

SIOC

REFERENCE MANUAL

From the help file `sioceng.hlp` of SIOC.

Reworked by Jean-Paul Corbier and Brian Payne.

Version : 2012.03.04.

SIOC, as the rest of IOCards project, is freeware for personal use. For any other type of use, a specific authorization is required also used with commercial devices others of Opencockpits.

Author

Manuel Vélez Campos
e-mail : ferdie@arrakis.es

Testers

Luis Leal
Nico Kaan

Web : www.opencockpits.com

SIOC Simulation System

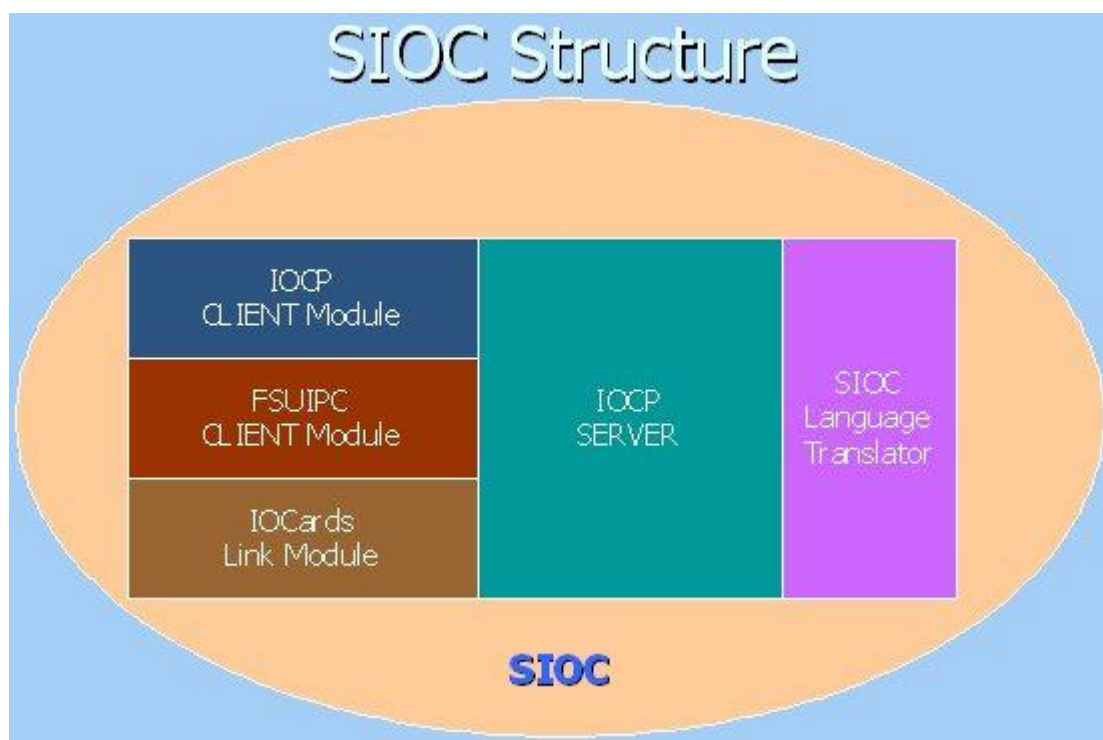
The SIOC Simulation System has been designed to cover those questions that the software of the IOCards didn't cover, assisting to requirements that arise when going beyond the electronic operation of the IOCards.

This software allows all types of simulations to be performed, that are defined by the user, affecting the electronics work, and also to the operation of the flight simulator or of the modules associated to the same one.

The System uses the IOCP protocol to send and receive the information, and to access the internal variables of the system.

The SIOC is based on:

- A IOCP server capable of giving inputs and outputs to any variable defined.
- A module IOCPclient to access to other systems (they can be the Flight Simulator using IOCPserver.dll, or X-Plane by XPLUIPC plugin).
- A FSUIPC client to access to other systems (Flight Simulator, Project Magenta, etc).
- An interconnection module to the IOCards system, to control the cards directly from the system.
- A visual scripts programmer to allows a wide operability among all these modules, with special functions, mathematical and conditional formulas, timers, etc.



The whole system is based on events.

An event can be defined as a group of operations that you perform when something happens.

In our system, the events are triggered for several reasons:

- 1) when changing the state or value of an internal variable.
- 2) when starting a timer. A timer is a kind of chronometer which starts events at a specified pace during a specified time.
- 3) when happening a state change in the incoming electronics.

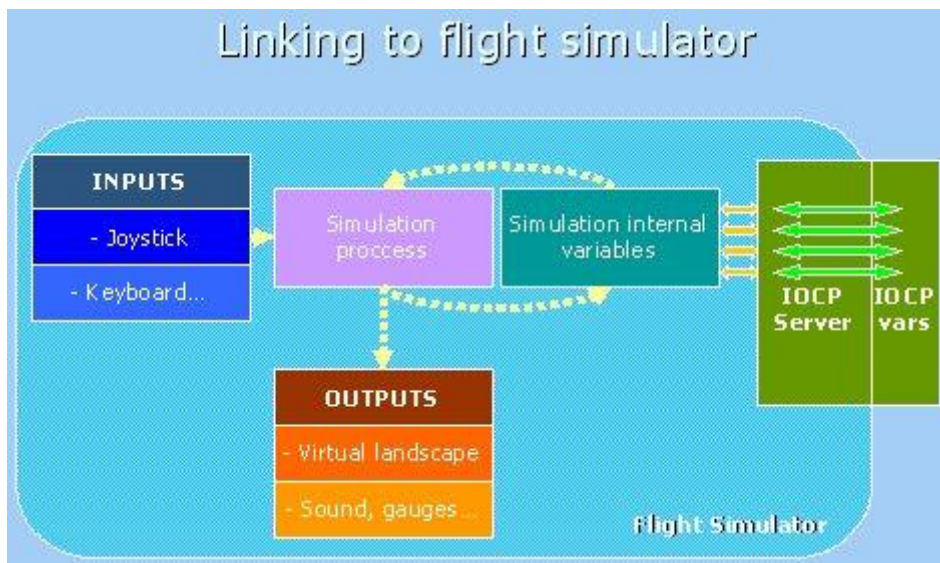
For us, everything always decreases to the same thing because everything ends in the section 1), a change in the state or value of an internal variable.

IOCP Protocol

The IOCP protocol (IOCards Protocol) was born from the need of communication of the cards with different software in different machines, being used for it the net protocol TCP/IP, being possible the interconnection so much in environments of local nets as through internet.

This protocol presents notables advances compared to other protocols used for the interconnection of modules or electronic cards to Flight Simulator:

- This protocol is quicker.
- It consumes less resources.
- It is based on events.
- It doesn't need applications type WideFS to be interconnected with other computers.
- It is multi-platform, being able to connect to Fsimulator or to X-Plane.
- It is totally freeware for non commercial environments.



Modules of the SIOC. Links

You have several basic *modules* to link your variables:

- the FSUIPC *module*,
- the IOCP *module*,
- the access *module* to the IOCards
- and another to external *modules*.

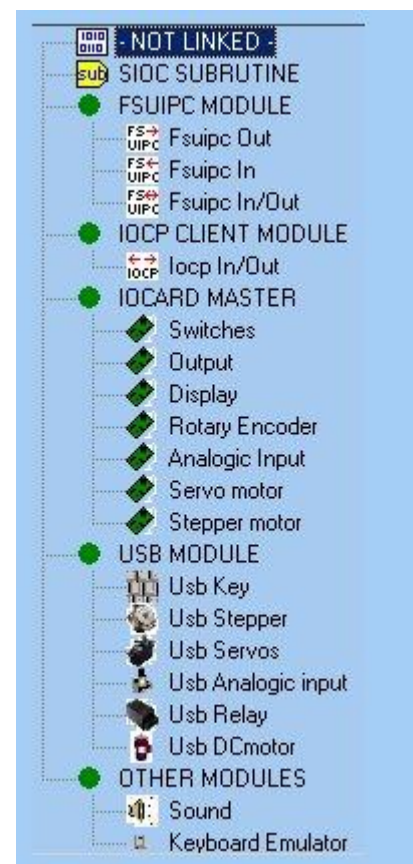
Also of course, you cannot link your variable to any *module*, for you will select it this possibility in the corresponding option of the menu. With what this variable will work with the server without the alteration of the same one affects to any *module*.

The formula used for the interaction with the modules is always the same:

- the variable takes the value resulting from an event or situation which was initiated by the module itself (with which it was already connected) and generates a script with which it is associated.
- on the other hand, in changing the value of a variable, the event or situation is initiated and thus sends to the module the information which it will modify one way or the other as will be explained further on.

Therefore, we will make a difference between different types of unions: those that are of “**send**” or “**outputs**”, those that are of “**reception**” or “**inputs**”, and those that are “**bi-directional**” or of “**inputs and outputs**”.

Here are the relationship between *modules* and available links for the SIOC:



Variables IOCP

The SIOC programming is based on the definition of its variables and of its associated scripts (events).

Each variable is distinguished by its number, which can be from 0 to 9999. In such a way that any client that connects with the SIOC server, and require information of the variable #0134, makes reference to this variable with this number. Any modification of the value will be notified to the client. Also, a request from the client to modify a variable, will launch the corresponding event and will execute the associated script.

Besides their identification number, the variables can have a description and an initial value (optional). This value will be the first one assigned to the variable when beginning the program.

To be able to access the rest of modules that the SIOC negotiates, the variables have a special characteristic: they can LINKS TO or TOGETHER to these modules in such a way that the modification of this variable will start an interaction with the module to which it is linked, or on the contrary, an action of one of the modules will start the modification and therefore, the launching of the event of the variable to which it is associate.

Therefore controlling the modules of the SIOC is done through the variables that we link to the module.

As soon as we define some type of link, the program requests the introduction of the data from the module to which we want to link. In the case of this example we can observe like when linking the variable to the module of switches of the IOCards, automatically it is asked to introduce the inputs number and the type of the switch.

The variables always store a whole value in the range -2147483648 at 2147483647. This allows to always have a conversion to store decimal values.

For example, if we have to keep a value with 4 decimals, let's say 1.0345, we always store the value multiplied by 10.000, that is 10345. When the real value is needed, we divide the value stored by 10.000 in the corresponding operations. This is the own specification of the IOCP protocol.

Local internal variables

When running a script, several local variables can be used. They are defined as *local* because their value only exists for the script in execution with the target of storing temporary values.

We have two types of variables, the real ones and the booleans:

The real variables store a real value, that is to say, decimals or whole, positive or negative in the range 5.0×10^{-324} to 1.7×10^{308} .

It exists 3 real variables: **L0**, **L1** and **L2**.

The booleans variables store the value of a condition that can be false or true. They are used fundamentally to form conditions since starting from simple conditions in the conditional commands a condition it is only allowed.

It exists three booleans variable : **C0**, **C1** and **C2**.

For example, to define a condition as

`"If L0>5 and L0 <10",`

we would make a first assignment as

`"C0 = L0>5, C1 = L0 <10, C0 = C0 AND C1".`

In such a way that **C0** will be false or true according to the previous conditions.

By default, when running a script, the real variables take the value 0, and the booleans take the value false.

Programs of the SIOC suite

The SIOC basically has two programs and two tools:

config_sioc : this program is used to edit and generate the SIOC scripts by using an advanced interface. You can use as parameter the `.ssi` file that you want to open.

```
config_sioc <compiled.ssi>
```

sioc : this is the program of the SIOC management. The program reads the script SIOC compiled in `.ssi` or `.ssc` file and executes it, negotiating with the IOCP server as well as with the different communication modules, FSUIPC, IOCP and the administration of the IOCards.

```
sioc <configuration.ini>
```

sioc_compiler : this program externally compiles an SIOC script (in a `.txt` file). The parameters to use are:

```
sioc_compiler <text_file> <compiled.ssi> <log>
```

IOCP_Console : this tool is fundamental to be able to purify and to know exactly that data are negotiated in the IOCP server. You can use as parameters the IP address and port for the initial connection.

```
IOCP_Console <IP address or name of the server> <port>
```


Files

New : prepares a new blank script where to insert variables and command lines.

Open : opens a `.ssi` file.

Save : saves the present file in `.ssi` format.

Save as : same as **Save**, but we can change the name and the folder of the `.ssi` file.

Export to TXT : generates a `.txt` file (in SIOC script language) that can be imported later.

Export to ASCII : generates a `.txt` file (ASCII) that can be sent by email or published on forums.

Export Codified : generates a `.ssc` that is 128 bits coded and can be only read by SIOC.

Import TXT : opens a `.txt` file (SIOC), generating the corresponding script, or showing any error found.

Import ASCII : opens a `.txt` file (ASCII).

Exit : quits the program.

Edit

Insert Variable : By this option of the menu or with pressing the right button of the mouse, we can insert a space dedicated to the definition of a variable just before the line selected at the moment. Once inserted this space, we can define the variable making double click with the mouse.

New Variable : By this option of the menu or with pressing the right button of the mouse, we can create a space dedicated to the definition of a variable just at the end of you line them existent. Once created this space, we can define the variable making double click with the mouse.

Insert Command : By this option of the menu or with pressing the right button of the mouse, we can insert a space to the definition of a command line before the line selected at the moment. Once inserted this space, we can define the corresponding line making double click with the mouse.

It is important to stand out that we should have selected a variable where the command line, or another line of command of the same level should insert.

If the selected line is an IF or ELSE sentence, the new line will be created in a level of having penetrated bigger corresponding to that IF or ELSE.

New Command : By this option of the menu or with pressing the right button of the mouse, we can insert a space to the definition of a command line at the end of you line them existent of that level

Once created this space, we can define the corresponding line making double click with the mouse.

It is important to stand out that we should have selected a variable where the command line, or another line of command of the same level should create.

If the selected line is an IF or ELSE sentence, the new line will be created in a level of having penetrated bigger corresponding to that IF or ELSE.

Delete : With this option will eliminate a variable definition or command line together with the whole rest of lines that you hang of that level.

Copy : This option allows to copy a variable definition together with all the command lines that hang of her.

Paste : once one has copied a variable definition, this option allows to insert a copy of this definition with the exception that the name of the variable will be omitted, and the number will be generated being located the first one automatically free in the script.

See

Expand : Expands all the script levels, showing all the lines.

Hide : Hides every command lines, showing only the variables definition.

By clicking on the + signs, we can expand and collect any part of the script.

Language

Spanish : Changes the interface language into Spanish.

English : Changes the interface language into English.

Compiler

Files : Defines a .txt input file and a .ssi output file to help with the continuous edition and compilation SIOC language process.

Run : This option run the compilation of the files defined previously. There is a button named COMPILER that does the same action.

Any compilation error is shown on the screen.

Group

With this option, the user can handle a group of .txt scripts. After compilation, only one .ssi file is generated.

All command lines with variables number 0000 will be grouped in a single 0000 variable. The rest of variables will be automatically renumbered to avoid collisions.

We should be careful not to have same variable names in different files of the grouping, because this will produce an error in the compilation.

Files : .txt files will be defined in each line of this window. These files must be located in the SIOC folder.

Run : runs the grouped compilation of every file previously defined.

Utils

Ficheros 2.* a versión 3.* : This option is used to convert .ssi files from version SIOC 2.* into version 3.*

Help

Help : With this option, the user can access to this help (in Spanish or English, as selected).

About SIOC : Information about the present Config_Sioc version.

SIOC.INI

In this file you can define all parameters for SIOC program. You can modify those parameters :

General parameters of SIOC :

Name : name of SIOC server.

IOCP_port : IOCP server port.

IOCP_timeout : max. response time for IOCP packets.

Minimized : when the parameters is Yes, SIOC is launched in background. A control icon appear in the tray.

toggle_delay : delay time needed for toggle operations (used in many variables of Project Magenta software).

CONFIG_FILE : script file used by SIOC,

The file can have the follow extensions :

.ssi Used in a compiled file ready for run.

.ssc Used in a compiled file, but this is codified.

.txt When the file has not been compiled. In this case, SIOC compile the script and run. If a problem happens, the SIOC sends an error message.

.lst The file is a list of sources .TXT (one by line). The SIOC program launches the compiler and merges all files in one. SIOC runs the result or sends an error message if a problem happens.

MONITOR module parameters:

Monitor_disable : when the parameter is Yes, SIOC disconnect the **Monitor Module** .

Monitor_time : delay for send the monitor broadcast signal.

IOCARDS module parameters:

IOCard_disable : when the parameter is Yes, SIOC disconnect the **IOCards Module** .

Divisor_AD : frequency divisor for delay the analogic axes. (1-999)

You can define the cards used using the parameters :

MASTER=(device index),(Type),(Number of cards),(device number)

Device Index : index used in SIOC variable like device, for indicate a card used.

By default, you need not to use parameter DEVICE in SIOC script if you use the index 0.

Type: to define the type of device used. Choose :

0 : *Master Card Emulator. // Obsolete*

1 : *Master Card connected directly to parallel port.// Obsolete*

2 : *Master Card connected through compatible cable.// Obsolete*

3 : Parallel port expansion card.// Obsolete

4 : USBExpansion Card.

5 : MCP module

6 : USBOutputs Card used

7 : EFIS module used

8 : COM radio module Card used

9 : NAV radio module used

10 : ADF radio module used

11 : ATC radio module used

12 : RMP airbus radio module used

Number of cards = Number of Master Cards used, 1 to 4 for Expansion Card (Lpt or USB), 1 for Master card or Emulator.

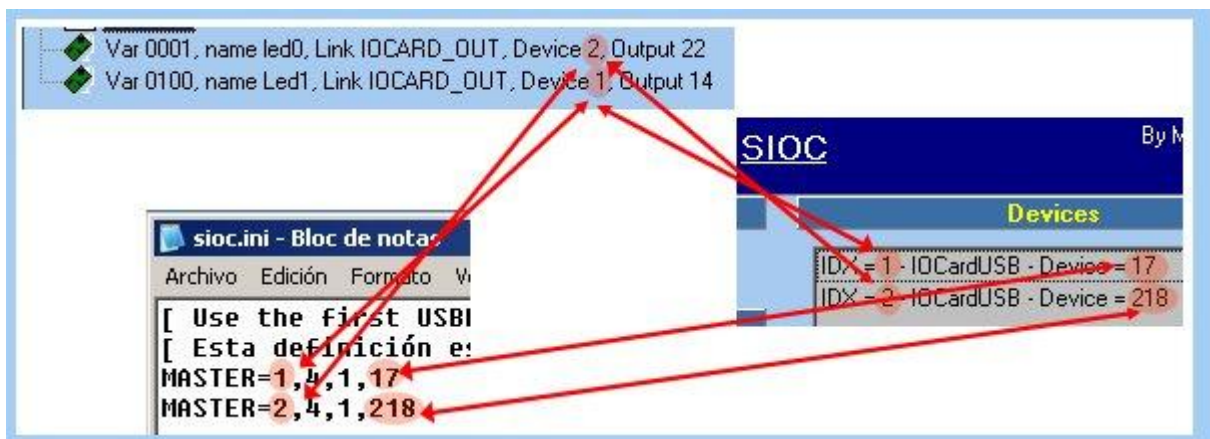
Device number = 0 for Emulator or for use first card detected. For LPT port, the port address (for example \$0378), for USB the device number assigned.

Example of 2 USBExpansion connected with 3 & 2 Master Cards :

MASTER=0, 4, 3, 22

MASTER=1, 4, 2, 24

This is a diagram to explain how you should use the parameter DEVICE in your script, and how you should configure SIOC . INI.



For other cards, you should use :

Card_name=(Device Index),(Device Number)

USBStepper=(Device Index),(Device Number)

For USBStepper Cards

USBKeys=(Device Index),(Device Number)

For USBKeys Cards

USBServos=(Device Index),(Device Number)

For USBServos Cards

USBRelays=(Device Index),(Device Number)

For USBRelays Cards

USBDCmotor=(Device Index),(Device Number)

For USBDCmotor Cards

Example of 2 USBServos using index 0 & 1, and device numbers 17 & 23

USBServos=0,17

USBServos=1,23

For Analogic axis you should use the physic cards allowed. If you use a axis of the USBServos card with device number 23, the device number for USBAnalogic should be 23.

USBAnalogic=(Device Index),(Device Number)

The index are independent for each type of device.

FSUIPC module parameters:

FSUipcdisable : when the parameter is `Yes`, the FSUIPC module is disconnected.

FSUipcRefresh : this is the time refresh for input/output pollings (in milliseconds). You can reduce this number for many access to FSUIPC but the computer can have more load and the result is bad.

IOCP clients modules parameters:

Sioc have two client modules for IOCP protocol. These modules can be connected to two different servers in two different computers. You can define the client used using `Device` parameter in SIOC scripts.

IOCPini_delay : this is the time in milliseconds for initiate a connection to server from clients when SIOC has been initialized.

This delay is used for give time to SIOC for initialize all variables.

IOCPclient0_disable : when the parameter is `Yes`, the SIOC disable the IOCP client module #0.

IOCPclient1_disable : when the parameter is `Yes`, the SIOC disable the IOCP client module #1.

IOCPclient0_host : defines the server host address when the client module #0 should be connected.

IOCPclient1_host : defines the server host address when the client module #1 should be connected.

IOCPclient0_port : define the server port when the client module #0 should be connected.

IOCPclient1_port : define the server port when the client module #1 should be connected.

Parameters of the SOUND module

SIOC have a sound module for simulate different sounds in same time when you can control Volume, Frequency and pan.

Sound_disable : when the parameter is Yes, The SIOC disable this module.

Volume : define the general value of Volume of all sounds used [0-100].

To define all sounds used, you can use the follow parameters :

SOUND=(Wav file),(Frequency),(Volume),(Pan)

Wav file = This file should be in the same directory of your SIOC. If you want to play this sound in continuous loop you should put * before file name.

Frequency = Frequency value (100 to 100000 0=original, -1=default)

Volume = Volume value (0 to 100 -1=default volume)

Pan = Pan value (-100 (Left) to +100 (Right), 0=center, -1=default)

You can define all sounds needed including a **SOUND** definition with all parameters. The index used with SIOC scripts is the order of sound definition, the first definition entry is index #1...

For example :

```
Sound=*outermk.wav,-1,-1,-1
```

EMULATION KEYS module parameters :

The SIOC have an emulation keys module to launch key commands to active windows. If you want to send commands to a specific window, you have to make sure that this window is active.

WINDOW=(Windows name)

To define each command keys to each index used in SIOC script you can use :

#(Index number used by SIOC script)=Key command

For example :

```
#1=\B\A
```

You can define special Key commands using \ (backslash) followed by these letters :

```
A = BKSP = #8;  
B = TAB = #9;  
C = ENTER = #13;  
D = ESC = #27;  
E = F1 = #228;  
F = F2 = #229;  
G = F3 = #230;  
H = F4 = #231;  
I = F5 = #232;  
J = F6 = #233;  
K = F7 = #234;  
L = F8 = #235;  
M = F9 = #236;  
N = F10 = #237;  
O = F11 = #238;  
P = F12 = #239;  
Q = HOME = #240;  
R = END = #241;  
S = UP = #242;  
T = DOWN = #243;  
U = LEFT = #244;  
V = RIGHT = #245;  
W = PGUP = #246;  
X = PGDN = #247;  
Y = INS = #248;  
Z = DEL = #249;  
1 = SHIFT_DN = #250;  
2 = SHIFT_UP = #251;  
3 = CTRL_DN = #252;  
4 = CTRL_UP = #253;  
5 = ALT_DN = #254;  
6 = ALT_UP = #255;  
\ = \
```

Also, you can define special Key commands using / (slash) followed by these letters :

```
'/' = '/'  
'0' = NUM_0  
'1' = NUM_1  
'2' = NUM_2
```

'3' = NUM_3
'4' = NUM_4
'5' = NUM_5
'6' = NUM_6
'7' = NUM_7
'8' = NUM_8
'9' = NUM_9
'A' = MULTIPLY
'B' = ADD
'C' = SEPARATOR
'D' = SUBSTRACT
'E' = DECIMAL
'F' = DIVIDE
'G' = SPACE

For example, the sequence SHIFT+S can be defined as : \1S\2 .

All keys use uppercase. If you need use lowercase can put the sign < before.

SIOC.EXE

SIOC is the program to run SIOC scripts, launch a IOCP server and connect to others modules.

The config file is `sio.c.ini` and you can execute some operations using these buttons :

EMPTY SCRIPT

Launch a RELOAD and use an empty file for run a IOCPServer.

RELOAD

Initialize the system and reload the script file defined in `sio.c.ini`.

CONFIG

Launch the program `config_sio.c.exe` with the script defined in `sio.c.ini`.

IOCP_CONSOLE

Launch the program `IOCPconsole.exe` using the name definition of script defined in `sio.c.ini`.

TRAY

The SIOC program is executed in background, and a control icon is shown in the tray.

EXIT

Disconnect all modules and exit of program.

General characteristics

Language identifications, variables, constants and the rest of elements, are written between spaces, colons, brackets or tabs.

Different levels are defined by { and }.

// can be used at any point to insert general or particular comments.

Only one definition or command can be written in one line.

There is no distinction between lower and upper case.

We can made reference to any SIOC variable by writting V with its number (for example V0001), or by writing & with its name (for example &course).

We can use internal variables using the names **L0**, **L1**, **L2**, **C0**, **C1** and **C2** (if the specific command and/or type allows this).

Variable definition

Var number **Attribute** parameter **Attribute** parameter (...)

```
{  
    Command parameters  
    Command parameters  
    Command parameters  
    ...  
}
```

number = Integer from 0 to 9999

parameter = information regarding attribute or command.

Examples:

```
Var 0001, Link FSUIPC_IN, Iffset $07F2, Length 2
```

```
Var 1002, Link IOCARD_ENCODER, Input 0, Aceleration 8, Type 1
```

```
Var 0086, Link IOCARD_DISPLAY, Digit 0, Numbers 5
```

```
Var 9023
```

Attributes

Valid attributes:

Link Type of link for the variable.

Type Defines special characteristics for the element.

Offset IOCP variable number or FSUIPC offset.

Name Symbolic name assigned to a variable.

Value Initial value for a variable.

Length FSUIPC offset length.

Input Input.

Output Output.

Digit First display in a group of displays.

Acceleration Defined acceleration for an encoder.

PosL Left position for calibration.

PosC Center position for calibration.

PosR Right position for calibration.

Device Index number for the peripheral referenced by the variable link.

Without link assigned

Defines the variable, so it's ready to be used.

Attributes :

Name Assigns a symbolic name to a variable.

Value Sets an initial value.

Example:

```
Var 0001  
Var 0006, Name Heading, Value 180
```

FSUIPC IN

The variable will store the value from FSUIPC offset. The associated script will be executed only if the variable value changes.

Attributes :

Name Assigns a symbolic name to a variable.

Offset FSUIPC offset.

Length Variable size in bytes. Values can be 1, 2, 4 and 8.

Numbers If we work with 8 bytes FLOAT, this attribute sets the division factor. If we set 1 and FLOAT is 12.3, the variable will store 123.

Type By setting 1, we force a binary to signed integer conversion. The value stored in the variable is signed.

Example:

```
Var 0015, Link FSUIPC_IN, Offset $02CC, Length 8, Numbers 1
```

FSUIPC_OUT

Link with FSUIPC module using offset and length. When variable change value, this new value is sended through FSUIPC to destine.

Attributes :

Name Assigns a symbolic name to a variable.

Value Sets an initial value.

Offset FSUIPC offset.

Length Variable size in bytes. Values can be 1, 2, 4 and 8.

Numbers If we work with 8 bytes FLOAT, this attribute sets the division factor. If we set 1 and FLOAT is 123, the variable will store 12.3.

Type By setting 1, we force a binary to signed integer conversion. The value stored in the variable is signed.

Example:

```
Var 0015, Link FSUIPC_OUT, Offset $02CC, Length 8, Numbers 1, Value 23
```

FSUIPC_INOUT

Is the combination of FSUIPC_IN and FSUIPC_OUT. FSUIPC offset changes are sent to the variable, and variable changes are sent to the FSUIPC offset.

Attributes :

Name Assigns a symbolic name to a variable.

Value Sets an initial value.

Offset FSUIPC offset.

Length Variable size in bytes. Values can be 1, 2, 4 and 8.

Numbers If we work with 8 bytes FLOAT, this attribute sets the division factor. If we set 1 and FLOAT is 123, the variable will store 12.3. The opposite for the reception process.

Type By setting 1, we force a binary to signed integer conversion. The value stored in the variable is signed.

Example:

```
Var 0015, Link FSUIPC_INOUT, Offset $02CC, Length 8, Numbers 1, Value 23
```

IOCP

We can link with the SIOC IOCP clients modules. Any change in the destination IOCP variable is mirrored in this variable.

Attributes :

Name Assigns a symbolic name to a variable.

Value Sets an initial value.

Offset Number of the destination IOCP variable.

Device Number of client module used. Values 0 or 1 allowed. (Client module default is 0).

Example:

```
Var 0015, Link IOCP, Offset 48, Device 1, Value 0
```

SUBROUTINE

Variables with this link are converted into a subroutine. The associated script will be automatically executed with the CALL command.

Example:

```
Var 9999, Link SUBROUTINE
```

IOCARD_SW

With this link we can read the values from the Master card inputs and configure them as buttons or switches.

If the input is ON, the variable will store 1; if the input is OFF, the variable will store 0.

Attributes :

Name Assigns a symbolic name to a variable.

Input Defines the input number linked to the variable.

Type By setting **Type** to P the variable will remain 1 or 0 until a new activation of the input. This simulates a toggle switch. By setting **Type** to I, the variable follows the state of the input, just like a push button.

Example:

```
Var 0378, name SW1, Link IOCARD_SW, Input 12
```

IOCARD_OUT

With this link we can access to the digital outputs in Master card.

If the variable value is 0, the output is disabled; if the value is 1 the output is enabled.

Attributes :

Name Assigns a symbolic name to a variable.

Output Defines the output number linked to the variable.

Example:

```
Var 0398, name OUT1, Link IOCARD_OUT, Output 23
```

IOCARD_DISPLAY

With this link we can control the Displays card.

The variable value is send to the selected group of displays. If the card is DisplaysII, then special characters can be sent.

The variable values for this special characters are:

-999999 = display OFF.

-999998 = display shows "-"

-999997 = display shows "6"

-999996 = display shows "t"

-999995 = display shows "d"

-999994 = display shows "_"

The 7-segment displays in a DisplayII card can be **dimmed** (versions since March 2006). You can adjust the brightness according to the environment conditions. Dimming is done by sending a value to digit 15 of the DisplayII card (digits are numbered 0..15). The drawback is that digit 15 cannot be used anymore for display purposes. You can select the dimmer value sending first the code -999994 and the dimmer value (0 to 15).

Attributes :

Name Assigns a symbolic name to a variable.

Digit Sets the first display in the group of displays. The first display is the less significant one.

Numbers Sets the number of display+s in the displays group (for example, course indication would be 3). We have to consider one extra display if negative numbers can be shown.

Type For future options. Not implemented yet.

Example:

```
Var 0008, name heading, Link IOCARD_DISPLAY, Digit 1, Numbers 3
```

Example of dimming :

```
Var 70 name D_Displ Link IOCARD_DISPLAY Digit 15 Numbers 1
Var 5 name Dimmer Link SUBROUTINE
{
  &D_Displ = -999994
  &D_Displ = &Dimmer
}
```


IOCARD_ENCODER

With this link we can control several types of rotary encoders connected to Master card. For each detent, the variable stores the value 1 multiplied by the factor defined in the Acceleration attribute. Then, once the script has been executed, the variable value returns to 0, but in this case there is no event generated, so the script is not more executed.

The value will be negative if we turn in the opposite direction.

Attributes :

Name Assigns a symbolic name to a variable.

Input Defines the first Master card input where the encoder is connected (a simulated encoder uses 3 consecutive inputs; rest of encoders use 2 consecutive inputs). For any case only the first input is defined.

Acceleration Coefficient to apply for each detent.

Type There are three types of encoders:

0 : (default): rotary switch simulating an encoder.

1 : two out of phase signals encoder (requires Encoders card).

2 : Gray type encoders directly connected to Master card inputs.

Example:

```
Var 0001, Link IOCARD_ENCODER, Input 11, Aceleration 6, Type 2
```

IOCARD_ANALOGIC

With this link we can read from the USB Expansion card analog axes.

The values from the A/D converter is stored in the variable. Values vary between 0 and 255.

Attributes :

Name Assigns a symbolic name to a variable.

Input Sets the axe (#1-4).

PosI Sets the left position (normal value is 0)

PosC Sets the center position (normal value is 127)

PosR Sets the right position (normal value is 255)

Example:

```
Var 2032, name POT1, Link IOCARD_ANALOGIC, Input #1, PosL 1, PosC 128, PosR 255
```

IOCARD_SERVO

With this link we can control IOCard Servos2 card. This card controls 2 servo motors.

These cards are connected to some Master card consecutive outputs. Each servo needs 8 outputs (10 in the case of 10 bits resolution – modified card).

The variable value is sent to the servo after doing a compensating process depending on the left, center and right positions. Values vary between 0-255 (0-1023 in the case of 10 bits resolution).

The servo that is controlled is the one connected to the outputs defined in the Output parameter.

Attributes :

Name Assigns a symbolic name to a variable.

Output Defines the first output at which the servo is connected.

Type Default:: the resolution is 8 bits.

If we want 10 bits resolution, we will set 2.

PosL Sets the left position (normal value is 1)

PosC Sets the center position (normal value is 127)

PosR Sets the right position (normal value is 255)

Example:

```
Var 1346, Link IOCARD_SERVO, Output 11, PosL 10, PosC 120, PosR 200
```

IOCARD MOTOR

With this link we can control stepper motors or DC motors connected to the corresponding cards.

These cards are connected to some Master card consecutive outputs: 2 outputs for DC motors or 4 outputs for stepper motors.

When the variable value is 127, any type of motor is stopped.

For values higher than 127, the motor turn in one direction (the higher the value the higher the speed).

For values lower than 127, the motor turn in the opposite direction (the lower the value the higher the speed).

Attributes :

Name Assigns a symbolic name to a variable.

Output Defines the first output at which the motor circuit is connected (2 outputs for DC motor or 4 outputs for stepper motor).

Acceleration Acceleration coefficient. 1 is the higher value and 255 the lower.

Type There are different types that can be used (even simultaneously).

(default): a stepper motor will be controlled.

C : a DC motor will be controlled.

M: in the case of stepper motors, it will run with half steps.

D: to obtain reverse direction.

Example:

Var 9675, Link IOCARD_MOTOR, Output 30, Aceleration 12, Type MD

USB KEYS

With this link we can access USB_KEYS cards.

Each pressed key is sent to the variable, that takes the corresponding value, runs the script and returns its value to 0.

Is possible to link with the keyboard emulator module just setting the TYPE attribute to K. This way, the key pressed is sent by USB_KEYS as defined in the SIOC.INI file.

Attributes:

Name Assigns a symbolic name to a variable.

Value Assigns an initial value.

Device Is the card device number where the variable is assigned. If it is 0, the first available number is considered (default). Clicking on the button gives access to a list of available IOCards USB cards connected. In this case we can select the device from the list. If it is not selected, a 0 will be considered.

Type Optionally, **K** can be set to indicate a keyboard simulation.

Example:

```
Var 0001, name Teclas, Link USB_KEYS, Device 123, Type K
```

USB STEPPER

With this link we can access USB_STEPPER cards.

The variable values are sent to the motor controller with the specific parameters.

Angle can take the values from 0 to 359,99 degrees, with two decimals. Variable values will vary from 0 to 35999 (value that corresponds with 359,99 degrees).

Attributes :

Name Assigns a symbolic name to a variable.

Device Is the card device number where the variable is assigned. If it is 0, the first available number is considered (default). Clicking on the button gives access to a list of available IOCards USB cards connected. In this case we can select the device from the list. If it is not selected, a 0 will be considered.

Type Optionally, **H** can be set to force the motor to work with half steps.

Output Sets the corresponding motor (1-3).

PosI Sets the speed value (0-255) (see the card manual to adjust this value).

PosC Sets the steps values or calibration steps. If it's 0, the counting of steps by turn is made automatically (0-65535).

PosD Sets the maximum number os steps by 1/10 seconds (0-255) (see the card manual to adjust this value). If the motor loses control, reduce this value.

Example:

```
Var 0234, Link USB_STEPPER, Output 2, PosL 6, PosC 0, PosR 4, Type H
```

USB SERVOS

With this link we can access USB_SERVOS cards.

The variable value is sent to the servo motor after doing a compensation depending on left, center and right positions. Values can vary from 0 to 1023.

The value 0 will indicate disconnection.

Attributes :

Name Assigns a symbolic name to a variable.

Device Is the card device number where the variable is assigned. If it is 0, the first available number is considered (default). Clicking on the button gives access to a list of available IOCards USB cards connected. In this case we can select the device from the list. If it is not selected, a 0 will be considered.

Output Sets the corresponding motor (1-6).

Type For future options. Not implemented yet.

PosL Sets the left position (normal value would be 1).

PosC Sets the center position (normal value would be 512).

PosR Sets the right position (normal value would be 1023).

Example:

```
Var 0078, Link USB_SERVOS, Device 23, Output 5, PosL 1, PosC 512,  
PosR 1023
```

USB RELAYS

With this link we can access USB_RELAYS cards.

If the value is 0, the relay indicated in Output is set to one position. If the value is 1, the relay is set to the other position, and the corresponding led on the card lights.

Attributes :

Name Assigns a symbolic name to a variable.

Device Is the card device number where the variable is assigned. If it is 0, the first available number is considered (default). Clicking on the button gives access to a list of available IOCards USB cards connected. In this case we can select the device from the list. If it is not selected, a 0 will be considered.

Output Sets the corresponding relay (1-7).

Example:

```
Var 0587, Link USB_RELAYS, Device 123, Output 6
```

USB_ANALOGIC

With this link we can access the analogue axes of USB_SERVOS, USB_RELAYS and USB_STEPPER cards.

The value from the A/D converter is sent to the corresponding variable. Values can vary from 0 to 255.

Attributes :

Name Assigns a symbolic name to a variable.

Device Is the card device number where the variable is assigned. If it is 0, the first available number is considered (default). Clicking on the button gives access to a list of available IOCards USB cards connected. In this case we can select the device from the list. If it is not selected, a 0 will be considered.

Input Assigns the corresponding axe (1-5 depending on the available axes on board).

PosI Sets the left position (normal value would be 0).

PosC Sets the center position (normal value would be 127).

PosR Sets the right position (normal value would be 255).

Example:

```
Var 0378, Link USB_ANALOGIC, Device 254, Input 3, PosL 1, PosC 127,  
PosR 255
```

SOUND

With this link we can access the sound module.

The variable will act according to the TYPE settings (**S** or not).

The .wav files will be defined in the SIOC.INI file, consecutively numbered from 1.

If nothing is defined for TYPE, when the variable value is different from 0, the sound module will open the corresponding .wav file (defined in SIOC.INI) and will play it.

If TYPE is set to **S**, when the variable value is different from 0, the sound module will stop and close the corresponding .wav file.

Attributes :

Name Assigns a symbolic name to a variable.

TYPE S can be set to force a sound stop as described above.

Example:

```
Var 0034, name ALARM, Link SOUND, Type S
```

KEYS

With this link we can access the keyboard emulator module.

Variable value can vary from 1 to 200. Each value corresponds to a keys sequence previously set in `SIOC.INI`.

These settings in `SIOC.INI` are in the form:

```
#1=\B\A
```

where #1 is the setting number (if the variable takes this value, the keys sequence is sent) and `\B\A` is the keys sequence.

To send the sequence #1, we make `V0010 = 1`, and automatically the variable value returns to 0 and another sequence can be sent.

There are special codes for special keystrokes (Ctrl, Caps, etc.). The code / (slash) will be written before one of the following letters:

- A = BKSP = #8;
- B = TAB = #9;
- C = ENTER = #13;
- D = ESC = #27;
- E = F1 = #228;
- F = F2 = #229;
- G = F3 = #230;
- H = F4 = #231;
- I = F5 = #232;
- J = F6 = #233;
- K = F7 = #234;
- L = F8 = #235;
- M = F9 = #236;
- N = F10 = #237;
- = F11 = #238;
- P = F12 = #239;
- Q = HOME = #240;
- R = END = #241;
- S = UP = #242;
- T = DOWN = #243;
- U = LEFT = #244;
- V = RIGHT = #245;
- W = PGUP = #246;
- X = PGDN = #247;
- Y = INS = #248;
- Z = DEL = #249;
- 1 = SHIFT_DN = #250;
- 2 = SHIFT_UP = #251;

- 3 = CTRL_DN = #252;
- 4 = CTRL_UP = #253;
- 5 = ALT_DN = #254;
- 6 = ALT_UP = #255;
- \ = \

USB_DCMOTOR

With this link we can access USB_DCMOTOR cards.

The variable value is sent to the controller. Values can vary from 0 (stopped) to 127 (max. speed) to turn right. Add +128 to turn left.

You need send a first value for start the card.

Attributes :

Name Assigns a symbolic name to a variable.

Device Is the card device index where the variable is assigned. If not used, default index is 0.

Output Sets the corresponding motor (1-6).

Example:

```
Var 0078, Link USB_DCMOTOR, Device 2, Output 5
```

Language reference

Assignment

Variable = **Constant** or **Variable**

Variable = **Constant** or **Variable** **Operator** **Constant** or **Variable**

Assigns the operation result to a internal or IOCP variable.

Example:

```
{
  V0002 = V0008 * 3.14
  L2 = 3.8673
  L1 = V0001 AND 128
  C1 = L1 < 5
}
```

CALL

CALL **Variable**(subroutine) **Constant** or **Variable**

Runs the variable associated script, and, if a parameter is included, the variable takes this value.

The script is ran anyway.

Example:

```
{
  CALL V9888
  CALL V1001 326
  CALL V3004 L0
}
```

IF and ELSE condition

IF **Condition**

```
{
  Commands
}
```

ELSE

```
{
  Commands
}
```


Commands after **IF** are only executed if **condition** is true; if it is not true, **commands** after **ELSE** will be executed.

The **ELSE** part is optional. If used, it must to be placed after the **IF** part.

It is possible to indent up to 100 **IF** commands.

The condition can be a boolean variable, or 2 boolean variables linked with **AND / OR operators**, or conditions between variables and/or real constants with the corresponding operators.

Example:

```
{
  IF L1 > 5
  {
    CALL V1000
  }
  ELSE
  {
    IF C2
    {
      L1 = L1 + 1
    }
  }
}
```

Operators list

Operators	Description
-----------	-------------

+	To sum two variables or constants.
-	To subtract a value from two variables or constants.
*	To multiple the value of two variables or constants.
/	To devide the value from two variables or constants.
AND	Logic Operation or AND condition
OR	Logic Operation or "OR" condition
>	"Greater Than" condition.
<	"Less Than" condition.
=	"Equal" condition.
>=	Condition "Greater Than or Equal"
<=	Condition "Less Than or Equal"
<>	"Not Equal" condition.

Functions definition

Variable = **Function** parameter1 parameter2 parameter3

parameter variable, real constant or integer constant.

Every function has 1, 2 or 3 **parameters** maximum, and the final value is assigned to the **variable**.

Example:

```
{
  V0002 = ROUND L0
  L2 = TOBCD V0005
  C1 = TESTBIT V1234 6
  V9888 = TIMER 100 5 10
}
```

ABS

Variable = **ABS** Parameter

Assigns the absolute value (positive) of the **parameter** to the **variable**.

Parameter : variable, real constant or integer constant.

Example :

```
V0002 = ABS V0008
```

ARCT

Variable = **ARCT** **Parameter**

Assigns to the variable the value of arctangent of the **parameter**.

Parameter : variable, real constant or integer constant.

Example :

```
V0002 = ARCT V0008
```

CHANGEBIT

Variable = **CHANGEBIT** **Parameter1** **Parameter2**

Changes the variable bit indicated in **parameter1**, depending on **parameter2** value. Bit changes to 0 if **parameter2** is 0; bit changes to 1 if **parameter2** is 1.

Parameter1 : Variable, real constant or integer constant. Bit to be changed.

Parameter2 : Variable, real constant or integer constant. Mode of change.

Example:

```
V1354 = CHANGEBIT 3 V0034
```

CHANGEBITN

Variable = **CHANGEBITN** **Parameter1** **Parameter2**

Changes the variable bit indicated in **parameter1**, depending on **parameter2** value. Bit changes to 1 if **parameter2** is 0; bit changes to 0 if **parameter2** is 1.

Parameter1 : Variable, real constant or integer constant. Bit to be changed.

Parameter2 : Variable, real constant or integer constant. Mode of change.

Examples:

```
V1354 = CHANGEBITN 3 V0034
```

CLEARBIT

Variable = **CLEARBIT** **Parameter**

Sets to 0 the bit indicated by **Parameter**.

Parameter : Variable, real constant or integer constant. Bit to be changed.

Examples:

```
V0002 = CLEARBIT 3
```

COS

Variable = COS Parameter

Assigns to the variable the value of cosine of the parameter (radians).

Parameter : variable, real constant or integer constant.

Example :

```
V0002 = COS V0008
```

DELAY

Variable = DELAY Parameter1 Parameter2

Runs the associated script previously waiting the time indicated by **Parameter2** (1/10 secs) and assigns the value indicated by **Parameter1** to the **variable**.

Parameter1 : Variable, real constant or integer constant. Value to set.

Parameter2 : Variable, real constant or integer constant. Delay time (1/10 secs).

Example:

```
V1354 = DELAY 57 10
```

DIV

Variable = **DIV** Parameter1 Parameter2

Does the division **Parameter1** / **Parameter2**. The result is stored in the **variable**.

Parameter1 : Variable, real constant or integer constant. Number to be divided.

Parameter2 : Variable, real constant or integer constant.

Example:

```
V0032 = DIV V0034 2
```

FROMBCD

Variable = **FROMBCD** Parameter

Converts the value in **parameter** (BCD) to decimal format and stores it in the **variable** (this format is used in several FSUIPC offsets).

Parameter : variable, real constant or integer constant.

Example:

```
V2302 = FROMBCD V9233
```

LIMIT

Variable = **LIMIT** Parameter1 Parameter2 Parameter3

Increases or decreases the **variable** value with the value in **Parameter3**. If the result is higher than **Parameter2**, then **variable** = **Parameter2**. If the result is lower than **Parameter1**, then **variable** = **Parameter1**.

Parameter1 : Variable, real constant or integer constant. Lower value.

Parameter2 : Variable, real constant or integer constant. Upper value.

Parameter3 : Variable, real constant or integer constant. Increment/Decrement.

Example:

```
V9302 = LIMIT 0 60000 V0456
```

LOGN

Variable = **LOGN** Parameter

Assigns the natural logarithm of **parameter** to the **variable**.

Parameter : variable, real constant or integer constant.

Example :

```
V0002 = LOGN V0008
```

NOT

Variable = NOT **Parameter**

Assigns to the boolean **variable** the opposite value as the one indicated by **Parameter**.

Parameter : Boolean variable.

Example:

```
C1 = NOT C0
```

MOD

Variable = MOD **Parameter1** **Parameter2**

Calculates the remainder of the integer division **Parameter1** / **Parameter2**. The value is stored in **variable**.

Parameter1 : Variable, real constant or integer constant. Number to be divided.

Parameter2 : Variable, real constant or integer constant.

Example:

```
V0032 = MOD V0034 2
```

POWER

Variable = POWER **Parameter1** **Parameter2**

Assigns to the **variable** the value of **Parameter1** as base and **Parameter2** as exponent.

Parameter1 : Variable, real constant or integer constant. Base

Parameter2 : Variable, real constant or integer constant. Exponent

Examples:

```
L0 = POWER V000 0.5
```

RANDOM

Variable = RANDOM **Parameter1** **Parameter2**

Variable gets a randomly generated value between **Parameter1** and **Parameter2**.

Parameter1 : Variable, real constant or integer constant. Lower limit for the randomly generated numbers.

Parameter2 : Variable, real constant or integer constant. Higher limit for the randomly generated numbers.

Example:

```
V1354 = RANDOM 0 9
```

ROTATE

Variable = ROTATE **Parameter1** **Parameter2** **Parameter3**

Increases or decreases the **variable** value with the value indicated by **parameter3**. When increase, if the result is higher than **Parameter2**, then **variable** = **Parameter1**. When decrease, if the result is lower than **Parameter1**, then **variable** = **Parameter2**.

This cyclic behavior is typical of gauges like VOR OBS.

Parameter1 : Variable, real constant or integer constant. Lower value.

Parameter2 : Variable, real constant or integer constant. Upper value.

Parameter3 : Variable, real constant or integer constant. Increment/Decrement.

Example:

```
V9002 = ROTATE 1 360 V0034
```

ROUND

Variable = ROUND **Parameter**

Assigns the rounded value (to the closer integer) of **Parameter** to the **variable**.

Parameter : Variable, real constant or integer constant.

Example:

```
V0002 = ROUND L1
```

SETBIT

Variable = SETBIT **Parameter**

The **variable** bit indicated by **Parameter** is set to 1.

Parameter : Variable, real constant or integer constant. Bit to be changed.

Example:

```
V0002 = SETBIT V0034
```

SETSOUND

Variable = SETSOUND **Parameter1** **Parameter2** **Parameter3**

The function need set the **variable** local assigned to the sound defined in `sioc.ini`.

The function change the parameters of Frequency, Volume and Pan of sound selected.

Parameter1 : Variable or integer constant. Frequency value.

(100 to 100000 0 = initial value, -1 = default)

Parameter2 : Variable or integer constant. Volume value.

(0 to 100, -1=default)

Parameter3 : Variable or integer constant. Pan value.

(-100 (left) to +100 (right), 0=center, -1=default)

Example:

```
Var 0002, Link SOUND, Type S
Var 0001, Link SOUND
Var 0000, Value 0
{
  V0001 = 1

  V0001 = 2
  L0 = 2
  L0 = SETSOUND -1 ,40 ,-1
  V0009 = -99
  V0009 = TIMER 100 ,1 ,3
}

Var 0009, Link SUBROUTINE
{
  L0 = 2
  L0 = SETSOUND -1 ,-1 ,V0009
  IF V0009 = 100
  {
    V0009 = TIMER -100 ,-1 ,3
  }
  IF V0009 = -100
  {
    V0002 = 2
  }
}
```

SIN

Variable = **SIN** **Parameter**

Assigns to the **variable** the value of sine of the **parameter** (radians).

Parameter : variable, real constant or integer constant.

Example :

```
V0002 = SIN L1
```

TESTBIT

Variable = **TESTBIT** **Parameter1** **Parameter2**

If the variable is boolean: the **variable** is set to true (false) if the bit of **parameter1** indicated by **parameter2** is 1 (0).

If the variable is an integer or real: the **variable** is set to 1 (0) if the bit of **parameter1** indicated by **parameter2** is 1 (0).

Parameter1 : Variable, real constant or integer constant. Number to be tested.

Parameter2 : Variable, real constant or integer constant. Bit number.

Example:

```
C0 = TESTBIT V0008 2
```

TIMER

Variable = **TIMER** Parameter1 Parameter2 Parameter3

The script associated to the **variable** is run periodically, as indicated in **Parameter3** (1/10 secs).

For each loop (the first one is run once the time indicated by **Parameter3** has passed) the **variable** value increases/decreases as indicated by **Parameter2**. The process ends when the **variable** value reaches **Parameter1**.

Parameter1 : Variable, real constant or integer constant. Final value.

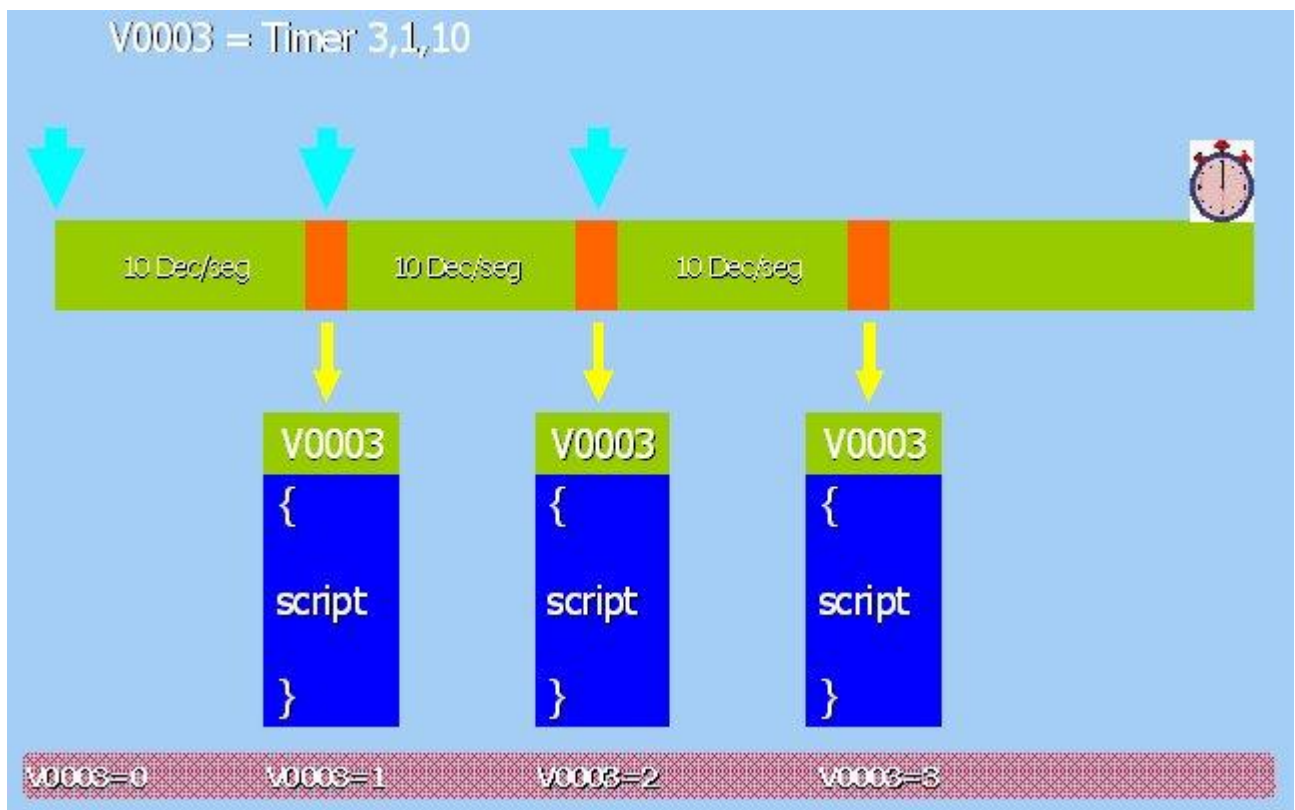
Parameter2 : Variable, real constant or integer constant. Increment/Decement.

Parameter3 : Variable, real constant or integer constant. Recurrence.

Example:

```
V9000 = 0  
V9000 = TIMER 1000 5 10
```

This script will be executed every 100 ms. Everytime, V9000 will be incremented by 5, until it reaches the value of 1000. Then the process will stop.



TOBCD

Variable = TOBCD **Parameter1**

Converts the value in **parameter1** into BCD format and stores it in **variable** (this format is used in several FSUIPC offsets).

Parameter1 : Variable, real constant or integer constant.

Example:

```
V2302 = TOBCD V9233
```

TOGGLE

Variable = TOGGLE **Parameter1**

Runs a 'toggle' process (set to 1 and then returns to 0) on the **variable** bit indicated by **parameter1**.

Parameter1 : Variable, real constant or integer constant.

Example:

```
V0023 = TOGGLE 3
```

TRUNC

Variable = TRUNC **Parameter1**

Truncate the decimal part of **parameter1**, converting it to an integer. The integer value is stored in the **variable**.

Parameter1 : Variable, real constant or integer constant.

Example:

```
L1 = TRUNC L0
```