

Global Variables: Alpha_Steps

INT ALPHA_STEPS

It contains a value between 0 and 255 to indicate the amount of transparency tables that are going away to define. By defect 16 levels of transparency are initialized.

A total of 256 levels of transparency is always created that are initialized to equal values in 256/ALPHA_STEPS blocks equal levels, for example, with 16 levels, the ALPHA levels until the 16 have the same degree of transparency. INT ALPHA_STEPS

It contains a value between 0 and 255 to indicate the amount of transparency tables that are going away to define. By defect 16 levels of transparency are initialized.

A total of 256 levels of transparency is always created that are initialized to equal values in 256/ALPHA_STEPS blocks equal levels, for example, with 16 levels, the ALPHA levels until the 16 have the same degree of transparency.

ALSO SEE:

[ALPHA](#)

Global Variables: ARGV

ARGV

INT ARGV

It contains the number of parameters received by the program

DESCRIPTION:

This variable contains the total parameters received by the program, included the name of the program.

Global Variables: ARGV

ARGV

string ARGV[32]

It contains the parameters received by the program

DESCRIPTION:

This variable contains the value of the parameters received by the program, included the name of program, which is in argv[0].

Fenix saves internally enough space for 32 parameters, but the total number of parameters received is obtained by [ARGC](#).

Global Variables: ASCII

ASCII

INT ASCII

It contains information of the last character that has been pressed in the keyboard

DESCRIPTION:

It contains the value ASCII of the key pressed in the last frame or a 0 if any key was not pressed, If some keys are pressed in a frame the pulsations are saved in a small buffer that avoids the loss of pulsations.

Global Variables: FPS

FPS

INT FPS

It indicates the current speed in Frames/Second of the program

DESCRIPTION:

This variable is updated with the current value of frames per second of the executed program.

The theoretical value of this variable must be equivalent at the value indicated with [SET FPS](#).

Global Variables: FULL_SCREEN

FULL_SCREEN

INT FULL_SCREEN

It adjusts if the program is executed in window mode or full screen

DESCRIPTION:

This variable indicates if the program is executed in window mode or full screen. To indicate window mode, we must indicate that his value is **FALSE** (or the value 0), for full screen the value of this variable must be **TRUE** (or a value different from 0). The default value is **TRUE**.

The value of this variable must be indicated before calling the function [SET_MODE](#).

NOTES:

If you want to change for example: a program from full screen to window mode you must to assign the correct value to this variable and then call again to [SET_MODE](#).

Global Variables: GRAPH_MODE

GRAPH_MODE
INT GRAPH_MODE

It adjusts the depth of colour that will be used

DESCRIPTION:

This variable indicates if the program uses a depth of colour of 8 or 16 bits. To indicate that the program uses a depth of 16 bits his value must be 1, for 8 bits his value must be 0. The default value is 0 (depth of 8 bits). Too can be indicated the use of 2XScale filter (only available for previous modes at 400x400 pixels).

To simplify this task, there are the following constants:

To indicate the use of 16 bits: **MODE_16BITS**

To indicate the use of 2XScale filter: **MODE_2XSCALE**

The value of this variable (to use 16 bits) must be indicated before calling to the function [SET_MODE](#).

If you want to use both constants, simply use:

```
graph_mode = mode_16bits | mode_2xscale ;
```

Global Variables: SCAN_CODE

SCAN_CODE

INT SCAN_CODE

It contains the code of the key pressed in the last frame

DESCRIPTION:

This value is the predefined constant for the function **KEY**. As the use of [ASCII](#) if the player has pressed two or more keys, the additional pulsations are saved in a small buffer to avoid their losses.

It is recommend tu use **KEY** to check what key has been pressed, so it is more efficient and quick that check it manually using code in our program.

Global Variables: TIMER

TIMER

INT TIMER[9]

Ten accessible programmable chronometers at any moment

DESCRIPTION:

Fenix has 10 accessible chronometers through the array of variables timer (from timer[0] to timer[9]). As it happens with any array to get the value of timer as a integer variable is just like to get the value of timer[0].

These timers are initialized to value 0 automatically when the program begins and are increased 100 times per second automatically.

In any moment the value of each one can be changed, for example inicializándolo de nuevo a 0, to control an ammount of concrete time. For example, to make a clock, we would must check the chosen timer against a value greater to 100 and to reduce it 100 to that timer.

NOTES:

The global variables are updated each instruction FRAME with the object of language therefore the values read in a timer will not be consecutive unless our program is being executed to 100 frames per second.

The verifications with the timers must be always made with the operators of \geq or $>$ and never with the $=$ so that it is not secure that after the update of the timer the value will be exactly equal to the wished one.

Local Variables: ALPHA

ALPHA

INT ALPHA

It indicates the transparency of the graph of the process

DESCRIPTION:

It contains a value between 0 and 255 to indicate the transparency degree that has the graph.
By default this value is set to 255.

NOTES:

ONLY AVAILABLE IN CVS VERSION

Local Variables: ANGLE

ANGLE

INT ANGLE

It specifies the angle of visualization in thousandth of degree.

DESCRIPTION:

This angle is the one that the interpreter uses to rotate the graph of the process before drawing it in screen. A value of 90000 will rotate the graph 90° to the left. Its value by default is 0 (without rotating).

Optionally it can be combined with the use of XGRAPH for that the graph of process changes according to the angle of visualization.

Local Variables: BIGBRO

BIGBRO

INT BIGBRO

Identifier of the last process created before ours

DESCRIPTION:

This identifier makes reference to the last process that the creator of the current process created before it. It will be 0 if any previous process to the current had not been created.

Local Variables: CTYPE

CTYPE

INT CTYPE

It defines the type of coordinates of the process

DESCRIPTION:

It allows to specify the kind of source of coordinates for the process, being affected the calculation of its positioning in screen.

Their values can be:

C_SCREEN

C_SCROLL

C_M7

C_SCREEN makes reference to the coordinates of screen

C_SCROLL makes reference to the coordinates of a **SCROLL** zone

C_M7 makes reference to the coordinates of a **Mode7** zone

The value by default is **C_SCREEN**, for other values must be specified in the variable **CFLAGS** the zones of **SCROLL** or **Mode7** in which the process will be showed.

Local Variables: CFLAGS

CFLAGS

INT CFLAGS

It indicates in what windows of scroll or modo7 the process visualizes

DESCRIPTION:

This variable allows to determine that window of scroll or modo7 is going to be used to visualize the process. Its value can obtain the one or sum of several of the following constants (sum or operation OR)

C_0 = 1

C_1 = 2

C_2 = 4

C_3 = 8

C_4 = 16

C_5 = 32

C_6 = 64

C_7 = 128

C_8 = 256

C_9 = 512

The value by defect indicates that it will see in all the active windows of scroll and/or modo7 (independently to the value of C_FLAGS, a window of scroll or modo7 must be active so that the processes can visualize in her).

This variable only has sense if local variable CTYPE is also initialized

Local Variables: FATHER

FATHER

INT FATHER

Identifier of the father process

DESCRIPTION:

This identifier makes reference to the process that created the current process. Using this identifier can be indicated the variables of that process, for example `father.graph` or `father.x`.

If the father process dies before that this process, the value of father will be 0.

Local Variables: FILE

FILE

INT FILE

Identifier of the current fpg file in use by the process.

PARAMETERS:

This command has no parameters.

DESCRIPTION:

This is a predefined local variable. This results in each Process having a value in its very own file variable.

If you happen to have several fpgs loaded at once, the file variable will contain the ID of the fpg currently in use by the process.

NOTE:

When you are using more than one file loaded, it is therefore necessary to indicate locally in each process which file you wish to use.

Local Variables: FLAGS

FLAGS

flags

DESCRIPTION:

This flags variable can contain values that are used to manipulate a graphics appearance. This flag can take the following values:

0: Normal graphic (no change)

1: Horizontal mirror

2: Vertical mirror

3: Horizontal and vertical mirror

4: Transparent graphic

5: Transparent and horizontally mirrored graphic

6: Transparent and vertically mirrored graphic

7: Transparent, horizontal and vertically mirrored graphic

By default this variable's value is set to 0.

Local Variables: GRAPH

GRAPH

INT GRAPH

Identificador del gráfico del proceso

DESCRIPTION:

Este identificador referencia un gráfico dentro del fichero FPG indicado por la variable local FILE.

En caso de tratarse de un gráfico de la librería FPG de sistema, identificada con el número 0, puede contener el identificador devuelto por cualquier función de carga de gráficos, como por ejemplo LOAD_MAP O LOAD_PNG, o de creación de nuevos gráficos, como por ejemplo NEW_MAP o MAP_CLONE.

Local Variables: Priority

INT PRIORITY

This variable contains the value of priority of the present process. The processes that have a greater priority execute before since at the time of drawing each FRAME of the game the interpreter uses the value of this variable to determine the execution order.

It is necessary to consider that the processes is executed first and only when no longer any processes to execute in the present FRAME, they are drawn in screen.

The value of this variable does not have anything to do with the ordering in depth of the processes, for that assignment must be used local variable Z.

ALSO SEE:

Z

Local Variables: Region

INT REGION

It indicates the number of section that will be used to trim the graph in screen. By defect it will be worth 0 (the region by defect that is equivalent to the complete screen). Its value must correspond with a number of section initialized with DEFINE_REGION

If the graph is within the zone of cut of the section will be visualized, the parts of the graph of the process that are outside the zone defined by the region will trim and they will not be shown.

ALSO SEE:

DEFINE_REGION

Y

Local Variables: RESOLUTION

RESOLUTION

Factor of extension for coordinates X and Y

DESCRIPTION

It allows to increase the resolution of coordinates X and Y of process, so it indicates whichever units logics exist by each pixel of screen. For example to advance 400 units in a process with resolution 100 only moves 4 pixels in the screen.

It is recommended to use this variable to slow down the movement of objects that must be moved slowly or to increase the smoothness in complex movements that depend on the precision in mathematical calculations like parabolic shots, gravity, etc.

By default his value is 0, indicating that equivalence is of 1:1

SEE ALSO

[X](#) Position X of the process

[Y](#) Position Y of the process

Local Variables: SON

SON

INT SON

Identifier of the last son process

DESCRIPTION:

This identifier makes reference to the last process that current process has created. If the process has not been created, his value will be 0.

Local Variables: SMALLBRO

SMALLBRO

INT SMALLBRO

Identifier of the process created after ours

DESCRIPTION:

This identifier makes reference to the first process that the creator of the current process created after it. It will be 0 if any later process to the current had not been created.

Local Variables: X

X

INT X

Position X of the process

DESCRIPTION:

This position X is relative to the type of coordinates indicated by the variable CTYPE. Changing the value of this variable, the graph of process will be drawn in the position indicated in the next frame.

By default the value for this coordinate is in pixels, where 0,0 corresponds to the top left corner of the destiny zone. To use coordinates with smaller units must be indicated the factor of reduction using the variable RESOLUTION. Automatically it will be rounded the values of fractions of pixel before drawing the graph on screen.

The indicated coordinate X is the one of the control post 0 of the graph or in its defect to its geometric center.

XGRAPH

Local Variables: Y

Y

INT Y

Position Y of the process

DESCRIPTION:

This position Y is relative to the type of coordinates indicated by the variable CTYPE. Changing the value of this variable, the graph of process will be drawn in the position indicated in the next frame.

By default the value for this coordinate is in pixels, where 0,0 corresponds to the top left corner of the destiny zone. To use coordinates with smaller units must be indicated the factor of reduction using the variable RESOLUTION. Automatically it will be rounded the values of fractions of pixel before drawing the graph on screen.

The indicated coordinate Y is the one of the control post 0 of the graph or in its defect to its geometric center.

Scroll Struct

Mode7 Struct

Joy Struct

Setup Struct

Data Types: FLOAT

FLOAT

DESCRIPTION:

A floating point value are numeric values which include a fractional part.

For example 1.5, 2.2, or -3.456

NOTES:

Floating point numbers are defined in Fenix using #. If you don't declare a varible it will default to an INTERGER.

Data Types: INT

INT

DESCRIPTION:

Integer values are numeric values with no fractional part in them, in other words they are whole numbers.

For example 4,24,-78

NOTES:

Integers are defined in Fenix using %. If you don't declare a variable it will default to an INTERGER.

Data Types: STRING

STRING

DESCRIPTION:

String values are used to store text.

For example "this is a string" , "" , "end of program"

NOTES:

Strings are defined in Fenix using \$! If you don't declare a variable it will default to an INTERGER.

Data Types: VARIABLES

VARIABLES

DESCRIPTION:

Variables may be of any basic data type. The variables type is determined by a special character which follows its identifier.

VARIABLE TYPES:

% = For interger variables

\$ = For string variables

= For floating point variables

NOTES:

If you don't provide the tag type the variable will by default be defined as an Interger value.

Preprocessor: #define

#define CONSTANT VALUE

It assigns a text to a macro

PARAMETERS

CONSTANT IDENTIFIER: Name of constant

TEXT VALUE: Value assigned to the constant

DESCRIPTION

This director of the preprocessor allows to define macros, words that when appearing in the code will be substituted by the text specified in the director #define, before being interpreted by the compiler.

These macros cannot receive parameters, like in C.

It is important to understand the differences between the defined constants with CONST and the macros #DEFINE. The constants have a determined type, and the compiler knows his fixed value final. If a constant with value "2+2" is declared, the compiler will use value 4. However, a defined macro as "2+2" cause the appearance of text "2+2" instead of their name in the code.

It observes east example:

```
#define I CALCULATE 8+10 SAY (4*CALCULO);
```

The compiler in fact understands it like:

```
SAY (4*8+10);
```

... caul surely does not cause the wished effect. In order to avoid this kind of problems, it uses parenthesis whenever a macro can contain expressions.

Another important use of the macros consists of using them jointly with # ifdef or # ifndef to cause that certain sections of code are compiled of optional form.

ALSO SEE

IFDEF Provoca that a code section is only compiled if a macro is defined

IFNDEF Provoca that a code section is only compiled if a macro is not defined

IF Provoca that a code section is compiled conditionally

CONST

Preprocessor: #if

#if EXPRESSION

...

else

...

endif

It causes that a code section is compiled conditionally

PARAMETERS

INT EXPRESSION: Constant expression of whole type

DESCRIPTION

This director of the preprocessor allows to mark a section of code (all the lines between # ifdef and # endif) like optional. The compiler will only receive these lines as long as the condition defined after # if is certain. This condition must be an expression of constant result, that is to say, that it only uses macros # defines and numbers or chains. A textual identifier in this expression will not be interpreted like varname, but like chain. Therefore if is valid to do "# MACRO == similar VALUE" or comparisons.

The section # else... # endif is optional, being able to restrict the block to only # if... # endif.

ALSO SEE

IFDEF Provoca that a code section is only compiled if a macro is defined

IFNDEF Provoca that a code section is only compiled if a macro is not defined

DEFINES Assigns a text to a macro

Preprocessor: #ifdef

```
# ifdef CONSTANT
```

```
...
```

```
# endif
```

It causes that a code section is only compiled if a macro is defined

PARAMETERS

CONSTANT INT: Name of a macro # defines

DESCRIPTION

This director of the preprocessor allows to mark a section of code (all the lines between # ifdef and # endif) like optional. The compiler will only receive these lines as long as the constant whose name specifies after # ifdef finds defined (even although it defines myself in target).

ALSO SEE

IFNDEF Provoca that a code section is only compiled if a macro is not defined

DEFINES Assigns a text to a macro

Preprocessor: # ifndef

```
# ifndef CONSTANT
```

```
...
```

```
# endif
```

It causes that a code section is only compiled if a macro is not defined

PARAMETERS

CONSTANT INT: Name of a macro # defines

DESCRIPTION

This director of the preprocessor allows to mark a section of code (all the lines between # ifdef and # endif) like optional. The compiler will only receive these lines as long as the constant whose name specifies after # ifdef not finds defined (a definition in target considers a valid definition to all the effects).

It is important to understand that the compiler will not receive the excluded lines, reason why any other type or syntax errors will not be detected in these lines.

ALSO SEE

IFDEF Provoca that a code section is only compiled if a macro is defined

DEFINES Assigns a text to a macro