Table of Contents

Table of Contents

Table of Contents

1 Angle

Up to Local Variables

1.1 Definition

INT angle = 0

Angle is a predefined local variable which holds the angle (measured in 1/1000 of a degree) at which the graphic of that process (assigned by the local variable graph) will be drawn. It also influences the result of the function advance().

An angle of 0 means to the right, 90000 means up, 180000 means left and 270000 and -90000 mean down.

1.2 Example

To make the graphic of a process spin:

```
import "mod_grproc"
import "mod_map"
import "mod_wm" // for exit_status
import "mod_key" // for key()

Process Main()
Begin
    graph = map_new(100,100,8);  //Create a cyan square and assign it to 'graph'
    map_clear(0,graph,rgb(0,255,255));
    x = 160;     //Position the graphic's center
    y = 100;     //in the center of the screen
    Repeat
        angle += 1000;    //increase the angle of graphic by 1000 each frame. 1000 = 1 degree.
        frame;
    Until(key(_ESC)) || exit_status)
OnExit
    map_unload(0,graph);
End
```

Used in example: map_new(), map_clear(), rgb(), map_unload(), key(), exit_status, graph, x, y, **angle**

This process will spin the cyan square by 1 degree each frame.

Template:Locals

2 Argc

2.1 Definition

INT argc

Argc is a global variable, holding the number of arguments passed when calling BGDI, including the bytecode file. The arguments can be found in the global variable argv.

Template:Moduledocbox

3 Argv

Up to Global Variables Up to Internal

3.1 Definition

STRING[32] argv

Argv is a global variable, holding the arguments with which BGDI was called, including the bytecode file.

argv[0]	- The bytecode file (possibly without extension).
argv[1]	- First argument.
argv[2]	- Second argument.
argv[*n*]	- *n*th argument.

If an argument was not given, the corresponding string will be `""` (empty string). The number of arguments passed can be found with argc. This means that `argv[argc-1]` is the last argument.

3.2 Example

```
import "mod_say"

Process Main()
Private
    int i;
Begin

    say("Bytecode file: " + argv[0]);

    i = 1;
    while(i<argc)
        say("Argument " + i + ": " + argv[i]);
        i++;
    end

End
```

Running this on Windows XP:

```
> bgdi argv mies noot "mies noot" "'mies noot'" "\"mies noot\"" 'mies noot'
Bytecode file: argv
Number of arguments: 8
Argument 1: mies
Argument 2: noot
Argument 3: mies noot
Argument 4: 'mies noot'
Argument 5: "mies noot"
Argument 6: 'mies
Argument 7: noot'
```

Running this on Linux:

```
$ bgdi argv mies noot "mies noot" "'mies noot'" "\"mies noot\"" 'mies noot'
Bytecode file: argv
Number of arguments: 7
Argument 1: mies
Argument 2: noot
Argument 3: mies noot
Argument 4: 'mies noot'
Argument 5: "mies noot"
Argument 6: mies noot
```

Here you can see some interesting things:

text	produces an argument *word* for each word (words are separated by spaces)
"*text*"	produces the argument *text*
"'*text*'"	produces the argument *'text'*.
"\"*text*\""	produces the argument *"text"*
'text'	doesn't combine multiple words into one argument on Windows, but it does on Linux.

The passing of arguments is totally unrelated to Bennu and is OS dependent. Usually the doubleqoutes (" ") work, but as you can see, the single quotes (' ') don't always work to combine multiple words into one argument.

Template:Moduledocbox

4 Bigbro

Up to Local Variables

4.1 Definition

INT bigbro

Bigbro is a predefined local variable. **Bigbro** holds the ProcessID of the process/function that was immediately called before by the father of the current process/function. There are several other local variables which refer to the ProcessID of a related process:

- Father
- Son
- Smallbro

Template:Locals

5 Cdinfo

Up to Global Variables

5.1 Definition

Struct cdinfo

cdinfo is a global variable struct, containing information about a CD/DVD drive, last obtained using cd_getinfo().

5.2 Members

filling this out later...

Member name	- *Description*
INT current_track	-
INT current_frame	-
INT tracks	-
INT minute	-
INT second	-
INT subframe	-
INT minutes	-
INT seconds	-
INT subframes	-
STRUCT track[99]	
INT audio	-
INT minutes	-
INT seconds	-
INT subframes	-

5.3 Notes

Template:Globals

6 Dump type

Up to Global Variables

6.1 Definition

INT dump_type = PARTIAL_DUMP

Dump_type is a global variable, holding the current dump_mode. The mode can be changed by assigning a different mode to the variable. Default is PARTIAL_DUMP.

See also restore_type.

Template:Globals

7 Exit status

Up to Global Variables

7.1 Definition

INT exit_status

exit_status is a predefined global variable, holding whether or not Bennu received a QUIT event this frame. A QUIT event is generated for example by pressing the X button of the window.

Value	*Description*
false	There is no QUIT event.
true	There is a QUIT event.

Template:Globals

8 Fading

Up to Global Variables

8.1 Definition

INT fading = false

Fading is a global variable, holding whether the screen is currently fading. This can be caused by the functions fade(), fade_on() or fade_off(). Its value will be true if there is fading going on and false if not.

8.2 Example

```
Program faders;
Private
    int text_id;
Begin

    // Write something
    text_id = write(0,160,100,4,"Look at this fading text!");

    // Fade screen on and off
    fade_off_and_on();

    // Wait for ESC key
    Repeat
        frame;
    Until(key(_ESC))

    // Kill all other processes and clear up text
    let_me_alone();
    delete_text(text_id);

End

Process fade_off_and_on()
Begin
    Loop
        fade(0,0,0,4); // Fade to black
        while(fading) frame; end // Wait for the fading to finish
        fade(100,100,100,4); // Fade to normal
        while(fading) frame; end // Wait for the fading to finish
    End
End
```

Used in example: write(), key(), let_me_alone(), delete_text(), fade()

Template:Globals

9 Father

Up to Local Variables

9.1 Definition

INT father

Father is a predefined local variable. **Father** holds the ProcessID of the process/function that called the current process/function. There are several other local variables which refer to the ProcessID of a related process:

- Son
- Bigbro
- Smallbro

9.2 Example

```
Program example;
Begin
    first_process();
    Loop
        frame;
    End
End

Process first_process()  // Declaration of the first process
Begin
    second_process();  // Call the second process
    Loop
        frame; // This loop makes "first_process()" a process rather than a function
    End
End

Process second_process()  //declaration of another process
Begin
    x = father.x; // These two lines use the father variable to move this process
    y = father.y; // to the position of first_process, as the father variable here
                  // holds the ProcessID of "first_process()"
    Loop
        frame;
    End
End
```

Used in example: process, function, processID

Template:Locals

10 Fileinfo

Up to Global Variables

10.1 Definition

Struct Fileinfo

Fileinfo is a global variable struct, containing information about a file/directory entry, lastly returned by glob().

10.2 Members

Member name	- *Description*
STRING path	- The path to the file/directory (without the name of the file/directory).
STRING name	- The name of the file/directory.
INT directory	- true/false: whether the file/directory is a directory or not
INT hidden	- true/false: whether the file is hidden or not
INT readonly	- true/false: whether the file is read only or not
INT size	- The size of the file/directory.
STRING created	- The date when the file/directory was created. *
STRING modified	- The date when the file/directory was last modified. *

* - The strings *created* and *modified* are in the format: `DD/MM/YYYY hh:mm`.

Template:Moduledocbox Template:Globals

11 Flags

Up to Local Variables

11.1 Definition

INT flags = 0

Flags is a predefined local variable which is used to manipulate how the graphic of a process, assigned to its local variable graph, is displayed.

To alter the effect, change the value of this local variable by assigning it blit flags. Like most bit flags, constants can be added together to combine effects. A horizontally mirrored translucent graphic would need flags B_TRANSLUCENT (4) and B_HMIRROR (1), so flags = B_TRANSLUCENT|B_HMIRROR (5) will do the trick.

11.2 Example

To make the graphic of a process spin:

```
Program mirror
Begin
    mirror_graphic();
    Loop
        frame;
    End
End
Process mirror_graphic()
Begin
    graph = new_map(50,50,8);
    map_clear(0,graph,rgb(0,255,255));
    x = 100;     //Position the graphic 100 pixels
    y = 100;     //from the top and left of the screen
    Loop
        if (key(_l))
            flags = B_HMIRROR; //if you press the L key, your graphic is horizontally mirrored
        else
            flags = 0;
        end
        frame;
    End
End
```

The process will mirror its graphic when the key L is held down.

Template:Locals

12 Focus status

Up to Global Variables

12.1 Definition

INT focus_status

focus_status is a predefined global variable, holding whether or not the Bennu window has input focus.

Value	*- Description*
false	- The Bennu window does not have input focus.
true	- The Bennu window has input focus.

Template:Globals

13 Fps

Up to Global Variables

13.1 Definition

INT fps

The global variable **fps** holds the current frames per second on which Bennu is running. This means how many times a frame is executed every second and how many times the screen is updated every second.

If a different FPS is needed, the target FPS can be altered by use of set_fps().

If a more accurate FPS is needed, use frame_time to calculate it.

Template:Globals

14 Frame time

Up to Global Variables

14.1 Definition

FLOAT frame_time

Frame_time is a global variable, holding the time passed the last frame. In other words: the difference in time between the start of the last frame and the current frame.

Doing a bit of physics, we see that:

FPS = 1 / frame_time

Be advised that frame_time is in milliseconds accurate, so it can be 0 at times, so one might prevent such a case from happening:

FPS = 1 / (frame_time + (frame_time==0)*0.0001);

This gives a sort of FPS which is accurate every frame.

14.2 Example

Display how long the program has been running, by use of frame_time:

```
Program timers;
Private
    int ft; // Help variable
    int i;  // how long the program has been running in 1/100s
    float f; // how long the program has been running in 1/100s
Begin

    set_mode(320,400,8);
    write_int   (0,0,300,0,&timer);
    write_int   (0,0,310,0,&i);
    write_float (0,0,320,0,&f);

    Repeat

        // Calculate how long the program has been running in 1/100s without a float
        ft %= 10; // keep the milliseconds from last time
        ft += frame_time*1000; // add the last passed time to it in milliseconds
        i += ft/10; // add it to the integer without the milliseconds

        // Calculate how long the program has been running in 1/100s with a float
        f+=frame_time*100;

        frame;
    Until(key(_ESC))

End
```

Used in example: set_mode(), write_int(), write_float(), key(), timer

Let a process wait for a certain time by calling this function:

```
Function int wait(float seconds)
Begin
    While( (seconds-=frame_time) > 0 ) frame; End
    return -seconds;
End
```

This can be done with a timer too, as is displayed here.

Template:Globals

15 Full screen

Up to Global Variables

INT full_screen = false

15.1 Description

full_screen is a predefined global variable. It defines whether Bennu (libvideo to be more precise) will be rendered in windowed (default) or fullscreen mode.

The value of full_screen can be changed. false for windowed mode and true for fullscreen mode. For the change to have effect, set_mode() needs to be called afterwards.

15.2 See also

- graph_mode
- scale_mode

Template:Globals

16 Graph

Up to Local Variables

16.1 Definition

INT graph = 0

Graph is a predefined local variable which holds the GraphID of the process. A graphic can be assigned to the process by assigning the GraphID of that graphic to graph. Assign 0 to the local variable graph to have the process display no graph. Keep in mind that this doesn't free the memory used by the graphic; to free it, use unload_map().

16.2 Example

```
Process cyan_graphic()
Begin
    graph = new_map(100,100,8);   // create a new 100x100x8 map.
    map_clear(0,graph,rgb(0,255,255));   // clear it cyan-colored
    x = 100;     //Position the graphic 100 pixels
    y = 100;     //from the top and left of the screen
    Repeat
        frame;
    Until(key(_ESC))
End
```

Used in example: new_map(), map_clear(), graphic, x, y

Template:Locals

17 Graph mode

Up to Global Variables

17.1 Definition

INT graph_mode

Graph_mode is a global variable, holding the current graph mode. The mode can be changed by assigning a different mode to the variable. Default is 0 and after a call to set_mode() it reflects the set graph mode.

17.2 See also

- sound_mode
- scale_mode
- full_screen

Template:Globals

18 Id

Up to Local Variables

18.1 Definition

INT id

id is a predefined local variable. It contains the process' processID.

Template:Locals

19 Local:File

Up to Local Variables

19.1 Description

INT file = 0

File contains the FileID of the file used to obtain the graphic indicated by the local variable GraphID.

19.2 Example

```
import "mod_map"
import "mod_grproc"
import "mod_key"
import "mod_wm"

Global
    int file_id;
End

Process Main()
Begin

    // Load FPG
    file_id = load_fpg("example.fpg");

    // Set locals for display of graph
    file = file_id;
    graph = 1;
    x = y = 50;

    Repeat
        frame;
    Until(key(_ESC)||exit_status)

End
```

Used in example: load_fpg(), key(), x, y, file, graph

Note: nothing will be seen unless you have an FPG "example.fpg" with a graphic with ID 1.

Template:Locals

20 Mouse

Up to Global Variables

20.1 Definition

Struct Mouse

Mouse is a global variable struct, containing information about the current state of the mouse. Also graphical settings can be configured to display a graphic following the mouse in a certain way.

20.2 Members

Member name	*- Description*
INT x	- The X-coordinate of the mouse.
INT y	- The Y-coordinate of the mouse.
INT graph	- The graphID of the graphic of the mouse.
INT file	- The fileID of the file in which the graphic of the mouse is located.
INT z	- The Z-coordinate of the mouse.
INT angle	- The angle of the mouse process.
INT size	- The size of the mouse process.
INT flags	- The render flags of the mouse process.
INT region	- The region of the mouse process.
INT left	- true/false: whether the left mouse button is pressed.
INT middle	- true/false: whether the middle mouse button is pressed.
INT right	- true/false: whether the right mouse button is pressed.
INT wheelup	- true/false: whether the mouse wheel is scrolled upwards.
INT wheeldown	- true/false: whether the mouse wheel is scrolled downwards.

20.3 Example

```
import "mod_map"
import "mod_mouse"
import "mod_wm"

Process Main()
Private
    int dmap;
    int rmap;
    int gmap;
    int bmap;
    int ymap;
    int cmap;
    int mmap;
    int wmap;
Begin

    // Create a dark graph
    dmap = new_map(100,100,8);
    map_clear(0,dmap,rgb(50,50,50));

    // Create a red graph
    rmap = new_map(100,100,8);
    map_clear(0,rmap,rgb(255,0,0));

    // Create a green graph
    gmap = new_map(100,100,8);
    map_clear(0,gmap,rgb(0,255,0));

    // Create a blue graph
    bmap = new_map(100,100,8);
    map_clear(0,bmap,rgb(0,0,255));

    // Create a yellow graph
    ymap = new_map(100,100,8);
    map_clear(0,ymap,rgb(255,255,0));

    // Create a cyan graph
    cmap = new_map(100,100,8);
    map_clear(0,cmap,rgb(0,255,255));
```

```
    // Create a magenta graph
    mmap = new_map(100,100,8);
    map_clear(0,mmap,rgb(255,0,255));

    // Create a white graph
    wmap = new_map(100,100,8);
    map_clear(0,wmap,rgb(255,255,255));

    Repeat
        if(mouse.left) // +Red
            if(mouse.right) // +Green
                if(mouse.middle) // +Blue
                    mouse.graph = wmap;
                else
                    mouse.graph = ymap;
                end
            else
                if(mouse.middle) // +Blue
                    mouse.graph = mmap;
                else
                    mouse.graph = rmap;
                end
            end
        elseif(mouse.right) // +Green
            if(mouse.middle) // +Blue
                mouse.graph = cmap;
            else
                mouse.graph = gmap;
            end
        elseif(mouse.middle) // +Blue
            mouse.graph = bmap;
        else // Dark
            mouse.graph = dmap;
        end
        frame;
    Until(exit_status)

End
```

Used in example: **Mouse**, new_map(), map_clear(), graph, exit_status

Shows some use of maps and the mouse.

Template:Image

Template:Globals

21 Mouse status

Up to Global Variables

21.1 Definition

INT mouse_status

mouse_status is a predefined global variable, holding whether or not the mouse cursor is inside the Bennu window.

Value	*- Description*
false	- The mouse cursor is outside the Bennu window.
true	- The mouse cursor is inside the Bennu window.

Template:Globals

22 Os id

Up to Global Variables

22.1 Definition

INT os_id

os_id is a predefined global variable, holding the OS code of the operating system Bennu is currently running on while executing. Default is *-1*.

Template:Globals

23 Priority

Up to Local Variables

23.1 Definition

INT priority = 0

priority is a predefined local variable, holding the priority of the process; default is 0.

Using this priority, the order of process-execution can be influenced, because processes with a higher priority are executed before processes with a lower priority.

23.2 Example

```
import "mod_say"
import "mod_proc"
import "mod_timers"

Process A()
Begin
    priority = 0; // Default
    Loop
        say("[" + timer[0] + "] " + "A");
        frame(100000000); // This is very high because Bennu runs without limiter here
    End
End

Process main()
Private
    int f=0;
Begin
    priority = 1; // Set to higher priority than A
    A();
    Repeat
        f++;
        say("[" + timer[0] + "] " + "Main");
        frame(100000000); // This is very high because Bennu runs without limiter here
    Until(f==5)
OnExit
    let_me_alone();
End
```

Used in example: say(), let_me_alone(), timer, **priority**, frame

Possible output:

```
[0] A
[0] Main
[26] Main
[26] A
[50] Main
[50] A
[74] Main
[74] A
[98] Main
[98] A
```

It can be seen here that regardless of priority, `A` is first executed, because `A` is called by `Main`. As soon as `A` reaches its frame statement, `Main` continues until its frame statement and this concludes the first frame. The second frame it is shown that the priority has effect: `Main` is executed before `A`.

Template:Locals

24 Region

Up to Local Variables

24.1 Definition

24.1.1 Local variable

INT region = 0

Region is a predefined local variable. **Region** holds the RegionID of the **region** in which the process' graphic should only be displayed in. By default this is region *0*, the whole screen.

The graphic of the process is only displayed in its region, even if the x and y coordinates are outside of the region, the part inside the region will still be displayed.

24.1.2 Concept

A region is a rectangular field inside the screen. It can be defined with define_region() and can be useful for displaying graphics in only certain parts of the screen and for the function region_out(). There are 32 regions (*0..31*) and region *0* is the whole screen.

Template:Locals

25 Reserved

Up to Local Variables

25.1 Definition

Struct Reserved

Reserved is a local variable struct, containing information that is reserved for Bennu's internals. However, sometimes you may wish to use certain members of it. Using in the sense of reading only, **do not under any circumstances alter their values, unless you know what you are doing**.

If it's not documented what these members do, their use is reasonably limited. If you want to know what these members do, your programming goals are of a level, where you can also look in the sourcecode. This is because most of these members would require a lengthy explanation about what they do and why.

25.2 Members

Member name	- *Description*
INT ID_scan	- Internal use only (formerly got processID from within the process).
INT process_type	- The ProcessTypeID of the process.
INT type_scan	- Internal use only (formerly got processTypeID from within the process).
INT status	- The status of the process, containing a status code.
INT changed	- Internal use only.
INT xgraph_flags	- Internal use only (blit flags for xgraph).
INT saved_status	- Internal use only (for signals).
INT prev_z	- Previous z value.
INT distance1	- Not used.
INT distance2	- Not used.
INT frame_percent	- Internal use only.
INT box_x0	- The x-coordinate of the topleft corner of the process' graphic (process.x-graphic.width/2).
INT box_y0	- The y-coordinate of the topleft corner of the process' graphic (process.y-graphic.height/2).
INT box_x1	- The x-coordinate of the bottomright corner of the process' graphic (process.x+graphic.width/2).
INT box_y1	- The y-coordinate of the bottomright corner of the process' graphic (process.y+graphic.height/2).

Template:Globals

26 Resolution

Up to Local Variables

26.1 Definition

26.1.1 Local variable

INT resolution = 0

Resolution is used to alter the precision of the position of processes on screen; the level of precision is defined by the value of resolution.

This simulating of fractions in positions is useful for calculations or animations in need of a high precision in order to work properly. It causes the coordinates of all processes to be interpreted as being multiplied by the value of the local variable resolution, associated with that process. So when a process' graphic is displayed, it will appear as if the process' x and y values were divided by the value of resolution. A resolution of `0` is interpreted as if it were `1`.

The default value of **resolution** is 0, so set it to 1 if the correct value is needed.

26.1.2 Screen Resolution

The resolution of a screen is the dimensions of the screen in pixels. Bennu's default screen resolution is 320×200 pixels. This can be altered by use of set_mode().

26.2 Example

```
import "mod_grproc"
import "mod_time"
import "mod_key"
import "mod_video"
import "mod_map"
import "mod_draw"
import "mod_proc"
import "mod_wm"

Process Main()
Begin

    // Set screen resolution to 320x200 with a color depth of 8bit
    set_mode(320,200,8);

    // Set the FPS to 60
    set_fps(60,0);

    // Set resolution for this process (try changing it to see the effect)
    resolution = 100;

    // Create a 200x200 cyan circle and assign its graphID to the local variable graph
    graph = map_new(200,200,8);
    drawing_map(0,graph);
    drawing_color(rgb(0,255,255));
    draw_fcircle(100,100,99);

    // Set size
    size = 10;

    // Set the coordinates at screen position (160,180).
    x = 160 * resolution;
    y = 180 * resolution;

    // Move around in circles while leaving a trail behind
    Repeat
        trail(x,y,graph,(int)(0.2*size),get_timer()+1000); // create a mini reflection of this process,
                                                           // lasting one second
        advance(3*resolution); // advance (3 * resolution) units (= 3 pixels)
        angle+=2000; // turn 2 degrees left
        frame;
    Until(key(_ESC)||exit_status)

OnExit

    let_me_alone();
    map_unload(0,graph);

End
```

```
Process trail(x,y,graph,size,endtime)
Begin

    // Get the resolution of the process calling this one
    resolution = father.resolution;

    // Remain existent until the specified endtime was reached
    Repeat
        frame;
    Until(get_timer()>=endtime)

End
```

Used in example: set_mode(), set_fps(), map_new(), drawing_map(), drawing_color(), draw_fcircle(), get_timer(), key(), let_me_alone(), map_unload(), advance(), **resolution**, graph, size, x, y, angle, exit_status

Here are a few screenshots with different resolutions to display the effect it can have.

Template:Image

Template:Image

Template:Image

Template:Image

The effect is clearly visible, so when you are moving processes with graphics around the screen, you might want to consider using a resolution of at least 10 in those processes.

Template:Locals

27 Restore type

Up to Global Variables

27.1 Definition

INT restore_type = PARTIAL_RESTORE

Restore_type is a global variable, holding the current restore_mode. The mode can be changed by assigning a different mode to the variable. Default is PARTIAL_RESTORE.

See also dump_type.

Template:Globals

28 Scale mode

Up to Global Variables

INT scale_mode = SCALE_NONE

28.1 Description

Scale_mode is a global variable, holding the current or to be scale mode. The mode can be changed by assigning a different mode to the variable and then calling set_mode(). Default is SCALE_NONE.

28.2 See also

- sound_mode
- graph_mode

Template:Globals

29 Scroll

Up to Global Variables

29.1 Definition

STRUCT[9] Scroll

Scroll is a global variable struct array, containing information about the current state of the ten available scrolls (0..9). It is used to influence the settings of a scroll window, initiated by start_scroll().

29.2 Members of one Struct

Member name	*Description*
INT x0	- The X-coordinate of the foreground graphic.*
INT y0	- The Y-coordinate of the foreground graphic.*
INT x1	- The X-coordinate of the the background graphic.*
INT y1	- The Y-coordinate of the the background graphic.*
INT z	- The Z-coordinate of the scroll window (default: 512).**
INT camera	- The processID of the process to be followed (default: 0).***
INT ratio	- The ratio in percent the foreground will move related to the background (default: 200).***,****
INT speed	- The maximum speed of the foreground, 0 for no limit (default: 0).***
INT region1	- The region in which the scroll won't move, -1 for none (default: -1).***
INT region2	- The region in which the maximum speed is ignored, -1 for none (default: -1).***
INT flags1	- The bit flags for the foreground graphic.
INT flags2	- The bit flags for the background graphic.
INT follow	- The scrollID of the scroll window to follow, -1 means none (default: -1).
INT[6] reserved	- Seven reserved integers.

* - These fields become *read only* when the scroll window is in automatic mode (see notes).

** - Processes on the scroll use this z value (see notes).

*** - These fields only make sense when the scroll window is in automatic mode (see notes).

**** - The ratio is in percent: 200 means the background moves twice as slow.

29.3 Notes

There are two main methods of controlling a scroll window. One is the manual way, thus changing the values of the x0,y0,x1 and y1 manually. This mode is used when the *camera* field is not set (is 0). When it is set with a processID, the scroll window is in automatic mode and will follow that process' coordinates and will try to keep it centered. This can be influenced by other members of the scroll struct.

The foreground is the plane to be controlled and the background moves relative to the foreground.

Processes in a scroll take over the z value of that scroll's z. Between processes on the same scroll the local variable z of that process does have the expected effect.

29.4 Example

Template:Globals

30 Size

Up to Local Variables

30.1 Definition

INT size = 100

Size is a predefined local variable that can be used to stretch or compress a graphic, equally along the horizontal axis and vertical axis. It is defined as a percentage of the original graphic size. The graphics's center will remain at the drawing coordinates when the graphic is drawn.

This variable only has effect for the appearance of a process' graphic when its local variables size_x and size_y are both `100`. When either is not equal to `100`, **size** doesn't affect the appearance of the graphic.

30.2 Example

To make the graphic of a process continually stretch:

```
Process Main()
Begin
    graph = new_map(50,50,8); // Create a new graphic
    x = 100;                  // Position the graphic 100 pixels
    y = 100;                  // from the top and left of the screen
    Loop
        size += 1;            // Increase the height and width of the graphic by 1 percent each frame.
        frame;
    End
OnExit
    unload_map(0,graph);
End
```

Used in example: new_map(), x, y, **size**, unload_map()

30.3 See also

- size_x
- size_y

Template:Locals

31 Size x

Up to Local Variables

31.1 Definition

INT size_x = 100

Size_x is a predefined local variable that can be used to stretch or compress a graphic along its horizontal axis. It is defined as a percentage of the original graphic size. The graphics's center will remain at the drawing coordinates when the graphic is drawn.

When either **size_x** or size_y of a process are unequal to [code]100[/code], that process' size has no effect.

31.2 Example

To make the graphic of a process continually stretch horizontally:

```
Process Main()
Begin
    graph = new_map(50,50,8); // Create a new graphic
    x = 100;                  // Position the graphic 100 pixels
    y = 100;                  // from the top and left of the screen
    Loop
        size_x += 1;          // Increase the width of the graphic by 1 percent each frame.
        frame;
    End
OnExit
    unload_map(0,graph);
End
```

Used in example: new_map(), [[x], y, **size_x**, unload_map()

31.3 See also

- size
- size_y

Template:Locals

32 Size y

Up to Local Variables

32.1 Definition

INT size_y = 100

Size_y is a predefined local variable that can be used to stretch or compress a graphic along its vertical axis. It is defined as a percentage of the original graphic size. The graphics's center will remain at the drawing coordinates when the graphic is drawn.

When either size_x or **size_y** of a process are unequal to [code]100[/code], that process' size has no effect.

32.2 Example

To make the graphic of a process continually stretch vertically:

```
Process Main()
Begin
    graph = new_map(50,50,8); // Create a new graphic
    x = 100;                  // Position the graphic 100 pixels
    y = 100;                  // from the top and left of the screen
    Loop
        size_y += 1;          // Increase the height of the graphic by 1 pixel each frame.
        frame;
    End
OnExit
    unload_map(0,graph);
End
```

Used in example: new_map(), [[x], y, **size_y**, unload_map()

32.3 See also

- size
- size_x

Template:Locals

33 Smallbro

Up to Local Variables

33.1 Definition

INT smallbro

Smallbro is a predefined local variable. **Smallbro** holds the ProcessID of the process/function that was immediately called after by the father of the current process/function. There are several other local variables which refer to the ProcessID of a related process:

- Father
- Son
- Bigbro

Template:Locals

34 Son

Up to Local Variables

34.1 Definition

INT son

Son is a predefined local variable. **Son** holds the ProcessID of the process/function that was last called by the current process/function. There are several other local variables which refer to the ProcessID of a related process:

- Father
- Bigbro
- Smallbro

Template:Locals

35 Sound channels

Up to Global Variables

35.1 Definition

INT sound_channels = 8

Sound_channels is a global variable, holding the number of sound channels, which is set when sound is used for the first time, meaning altering the value of this variable will have no effect after sound has been initialized. This number can range from 1 to 32 and the default is 8.

35.2 See also

- sound_mode
- sound_freq

Template:Globals

36 Sound freq

Up to Global Variables

36.1 Definition

INT sound_freq = 22050

Sound_freq is a global variable, holding the set sound frequency, which is set when sound is used for the first time, meaning altering the value of this variable will have no effect after sound has been initialized. The higher the frequency, the higher the quality is. Accepted frequencies are:

- 44100: high quality (recommended)
- 22050: medium quality (default)
- 11025: low quality (not recommended)

36.2 See also

- sound_mode
- sound_channels

Template:Globals

37 Sound mode

Up to Global Variables

37.1 Definition

INT sound_mode = MODE_STEREO

Sound_mode is a global variable, holding the set sound mode, which is set when sound is used for the first time, meaning altering the value of this variable will have no effect after sound has been initialized. The mode can be changed by assigning a different mode to the variable. Default is MODE_STEREO.

37.2 See also

- graph_mode
- sound_freq
- sound_channels

Template:Globals

38 Text flags

Up to Global Variables

38.1 Definition

INT text_flags = 0

Text_flags is a global variable. When a dynamic text is created (write(), etc), its flags value will equal the value of **text_flags** at the moment of creation. The default value is 0.

38.2 See also

- text_z
- TextID

Template:Globals

39 Text z

Up to Global Variables

39.1 Definition

INT text_z = -256

Text_z is a global variable. When a dynamic text is created (write(), etc), its z value will equal the value of **text_z** at the moment of creation. The default value is -256.

39.2 See also

- z
- text_flags
- TextID

Template:Globals

40 Timer

Up to Global Variables

40.1 Definition

INT[9] timer

Timer is a global variable, holding ten integers. Each frame a certain value is added to all of them. This value is the difference in time between the start of the last frame and the current frame, in 1/100 seconds.

So when all the timers are never altered, their values will be 1234 when the program has been running for about 12.34 seconds.

40.2 Examples

40.2.1 Display how long the program has been running

```
import "mod_timers"
import "mod_key"
import "mod_text"

Process Main()
Begin

    write_int(0,0,100,0,&timer[0]);

    Repeat
        frame;
    Until(key(_ESC))

End
```

Used in example: write_int(), key(), **timer**

This can be done more accurately with the use of frame_time, which is in milliseconds.

40.2.2 Let a process wait for a certain time by calling this function

```
Function int wait(int t)
Begin
    t += timer[0];
    While(timer[0]<t) frame; End
    return t-timer[0];
End
```

This can be done without a timer too, as is displayed here.

Template:Globals

41 Window status

Up to Global Variables

41.1 Definition

INT window_status

window_status is a predefined global variable, holding whether or not the Bennu window is visible. For example the window is not visible when it is minimized.

Value	*- Description*
false	- The Bennu window is not visible.
true	- The Bennu window is visible.

Template:Globals

42 X

This is about the local variable. Did you mean the scan code _X?

Up to Local Variables

42.1 Definition

INT x = 0

X is a predefined local variable that defines the vertical position of the process graph in the screen.

42.2 See also

- y
- z

Template:Locals

43 Y

This is about the local variable. Did you mean the scan code _Y?

Up to Local Variables

43.1 Definition

INT y = 0

Y is a predefined local variable that defines the horizontal position of the process graph in the screen.

43.2 See also

- x
- z

Template:Locals

44 Z

This is about the local variable. Did you mean the scan code _Z?

Up to Local Variables

44.1 Definition

INT z = 0

Z is a predefined local variable that defines the depth position of the process graph in the screen. This variable affects what process graph will be drawn before other one. A process with higher z will be drawn beneath a process with a lower z.

44.2 See also

- x
- y

Template:Locals