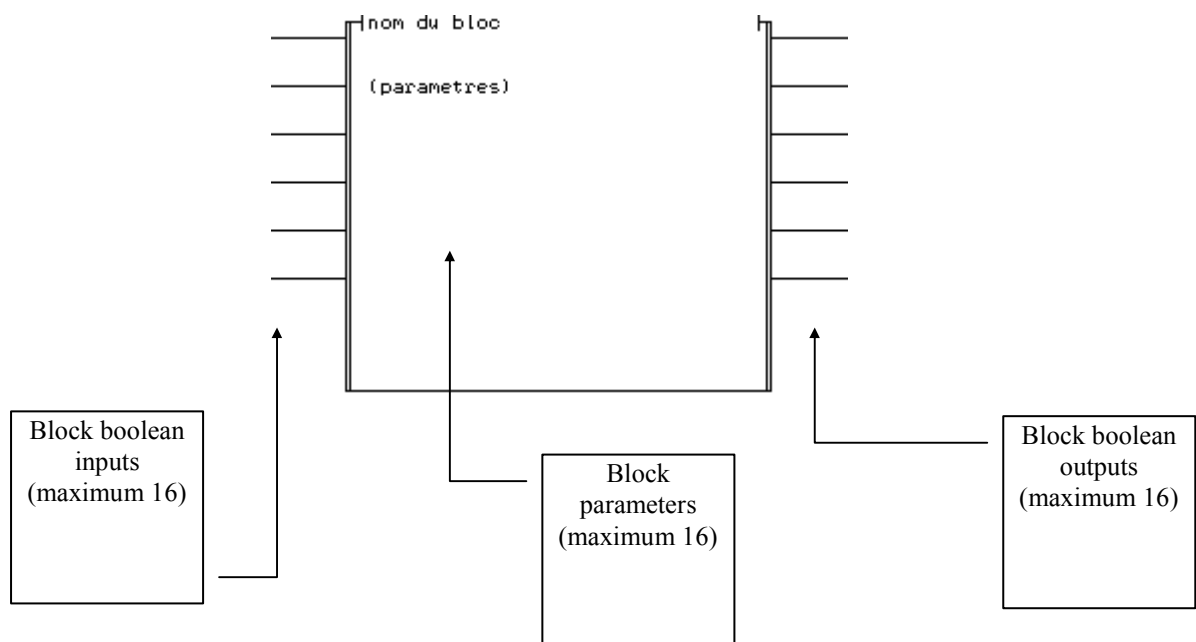
## Function blocks

AUTOSIM implements the use of function blocks.

This modular programming method is used to associate a set of instructions written in literal language to a graphic element .

Function blocks are defined by the programmer. Their number is not limited. It is possible to create sets of function blocks to allow a modular and standardize concept of applications.

Function blocks are used within flow chart or ladder type models, they have n boolean inputs and n boolean outputs. If the block is going to treat variables which are not boolean, then they will be mentioned in the drawing of the function block. The inside of the block can receive parameters: constant or variable.



### Creating a function block

A function block is composed of two separate files. One file has « .ZON » extension which contains the drawing of the function block and a file with « .LIB » extension which contains a series of instructions written in literal language which establish the functionality of the function block.

The « .ZON » and « .LIB » files must bear the name of the function block. For example, if we decide to create a function block « MEMORY », we need to create the files « MEMORY.ZON » (to draw the block) and « MEMORY.LIB » (for the functionality of the block).

### Drawing a block and creating a « .ZON » file

The envelop of a function block is composed of a code box to which blocks dedicated for the function block are added.


To draw a function block follow the steps below:

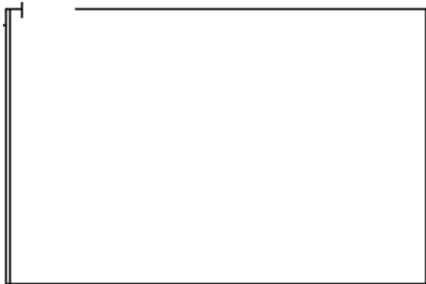
⇒ use the assistant (recommended)


Or:

⇒ draw a code box (use the command « Add .../Code box » from the menu):



⇒ place a block  (key [8]) on the upper right corner of the code box:



⇒ place a block  (key [9]) on the upper right corner of the code box:







⇒ delete the line at the top of the block (key [A] is used to place blank blocks):



⇒ click with the left side of the mouse on the upper left corner of the functional block, then enter the name of the functional block which must not be more than 8 characters (the « .ZON » and « .LIB » files must bear this name), then press [ENTER].



- ⇒ if additional boolean inputs are necessary, a block must be used  (key [;]) or  (key [:]), the added inputs must be located right below the first input, no free space should be left,
- ⇒ if additional boolean outputs are needed a block must be added  (key [>]) or  (key [?]), the added outputs must be located right below the first output, no free space should be left,
- ⇒ the interior of the block can contain comments or parameters, the parameters are written between braces « {...} ». Everything not written between braces is ignored by the compiler. It is interesting to indicate the use of boolean inputs and outputs inside the block.
- ⇒ when the block is finished, the command « Select » must be used from the « Edit » menu to select the drawing of the functional block, then save it in the « .ZON » file with the « Copy to » command from the « Edit » menu.

### Creating an « .LIB » file

The « .LIB » file is a text file containing instructions in literal language (low level or extended). These instructions establish the functionality of the function block.

A special syntax is used to refer to block boolean inputs, block boolean outputs and block parameters.

To refer to a block boolean input, use the syntax « {Ix} » where x is the number of the boolean input expressed in hexadecimal (0 to f).

To refer to a block boolean output, use the syntax « {Ox} » where x is the number of the boolean output expressed in hexadecimal (0 to f).

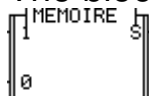
To refer to a block parameter use the syntax « {?x} » where x is the number of the parameter in hexadecimal (0 to f).

The .LIB can be placed in the « lib » sub-directory of the AUTOSIM installation directory or in the project resources.

## Simple example of a function block

We are going to create a « MEMORY » function block which contains two boolean inputs (set to one and reset) and a boolean output (memory state).

The block drawing contained in the « MEMORY.ZON » file is:

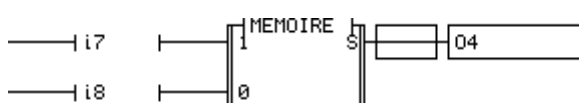


Block functionality contained in the « MEMORY.LIB » file is:

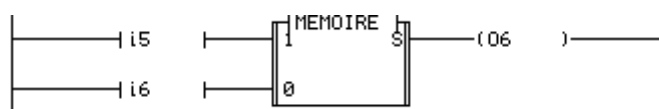
```
{O0}=(1)({I0})
```

```
{O0}=(0)({I1})
```

The block can then be used in the following way:



or



To use a function block in an application, select the command «Paste from» from the «Edit» menu and select the «.ZON» file corresponding to the function block used.

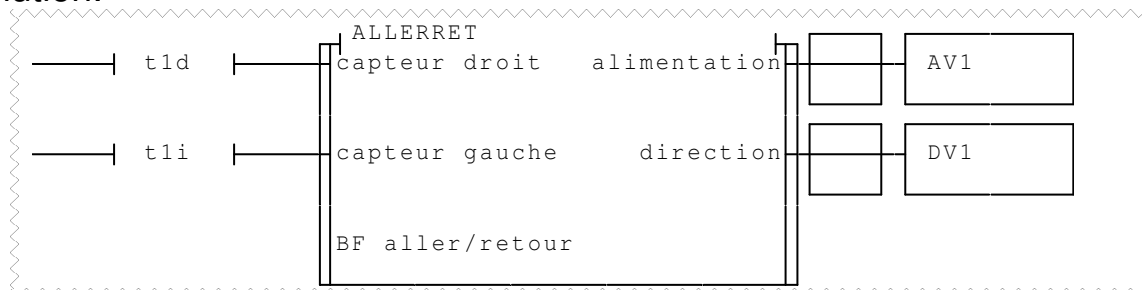
## Illustration

Let's go back to our typical example.

Conditions:

Round trip of a locomotive on track 1 of the model.

Solution:



 Example\fb\fb 1.agn

```
; bloc fonctionnel ALLERRET
```

```
; aller retour d'une locomotive sur une voie
```

; les entrées booléennes sont les fins de course  
; les sorties booléennes sont l'alimentation de la voie (0) et la direction (1)

; toujours alimenter la voie  
set {O0}

; piloter la direction en fonction des fins de course

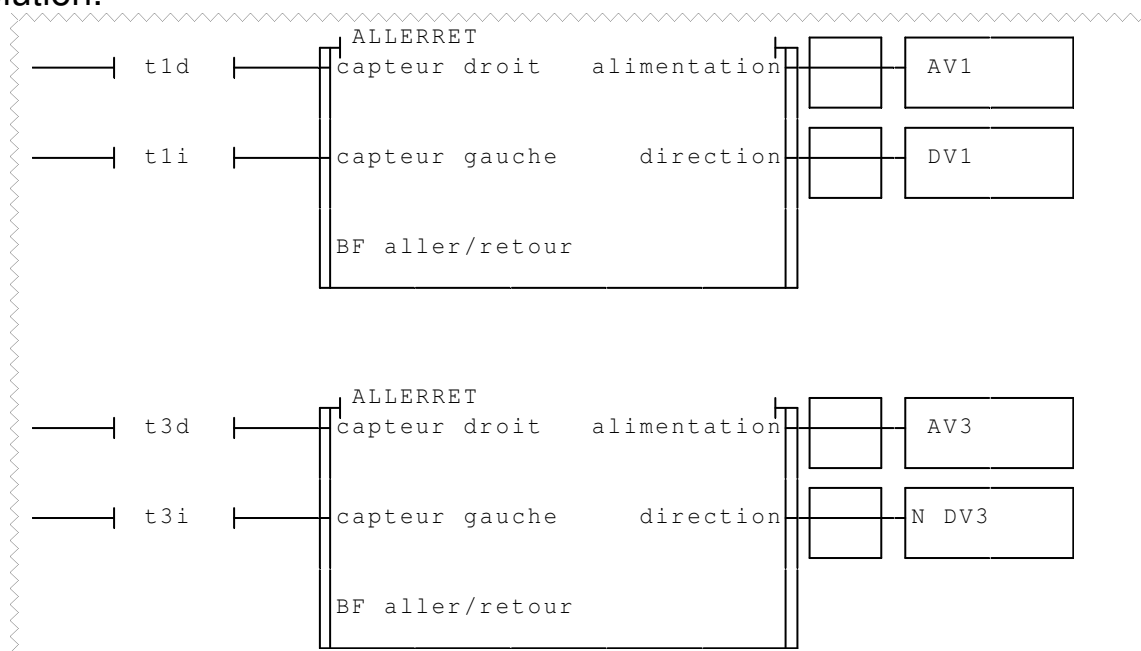
{O1}=(1)({I0})  
{O1}=(0)({I1})

To illustrate the use of function blocks, let's complete our example.

Conditions:

Round trip of two locomotives on tracks 1 and 3.

Solution:



 Example\fb\fb 2.agn

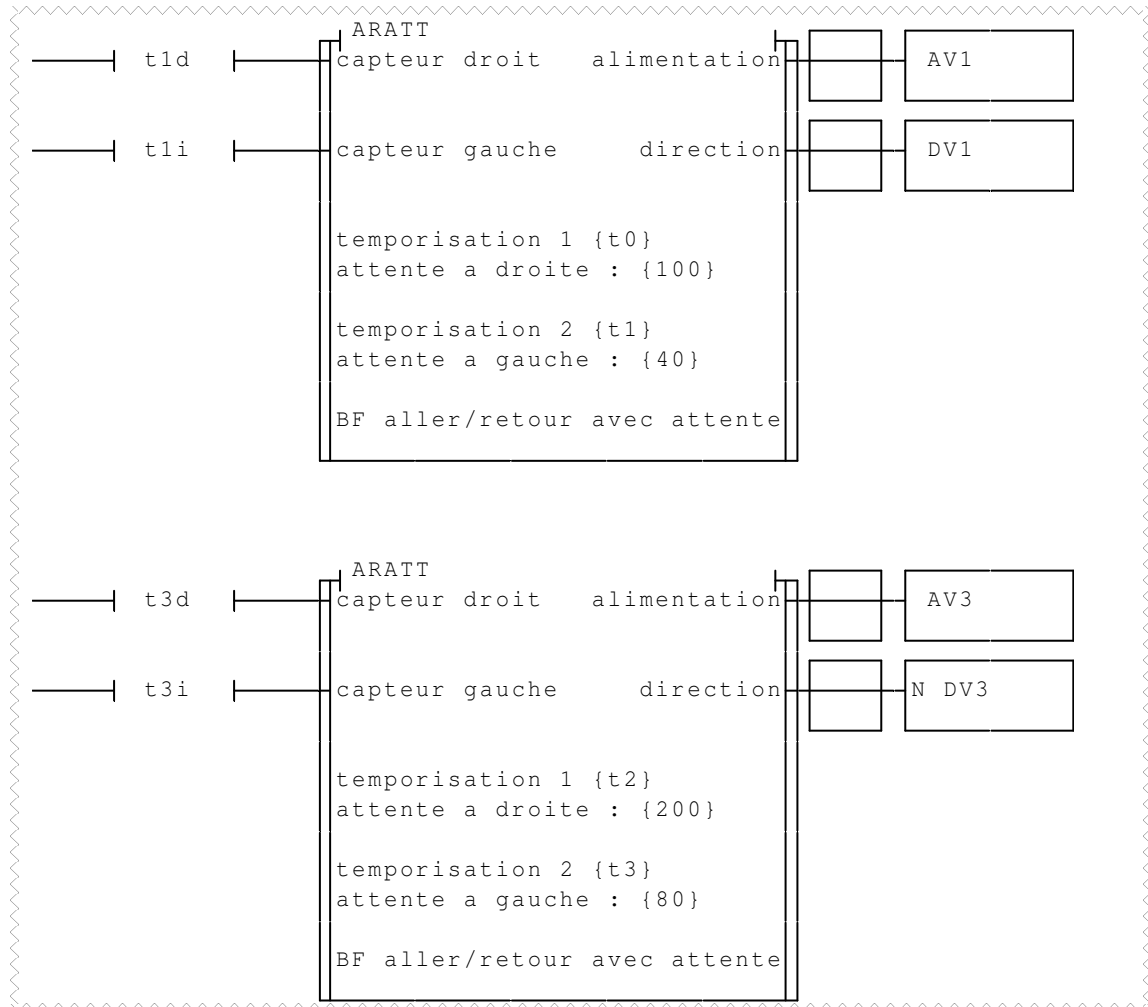
This example shows that with the same function block it is easy to make different modules of an operative party function in the identical manner.

Let's complete our example to illustrate the use of parameters

Conditions:

The two locomotives must make a delay at the end of the track. For locomotive 1: 10 seconds on the right and 4 seconds on the left, for locomotive 2: 20 seconds on the right and 8 seconds on the left.

# Solution:



```

; bloc fonctionnel ARATT
; aller retour d'une locomotive sur une voie avec attente
; les entrées booléennes sont les fins de course
; les sorties booléennes sont l'alimentation de la voie (0) et la
direction (1)
; les paramètres sont:
;           0: première temporisation
;           1: durée de la première temporisation
;           2: deuxième temporisation
;           3: durée de la deuxième temporisation

; prédisposition des deux temporisations
${?0}={?1}
${?2}={?3}

; alimenter la voie si pas les fins de course ou si tempo. terminées
set {00}
res {00} orr {I0} eor {I1}
set {00} orr {?0} eor {?2}

; gestion des temporisations
{?0}={({I0})}
{?2}={({I1})}

; piloter la direction en fonction des fins de course
{O1}=(1) ({I0})
{O1}=(0) ({I1})

```

 Example\fb\fb 3.agn

## Supplementary syntax

Supplementary syntax is used to make a calculation on the reference variable numbers in the « .LIB » file.

The syntax « ~+n » added after a reference to a variable or a parameter, adds n.

The syntax « ~-n » added after a reference to a variable or a parameter subtracts n.

The syntax « ~\*n » added after a reference to a variable or parameter, multiplies by n.

It is possible to write many of these commands, one after the other, they are evaluated from left to right.

This mechanism is useful when a function block parameter needs to be used to refer to a table of variables.

### Examples:

```
{?0}~+1
```

referring to the following element the first parameter, for example if the first parameter is m200 this syntax refers to m201.

```
M{?2}~*100~+200
```

referring to the third parameter multiplied by 100 plus 200, for example if the third parameter is 1 that syntax refers to M 1\*100 + 200 thus M300.

## Evolved function blocks

This functionality is used to create very powerful function blocks with greater ease than the function blocks managed by files written in literal language. This programming method uses a functional analysis approach.

It does not matter which sheet or set of sheets become a function block (sometimes called encapsulating a program).

The sheet or sheet which describe the functionality of a function block can access variables which are outside the function block: block boolean inputs, boolean outputs and parameters.

Principles for use and more importantly the use of external variables is identical to the old function blocks.

## Syntax

To refer a variable outside a function block it is necessary to use a mnemonic included in the following text: {In} to refer the boolean input n, {On} to refer the boolean output n, {?n} to refer parameter n. The mnemonic must start with a letter.

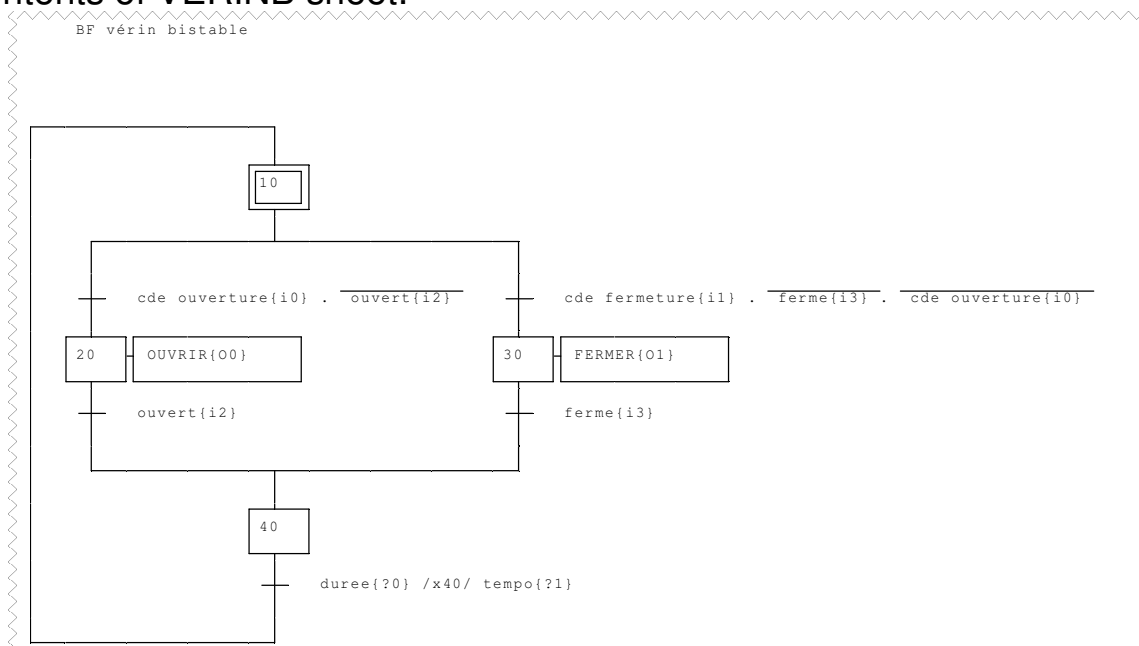
## Differentiating between new and old function blocks

The file name written on the function block drawing indicates if it is an old (managed by an LIB file) or new function block (managed by a GR7 sheet). The name of an old function block does not have an extension, for a new one the extension GR 7 must be added. The sheet containing the code which manages the functionality of the function block must be entered in the list of project sheets. In the sheet properties « Function Block » must be selected.

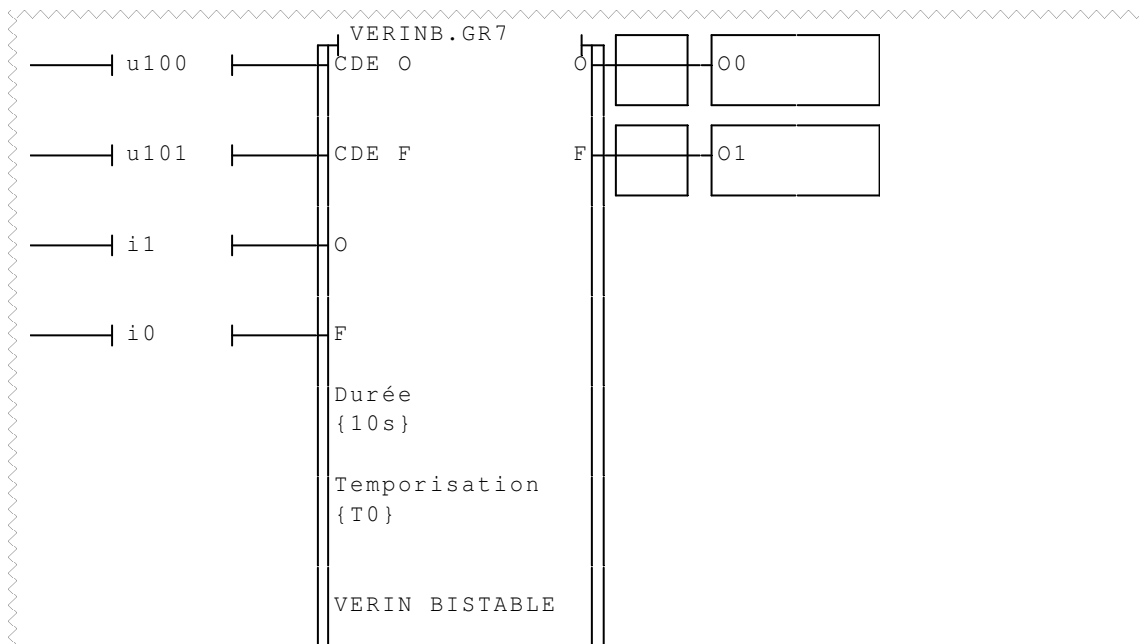


## Example

### Contents of VERINB sheet:



### Call up a function block



 Example\fb\Fb with sfc inside.agn

## Predefined function blocks

Conversion function blocks are located in the sub-directory « \LIB » of the directory where AUTOSIM is installed. The equivalents in macro-instructions are also present.

To insert a function blocks and its parameters in an application select « Pre-set function block » from the « Assistant / Function block» dialog box.

### Conversion blocks

ASCTOBIN: converts ASCII to binary  
BCDTOBIN: converts BCD to binary  
BINTOASC: converts binary to ASCII  
BINTOBCD: converts binary to BCD  
GRAYTOB: converts gray code to binary  
16BINTOM: transfers 16 boolean variables to a word  
MTO16 BIN: transfers a word to 16 boolean variables

### Time delay blocks

TEMPO: upstream time delay  
PULSOR: parallel output  
PULSE: time delay pulse

### String blocks

STRCMP: comparison  
STRCAT: concatenation  
STRCPY: copy  
STRLEN: calculate the length  
STRUPR: set in lower case  
STRLWR: set in upper case

### Word table blocks

COMP: comparison  
COPY: copy  
FILL: fill

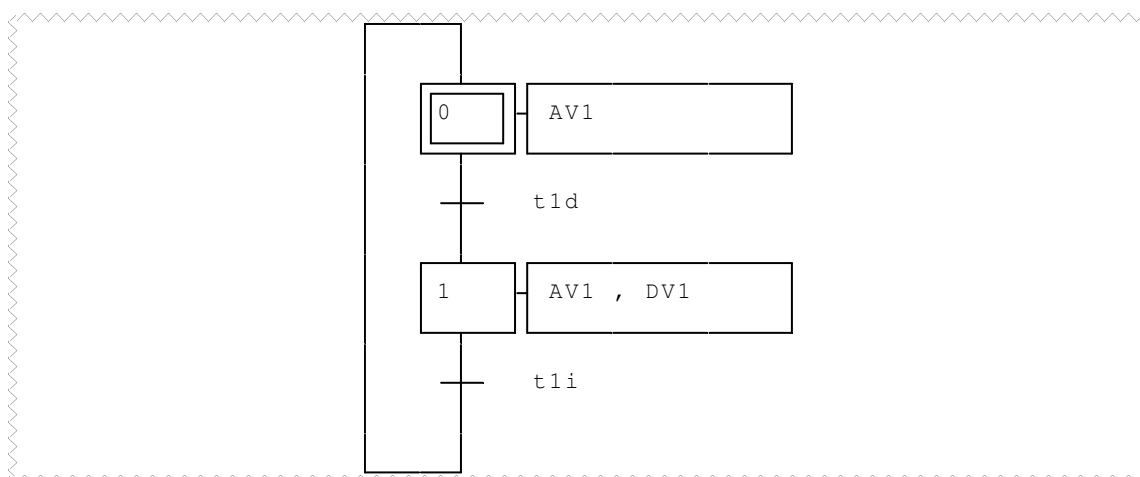
## Advanced techniques

### Compiler generated code

This chapter deal with the form of code generated by compilation of such or that type of program.

The utility « CODELIST.EXE » is used to translate « in clear » a file of intermediate code « .EQU » (also called pivot language).

We are going to do the following: load and compile the first programming example in the « Grafcet » chapter: « simple1.agn » from the directory « Example\grafcet »:



Double click on « Generated files/Pivot code » in the browser.

You will obtain the following list of instructions:

```
:00000000: RES x0      AND i0
:00000002: SET x0      AND b0
:00000004: SET x0      AND x1    AND i1
:00000007: RES x1      AND i1
:00000009: SET x1      AND x0    AND i0
; Le code qui suit a été généré par la compilation
de: 'affectations (actions Grafcet, logigrammes et
ladder) '
:0000000C: EQU o0      ORR @x0  EOR @x1
:0000000F: EQU o23     AND @x1
```

This represents the translation of a « simple1.agn » application into low level literal language

The comments indicate where the portions of code came from, this is useful if an application is composed of multiple sheets.

Obtaining this list of instructions may be useful for answering questions regarding code generated for some program form or the use of some language.

In certain cases « critiques », for which it is important to know information such as « how many cycles does it take before this action becomes true ? » a step by step way and an in-depth examination of generated code will prove to be indispensable.

### Optimizing generated code

Various levels of optimization are possible.

### Optimizing compiler generated code

The compiler optimization option is used to greatly reduce the size of generated code. This command requires that the compiler manage fewer lines of low level literal language, consequently increasing compiling time.

Depending on the post-processors used, this option involves an improvement in the size of the code and/or the execution time. It is advisable to carry out some tests to determine if this command is of interest or not depending on the nature of the program and the type of target used.

Normally, it is useful with post-processors for Z targets.

### **Optimizing post-processor generated code**

Each post-processor may possess options for optimizing generated code. For post-processors which generate construction code, see the corresponding information.

### **Optimizing cycle time: reducing the number of time delays on Z targets**

For Z targets, the number of stated time delays directly affects the cycle time. Try to state the minimum time delays based on the application requirements.

### **Optimizing cycle time: canceling scanning of certain parts of the program**

Only targets which accept JSR and RET instruction support this technique.

Special compilation commands are used to validate or « invalidate » scanning of certain parts of the program.

They are the sheets which define the portions of applications.

If an application is broken down into four sheets than each one can be separately « validated » or « invalidate ».

A command « #C(condition) » placed on the sheet conditions the searching of the sheet up to a sheet containing a « #R » command.

This condition must use the syntax established for the tests.

Example:

If a sheet contains the two commands:

```
#C (m200=4)
```

```
#R
```

Then everything that it contains will not be executed except word 200 containing 4.

## Examples

### Regarding examples

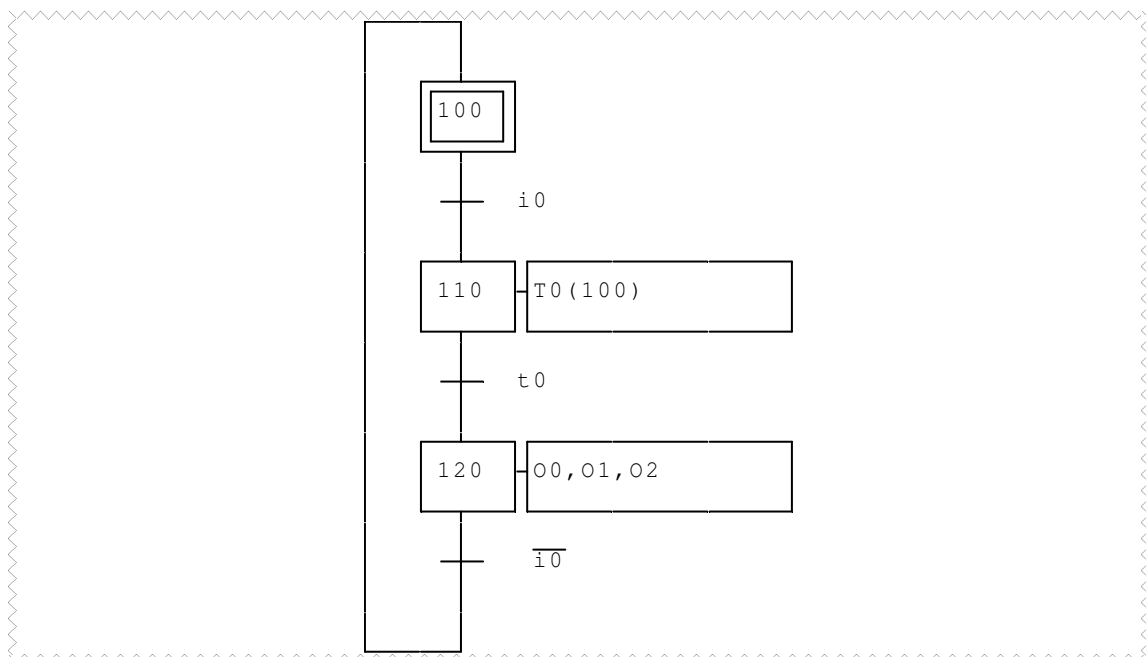
This part contains a series of examples providing an illustration of the different programming possibilities offered by AUTOSIM.

All of these examples are located in the « example » sub-directory in the directory where AUTOSIM is installed.

This section contains the most complete and complex examples developed for a train model. The description of this model is located at the beginning of the language reference manual.

### Simple grafcet

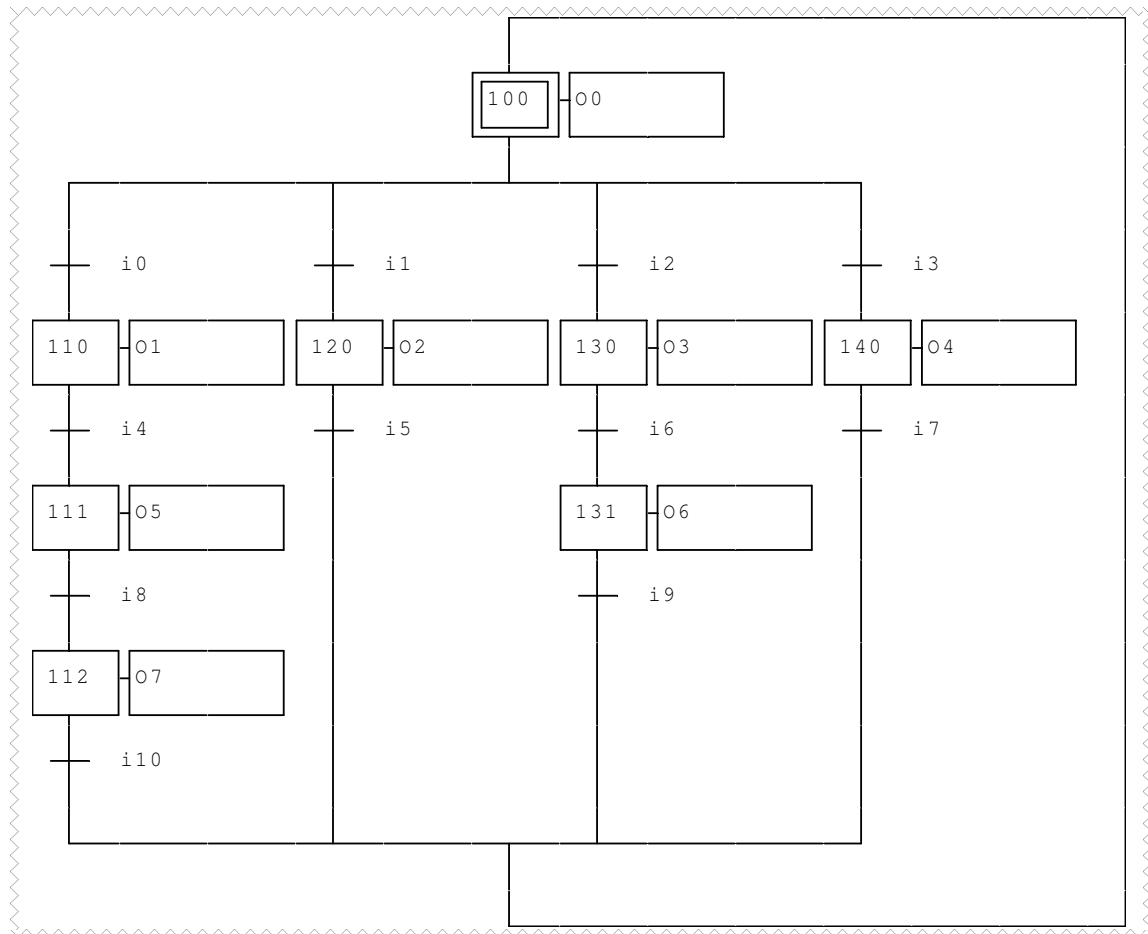
The first example is a simple line Grafcet



 Example\grafcet\sample1.agn

- ⇒ the transition in step 100 and step 110 is made up of a test on input 0,
- ⇒ step 110 activates the time delay 0 for a duration of 10 seconds, this time delay is used as a transition between step 110 and step 120,
- ⇒ step 120 activates outputs 0, 1 and 2,
- ⇒ the complement of input 0 will be the transition between step 120 and 100.

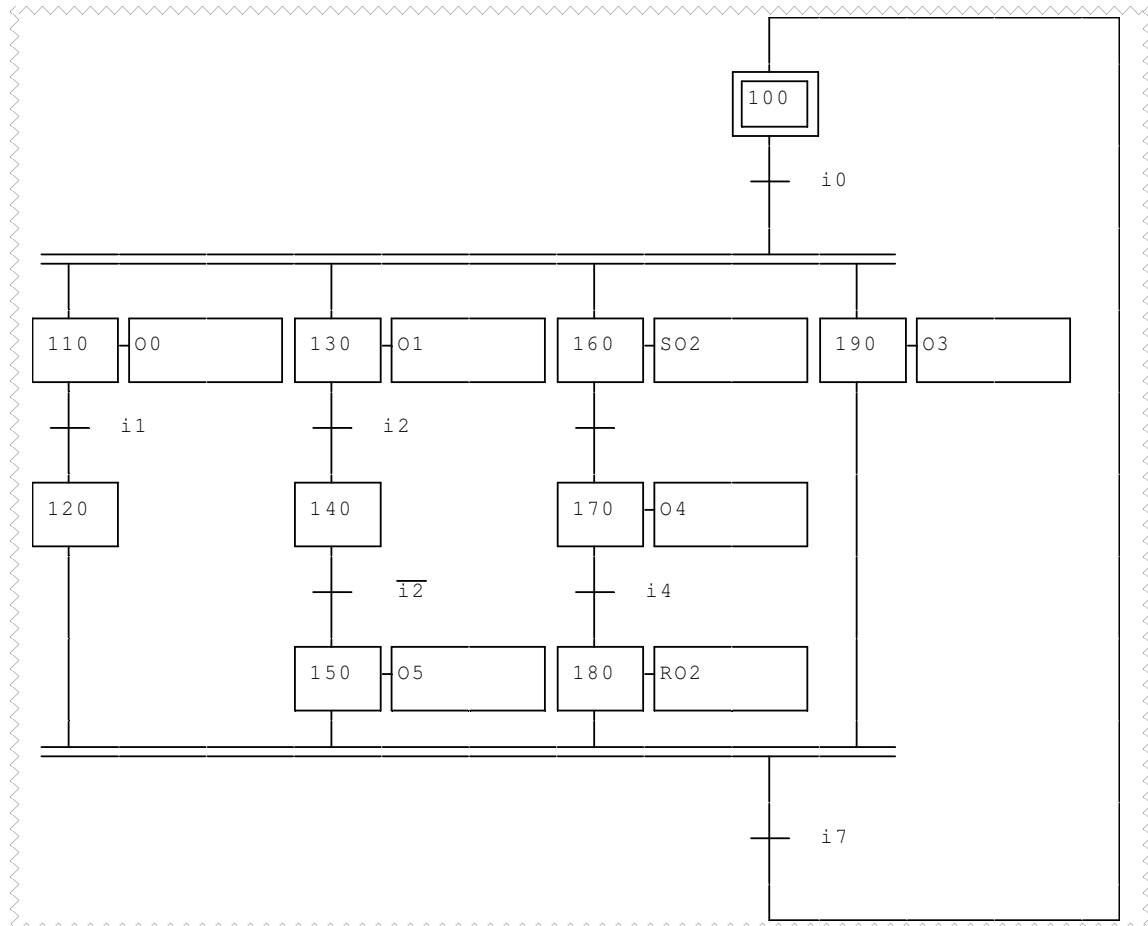
## Grafcet with an OR divergence



 Example\grafcet\sample2.agn

This example shows the use of « Or » divergences and convergences. The number of branches is not limited by the size of the sheet. It is a non-exclusive « Or » by standard. For example, if inputs 1 and 2 are active, then steps 120 and 130 will be set to one.

## Grafcet with an AND divergence

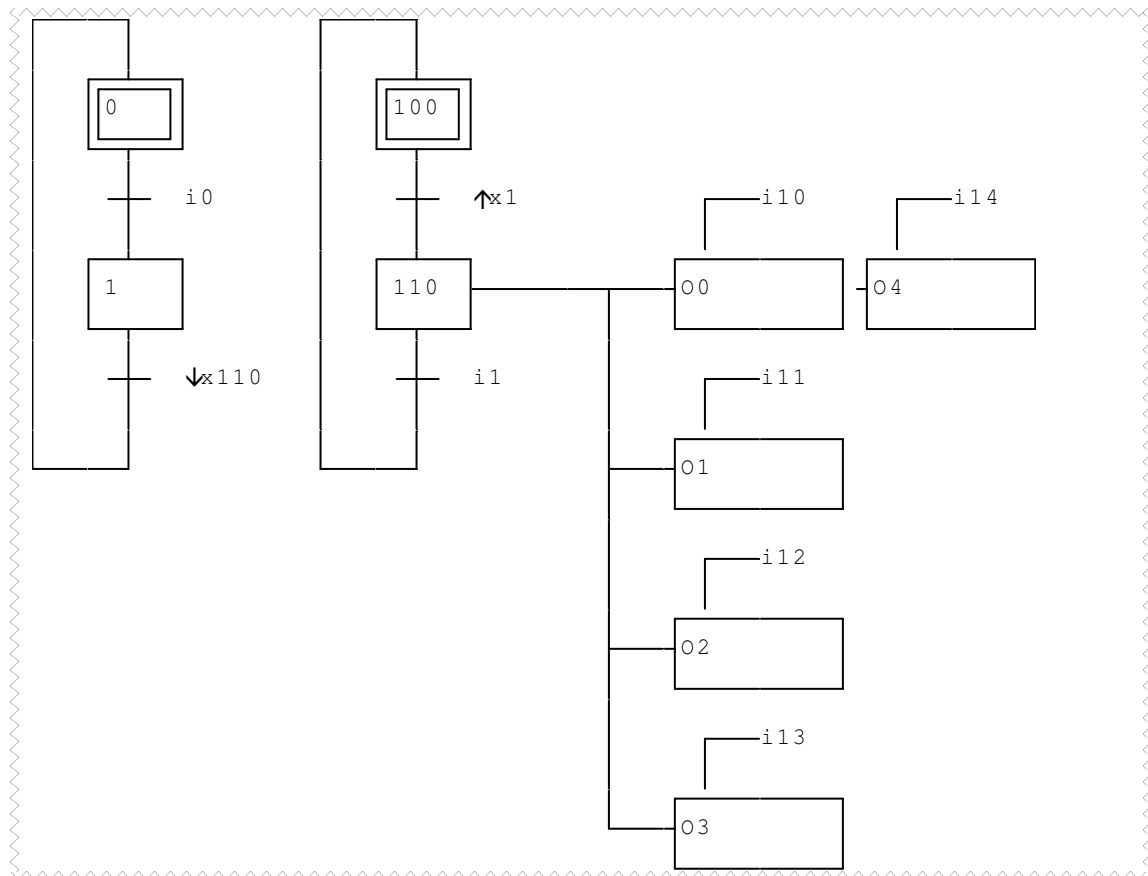


Example\grafcet\sample3.agn

This example shows the use of « And » divergences and convergences. The number of branches is not limited by the size of the sheet.. Also note the following points

- ⇒ a step may not lead to an action (case of steps 100, 120, and 140),
- ⇒ orders « S » and « R » were used with output o2 (steps 160 and 180),
- ⇒ the transition between step 160 and 170 is left blank, so it is always true, the syntax « =1 » could also have been used.

## Grafcet and synchronization

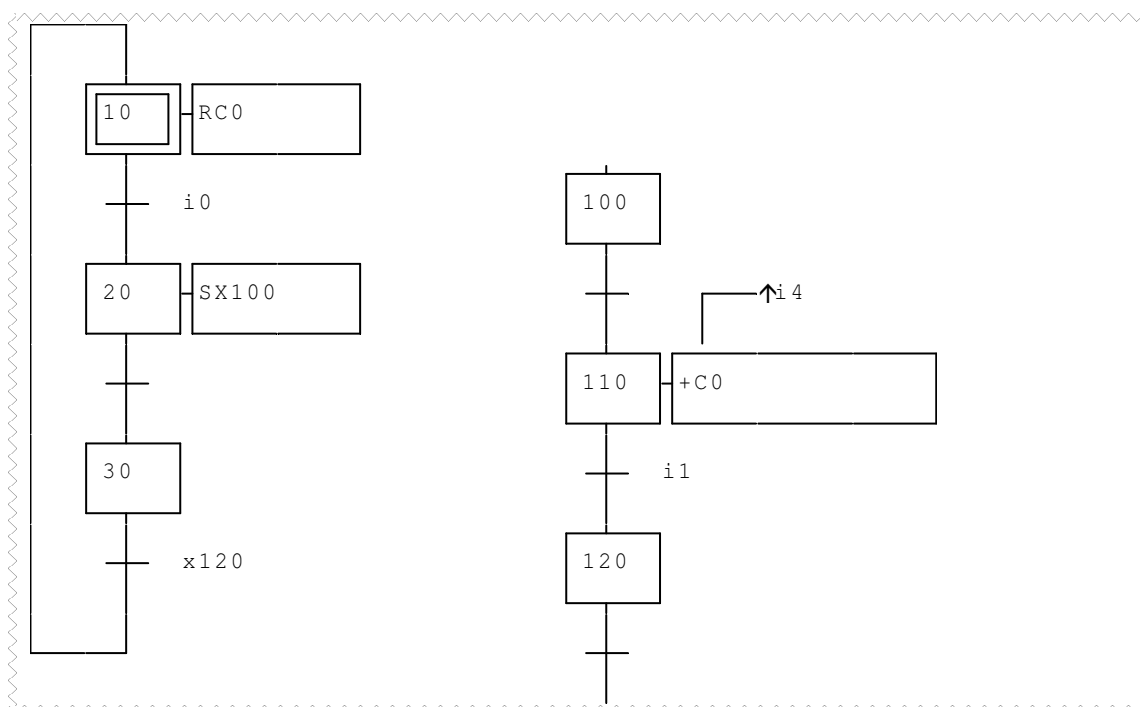


Example\grafcet\sample4.agn

This example shows the possibilities AUTOSIM offers for synchronizing multiple Grafcets. The transition between step 100 and 110 «  $\uparrow x1$  » means « wait for a rising edge on 1 ». The transition «  $\downarrow x110$  » means « wait for a falling edge on step 110 ». The step by step execution of this program shows the exact evolution of the variables and their front at each cycle. This makes it possible to understand exactly what happens during the execution. We can also see the use of multiple actions associated to step 110, which are individually conditioned here.



## Step setting



Example\grafcet\sample5.agn

In this example an order « S » (set to one) has been used to set a step. AUTOSIM also authorizes setting of a Grafcet integer (see examples 8 and 9). Again in this example, the step by step execution lets us understand the exact evolution of the program over time. We can also see:

- ⇒ use of an non-looped Grafcet (100, 110, 120),
- ⇒ use of the order « RC0 » (reset by counter 0),
- ⇒ use of the order « +C0 » (incremented by counter 0), conditioned by the rising edge of input 4, due to incrementation by the counter, so it is necessary that step 100 be active and that a rising edge is detected on input 4.

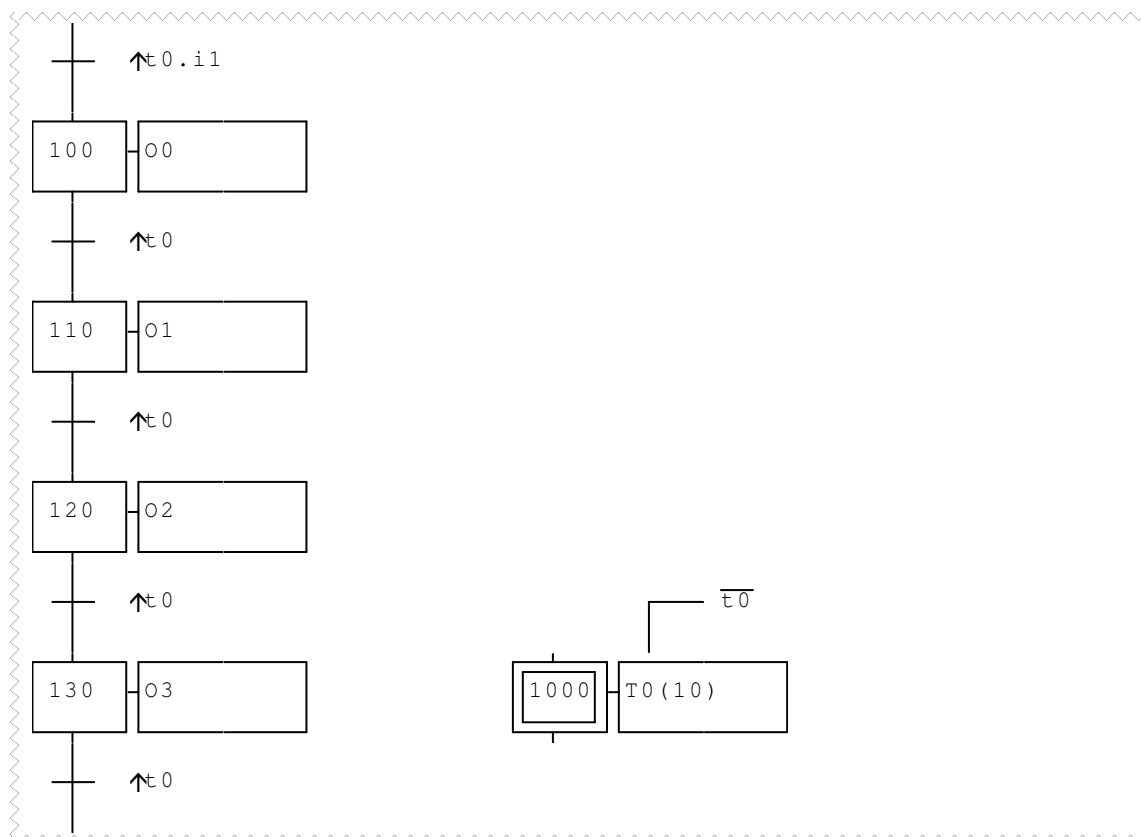
## Destination and source steps



 Example\grafcet\sample6.agn

We have already seen similar forms, where the first step is activated by another Grafcet. Here activation of step 100 is realized by the transition «  $\uparrow i0 . i1$  » (rising edge of input 0 and input 1). This example represents a shift register. «  $i1$  » is information to memorize in the register and «  $i0$  » is the clock which makes the shift progress. Example 7 is a variation which uses a time delay as a clock.

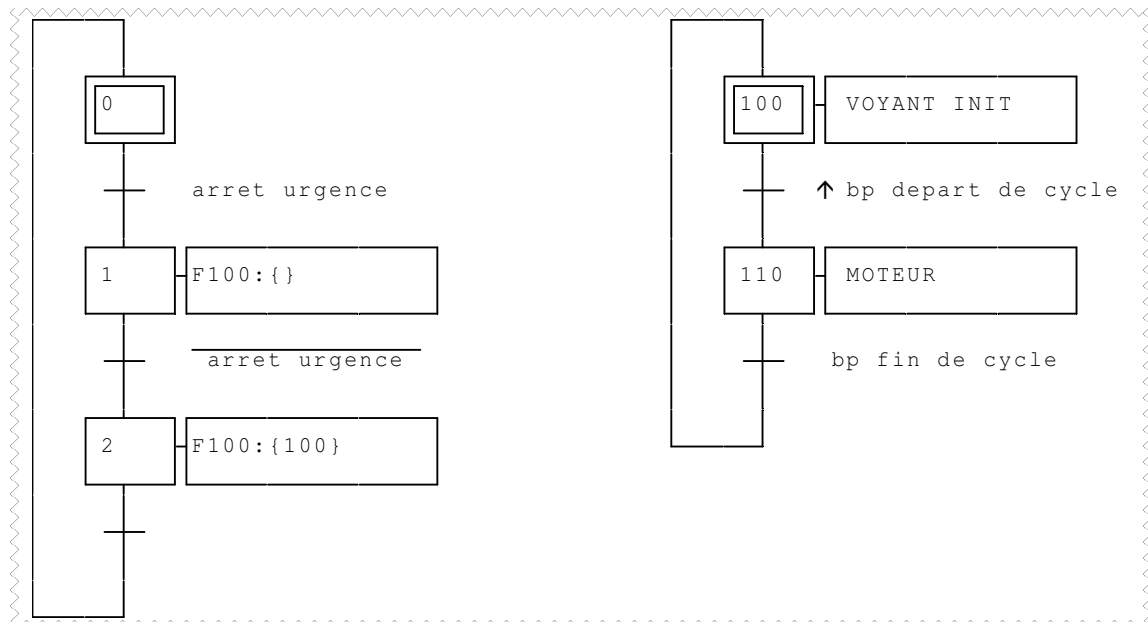
## Destination and source steps



Example\grafcet\sample7.agn

Here again is the structure of the shift register used in example 6. This time the shift information is generated by a time delay ( $t_0$ ). «  $\uparrow t_0$  » represent the rising edge of the time delay, this information is true during a cycle when the time delay has finished. Step 1000 manages the launch of the time delay. The action of this step can be summed up as: « activate the count if it is not finished, otherwise reset the time delay ». The functionality diagram of the time delays of this manual will help you to understand the functionality of this program.

## Setting Grafquets

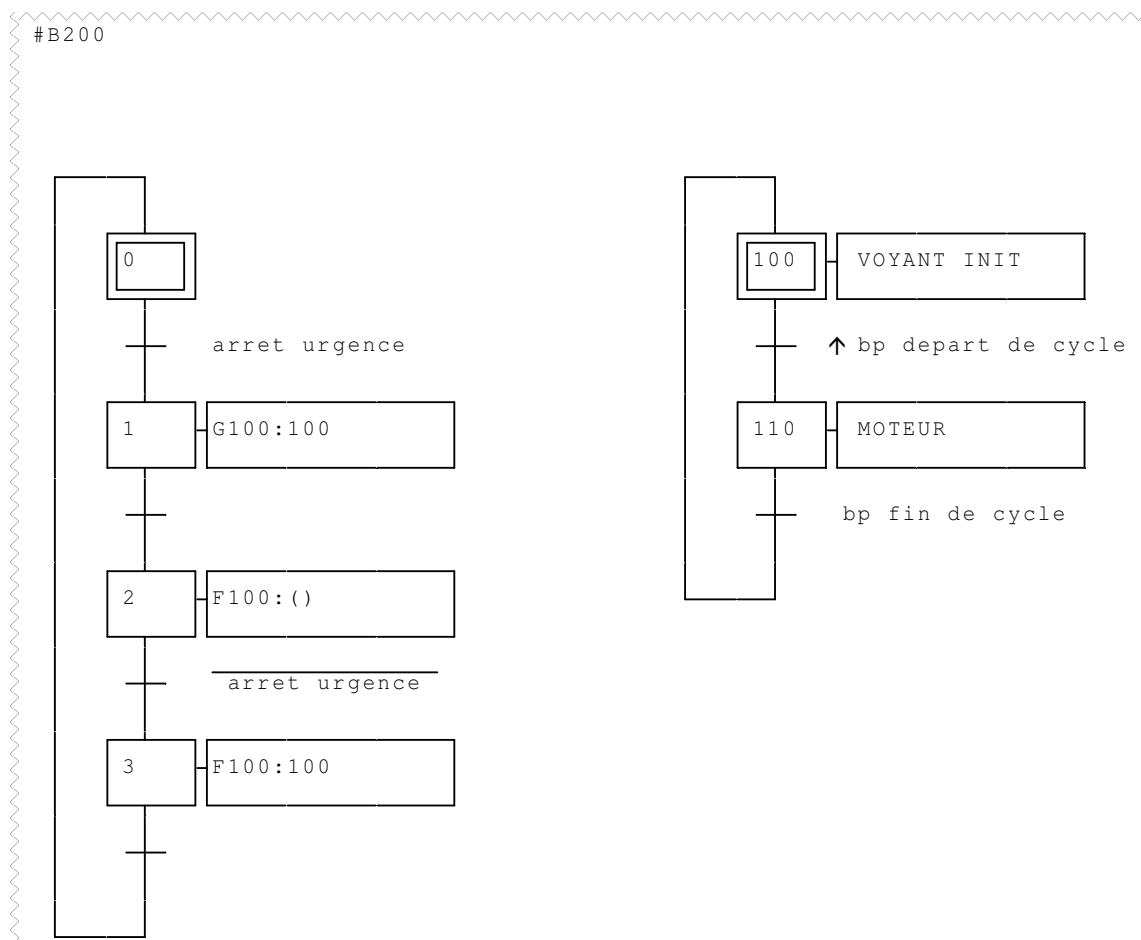


Example\grafcet\sample8.agn

This example illustrates the use of a Grafset set command. The order « F100:{} » means « reset all the Grafset steps where one of the steps bears the number 100 ». Order « F100:{100} » is identical but sets step 100 to 1. We have used symbols for this example:

arret urgence	i0	
bp depart de cycle	i1	
bp fin de cycle	i2	
VOYANT INIT	o0	
MOTEUR	o1	

## Memorizing Grafquets

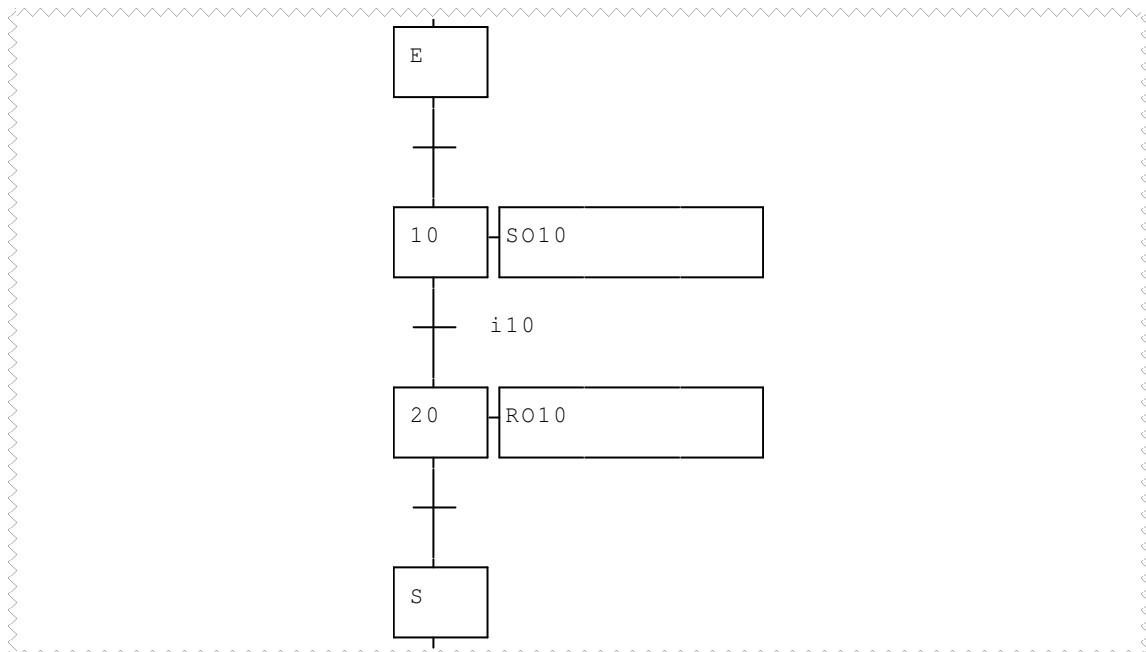
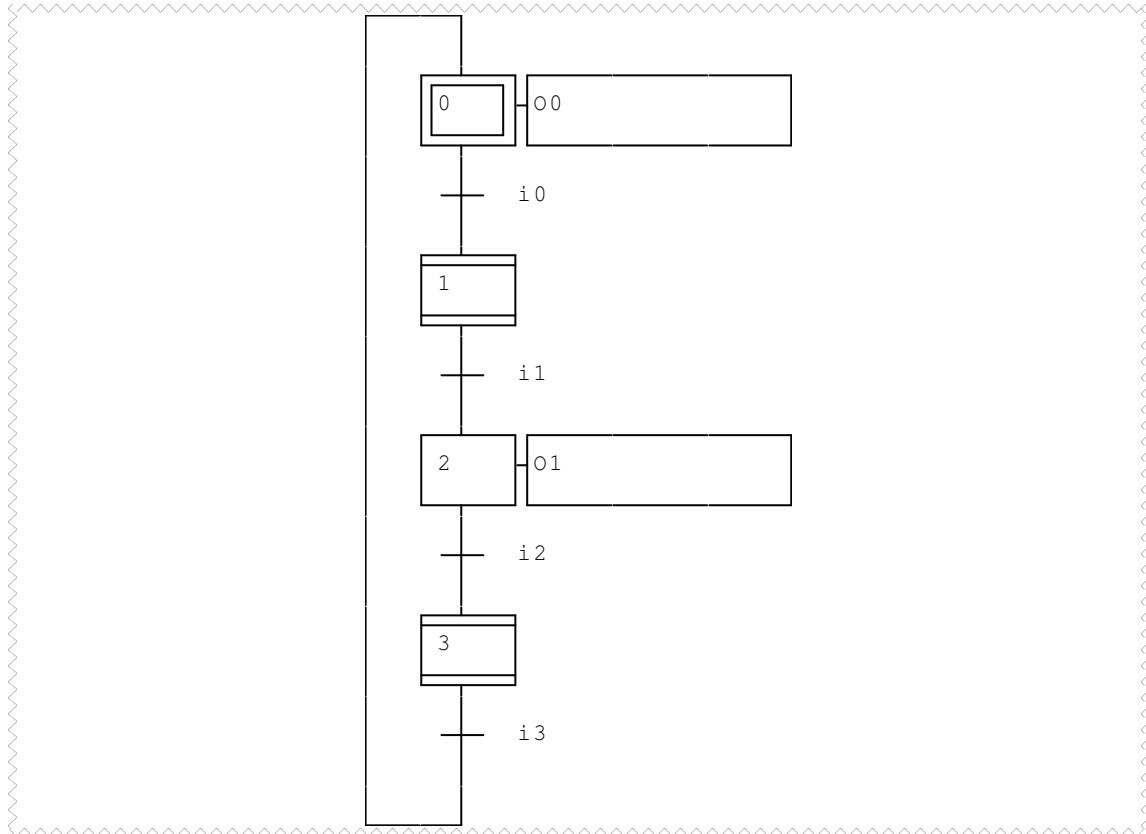


Example\grafcet\sample9.agn

arret urgence	i0	
bp depart de cycle	i1	
bp fin de cycle	i2	
VOYANT INIT	o0	
MOTEUR	o1	

This example is a variation of the previous program. The order « G100:100 » of step 1 memorizes the Grafcet production state before it is reset. When it starts again the production Grafcet will be put back in the state or the state it was in before the break, with order « F100:100 ». The Grafcet production state is memorized starting from bit 100 (this is the second parameter of orders « F » and « G » which indicates this site), command « #B200 » reserves bits u100 to u199 for this type of use. We can see that a « #B102 » command would have been sufficient here because the production Grafcet only needed two bits to be memorized (one bit per step).

## Grafcet and macro-steps

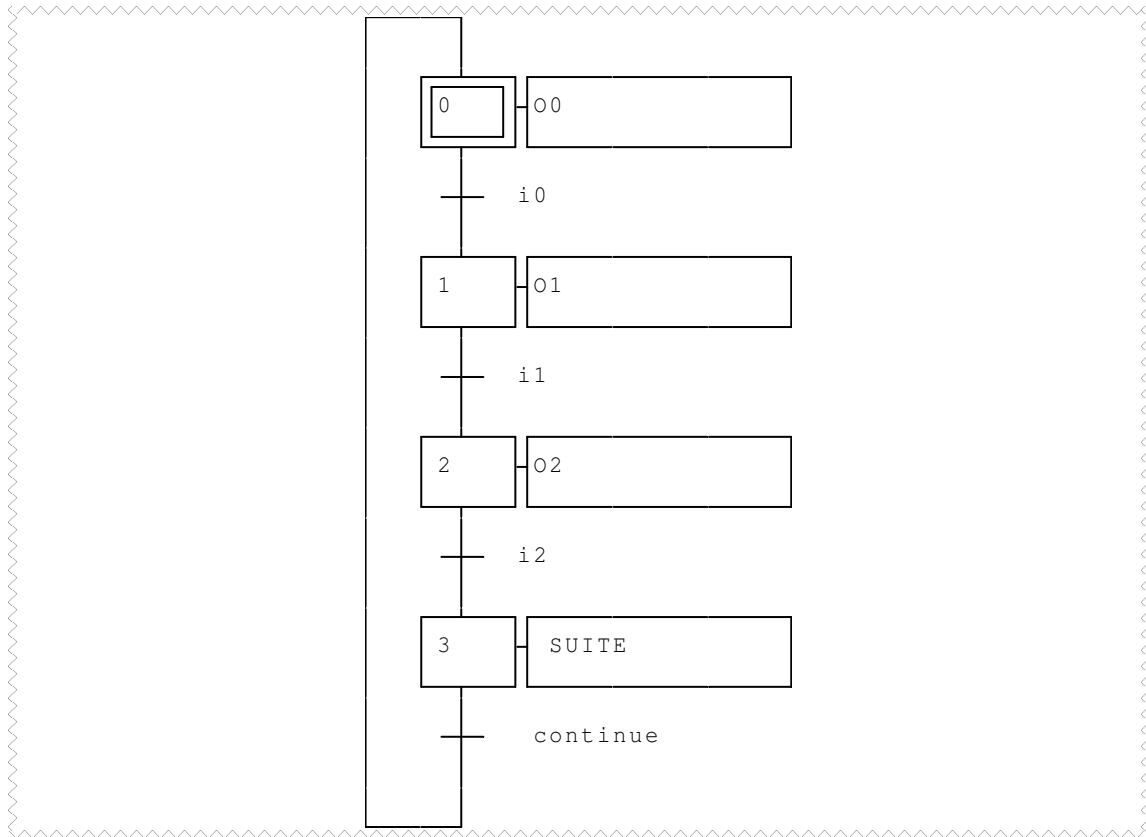


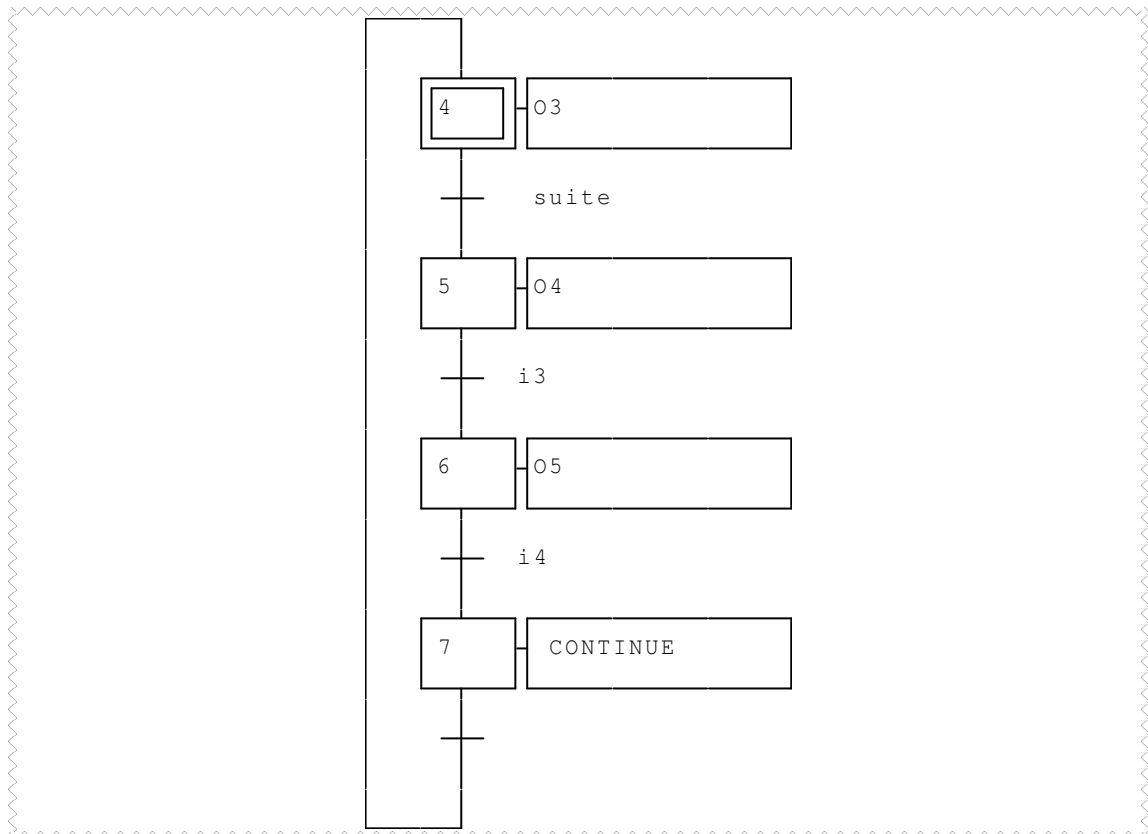
 Example\grafcet\sample11.agn

This example illustrates the use of macro-steps. The «Macro-step 1 » and « Macro-step 3 » sheets represent the expansion of macro-steps

with the input and output steps. Steps 1 and 3 of the «Main program » sheet are defined as macro-steps. Access to macro-step expansion display can be done by clicking the left side of the mouse on the macro-steps.

## Linked sheets



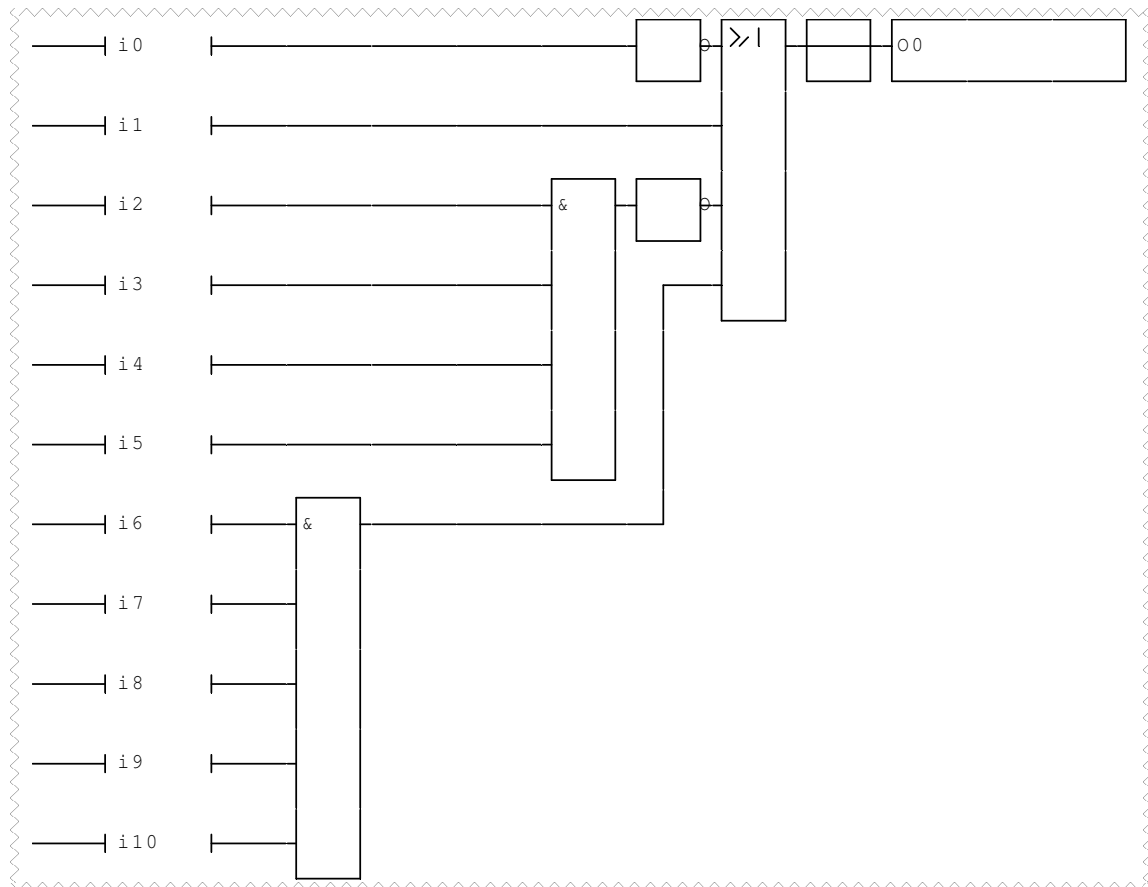


 Example\grafcet\sample12.agn

In this example two sheets have been used to write a program. The symbols « `_NEXT_` » and « `_CONTINUE_` » have been stated as bits (see the symbol file) and are used to make a link between the two Grafquets (this is another synchronization technique that can be used with AUTOSIM).



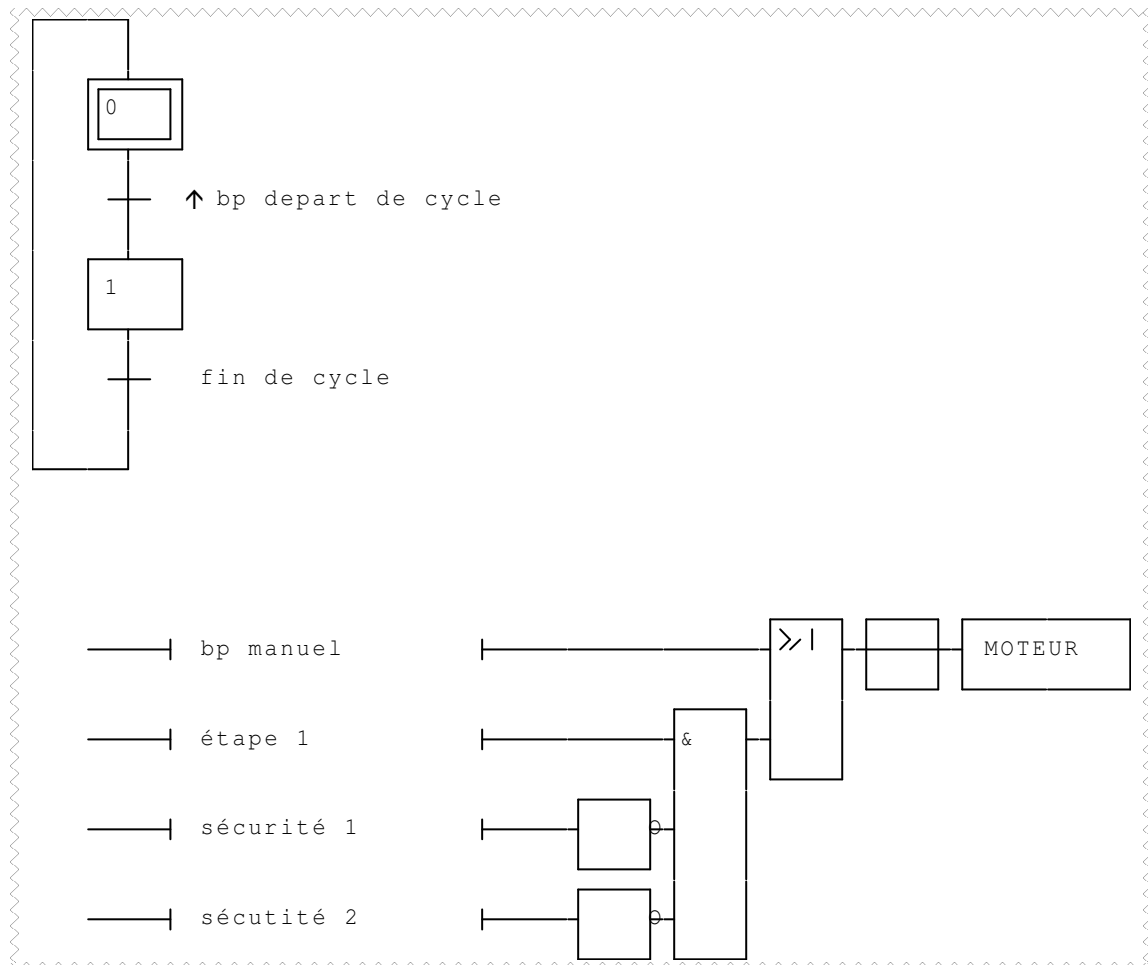
## Flow chart



 Example\logigramme\sample14.agn

The flow chart example shows the use of different blocks: the assignment block associated to key [0] to the left of the action rectangle the « no » block associated with key [1] which complements a signal and the test fixing blocks and « And » and « Or » functions.

## Grafcet and Flow Chart

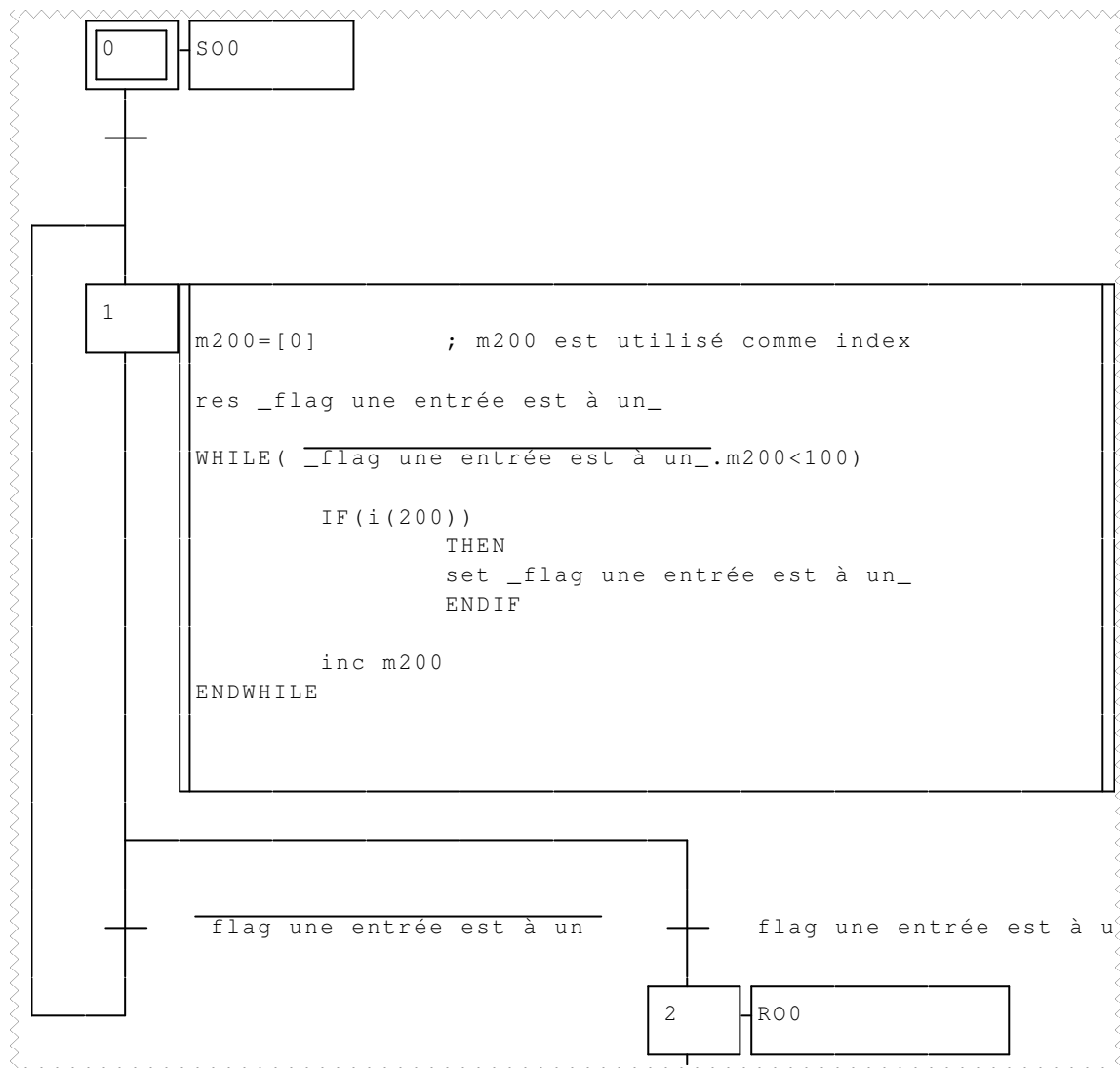


Example\logigramme\exempl15.agn

In this example a Grafcet and a Flow Chart are used together. The symbol « \_step1\_ » used in the flow chart is associated to variable « x1 ».

This type of programming clearly displays activation conditions of an output.

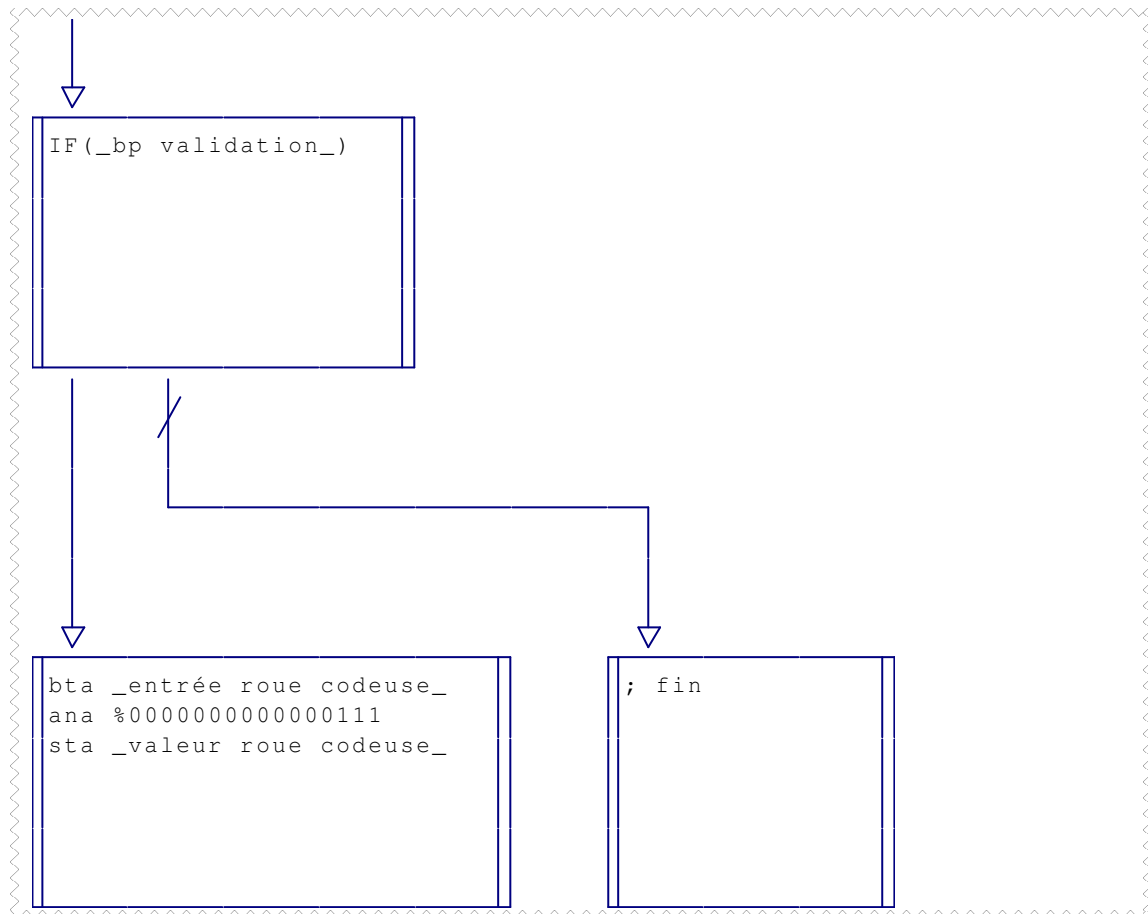
## Literal language box



 Example\lit\sample16.agn

This program which associates Grafcet and literal language box is for testing inputs i0 to i99. If one of the inputs is at one, then step 2 is active and the Grafcet is in a state where all evolution is prohibited. The symbol. « \_flag an input is at one\_ » is associated to bit u500. An indexed addressing is used to scan the 100 inputs. We can also see the simultaneous use of low level and extended literal language.

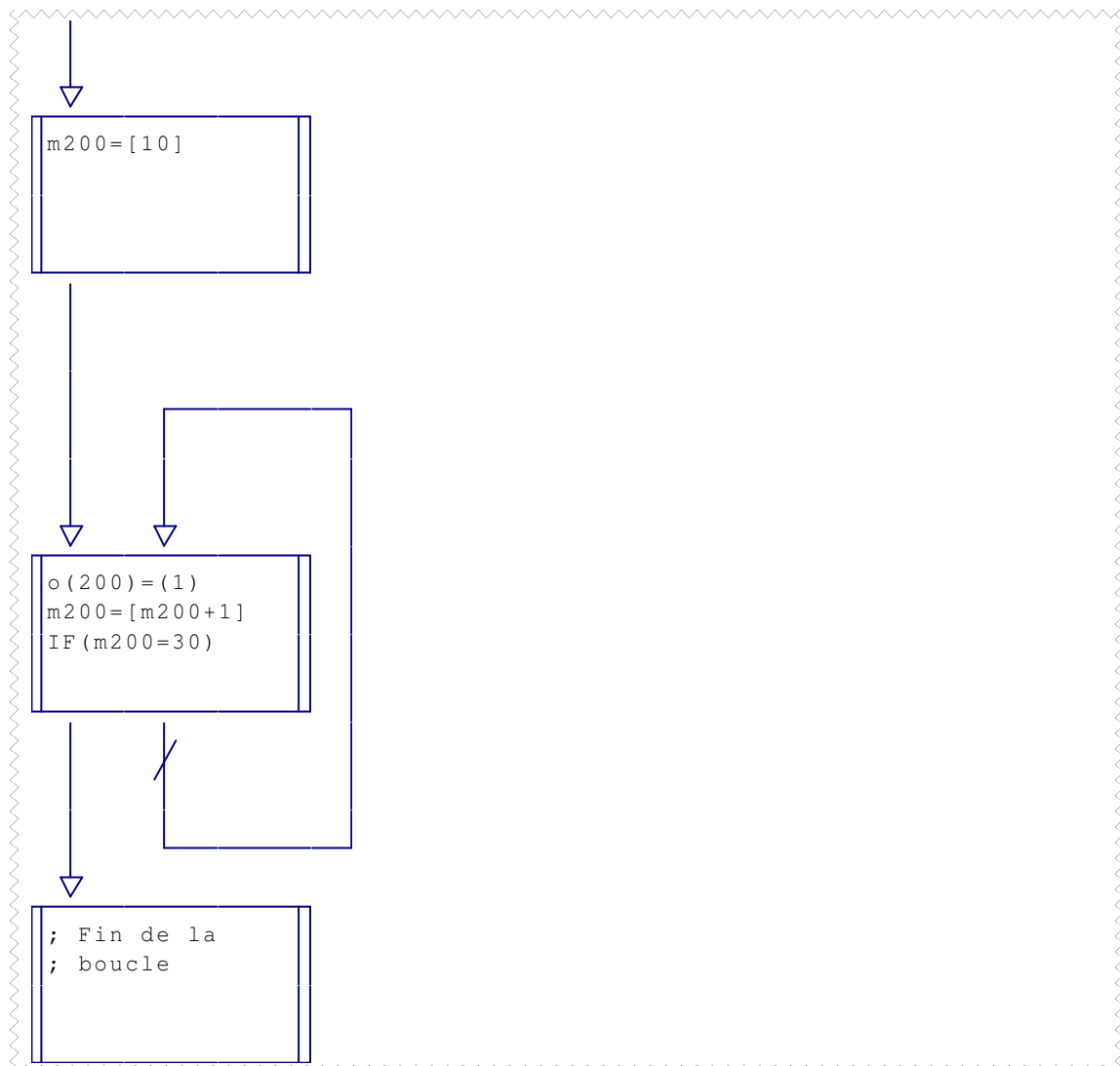
## Organizational chart



Example\organigramme\sample18.agn

This example shows the use of an organizational chart for effecting an algorithmic and numeric treatment. Here three inputs from a code wheel is read and stored in a word if a validation input is active.

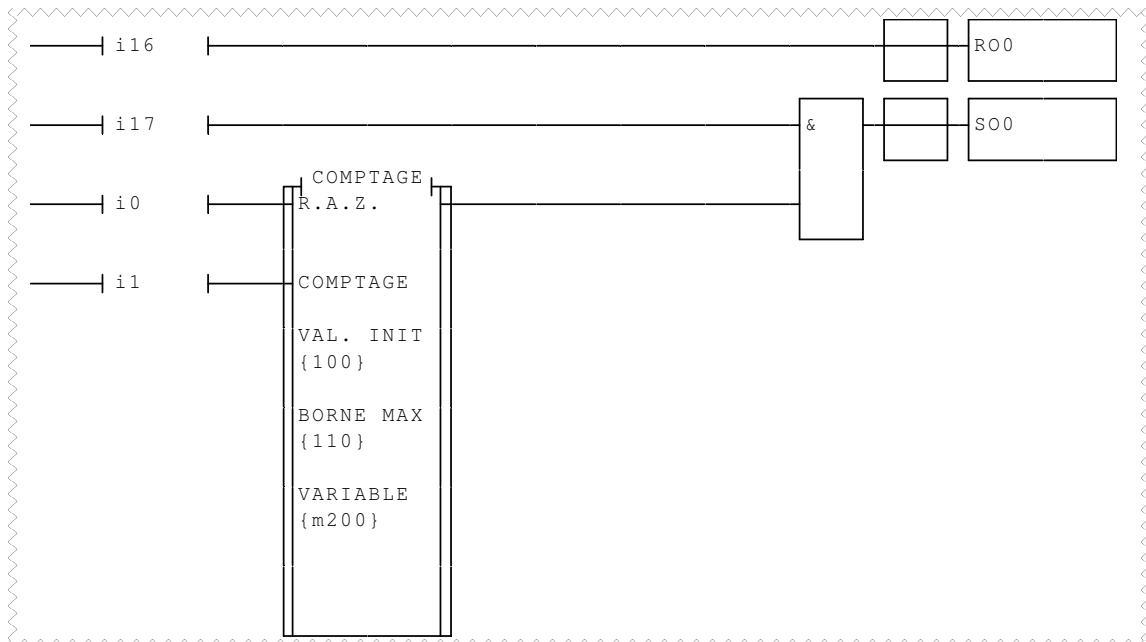
## Organizational chart




 Example\organigramme\sample19.agn

This second example of an organizational chart creates a loop structure which is used to set a series of outputs (o10 to o29) with an indirect addressing (« o(200) »).

## Function block



 Example\bf\sample20.agn

```
; Gestion de l'entrée de RAZ
IF({I0})
    THEN
        {?2}=[ {?0}]
    ENDIF

; Gestion de l'entrée de comptage
IF({I1})
    THEN
        {?2}=[ {?2}+1]
    ENDIF

; Teste la borne maxi

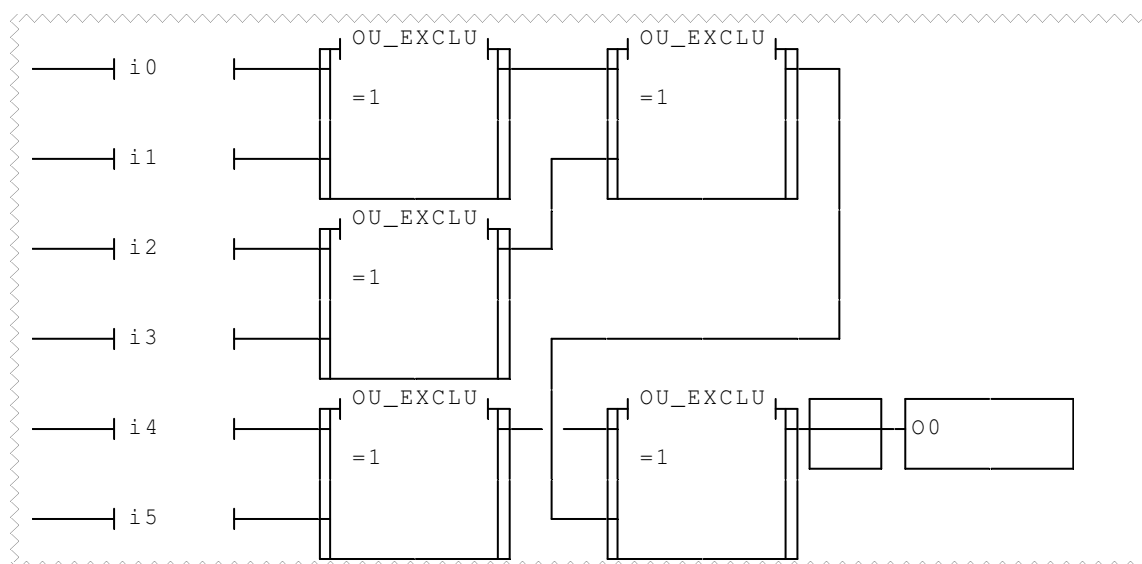
IF({?2}={?1})
    THEN
        {O0}=(1)
        {?2}=[ {?0}]
    ENDIF
ELSE
    {O0}=(0)
ENDIF
```

count lib (included in project resources)

This example illustrates the use of a function block. The functions of the « COUNT » block that we have defined here are as follows:

- ⇒ the count will start from an init value and will finish at a maximum limit value
- ⇒ while the count value waits for the maximum limit it will be set the initial value and the block output will pass to one during a program cycle.,
- ⇒ the block will have a RAZ boolean input and a count input on the rising edge.

## Function block



 Example\bf\sample21.agn

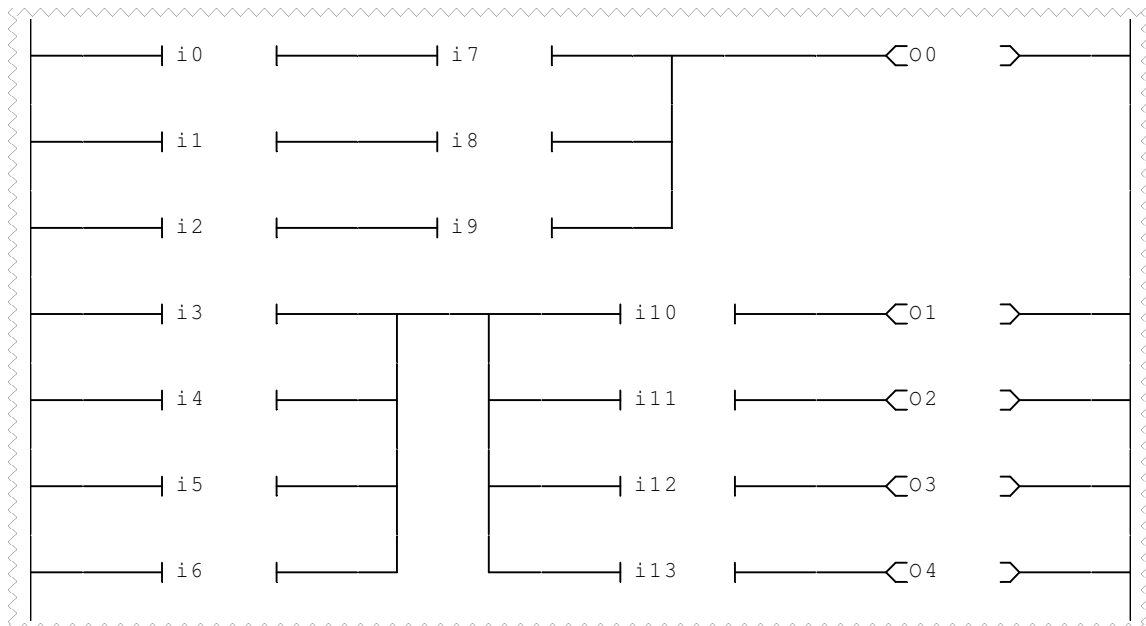
```
; Ou exclusif
neq {o0} orr /{i0} eor {i1} orr {i0} eor /{i1}
```

ou\_exclu.lib (included in the project resources)

This second example of a function block shows the multiple use of the same block. The « EXCLUSIVE\_OR » block creates an exclusive or between the two boolean inputs. This example uses 5 blocks to create an exclusive or among 6 inputs (i0 à i5). The « EXCLUSIVE\_OR.LIB » listed below supports the functionality of the block. The exclusive or boolean equation is as follows: « (i0./i1)+(i0.i1) ».

The equivalent form used here makes it possible to code the equation on a single line of low level literal language without using intermediate variables.

## Ladder



Example\laddersample22.agn

This example illustrates the use of ladder programming.