Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

1 Abs

1.1 Definition

FLOAT abs (<**FLOAT** value>)

Returns the absolute value of *value*.

1.2 Parameters

FLOAT value - The value.

1.3 Returns

FLOAT : The absolute value of *value*.

1.4 Example

```
Global
    float value1;
    int value2;
End

Process Main()
Begin

    write_float(0,0, 0,0,&value1);
    write_int(0,0,10,0,&value2);

    value1 = abs(3);
    value2 = abs(-4);

    Repeat
        frame;
    Until(key(_ESC))

End
```

Used in example: write_float(), write_int(), **abs**(), key()

Template:Funcbox

2 Acos

2.1 Syntax

FLOAT acos (<**FLOAT** value>)

2.2 Description

Returns the arccosine of a certain *value*.

This function performs an arccosine calculation on a certain value and returns an angle between and including 0 and 180000 (0-180º).

2.3 Parameters

FLOAT value - The value to be performed an arccosine calculation on.

2.4 Returns

FLOAT : The arccosine result of the specified value, an angle between and including 0 and 180000 (0-180º).

2.5 Notes

The angle value returned by this function is in thousandths of degrees, as most angles within Bennu are.

To read about all aspects of trigonometry, you can visit Wikipedia's Trigonometric function page.

Template:Moduledocbox

3 Advance

3.1 Definition

INT advance (<**INT** distance>)

Moves the calling process forward by *distance* units in the direction of the process' angle.

This function is influenced by the local variables angle and resolution.

3.2 Parameters

INT distance - Distance to advance in units.

3.3 Returns

INT : Returns true if successful and false if failed.

3.4 Example

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

Process Main()
Private
    int my_proc;
Begin

    proc(); //create a new process
    proc2(); //create a new process

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

OnExit

    signal(my_proc,S_KILL);

End

Process proc()
Begin

    // Create a cyan square and assign it to 'graph'
    graph = map_new(100,100,8);
    map_clear(0,graph,rgb(0,255,255));

    // Set starting position
    x = 50;
    y = 50;

    // This loop makes this process advance 3 pixels every frame
    Loop
        advance(3); // advance 3 pixels
        frame;
    End

End

Process proc2()
Begin

    // Set resolution to 100
    resolution = 100;

    // Create a cyan square and assign it to 'graph'
    graph = map_new(100,100,8);
    map_clear(0,graph,rgb(0,255,255));

    // Set starting position
    x = 50*resolution;
    y = 150*resolution;
```

```
    // This loop makes this process advance 3/100 pixels every frame
    Loop
        advance(3); // advance 3/100 pixels
        frame;
    End

OnExit

    map_unload(0,graph);

End
```

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

Template:Moduledocbox

4 Alloc

4.1 Syntax

VOID POINTER alloc (<**INT** size>)

4.2 Description

Allocates a block of memory of a certain size. Returns a pointer to the newly allocating memory block, or NULL on failure.

4.3 Parameters

INT size - The size of the to be allocated memory in bytes.

4.4 Returns

VOID POINTER : Pointer to the first element of the allocated memory block.

NULL - There was are an error allocating the memory, like insufficient memory available.
!NULL - Pointer to the first element of the allocated memory block.

4.5 Example

```
import "mod_mem"
import "mod_say"

Process Main()
Private
    byte* pbyte;
    word* pword;
    int* pint;
    int elements = 10;
    int i;
Begin

    // Allocate memory
    pbyte = alloc(elements);
    pword = alloc(elements*sizeof(*pword)); // note the sizeof() here, this is possible!
    pint  = alloc(elements*sizeof(int));

    // Reset memory to 0's
    memset (pbyte,0,elements);
    memsetw(pword,0,elements); // same as  memset(pword,0,elements*sizeof(word));
                               // because value-parameter is 0.
    memseti(pint ,0,elements); // same as  memset(pint,0,elements*sizeof(int));
                               // because value-parameter is 0.

    // Write numbers to bytes and ints
    for(i=0; i<elements; i++)
        pbyte[i]  = 133; // pbyte[i] is the same as *(pbyte+i)
        *(pint+i) = 4555; // pint[i] is the same as *(pint+i)
    end

    // Write numbers to words
    memsetw(pword,345,elements);

    // Show numbers
    for(i=0; i<elements; i++)
        say("byte["+i+"] = " + *(pbyte+i));
        say("word["+i+"] = " + pword[i]);
        say("int ["+i+"] = " + pint[i]);
    end

OnExit

    // Free the used memory
    free(pbyte);
    free(pword);
    free(pint);

End
```

Used in example: **alloc**(), memset(), memsetw(), memseti(), sizeof(), say(), free(), OnExit, pointer, sizeof

Template:Moduledocbox

5 Asc

5.1 Definition

BYTE asc (<**STRING** character>)

Returns the ASCII value of the first character of the string *character*.

5.2 Parameters

STRING character - The string of which the ASCII value of the first character will be returned.

5.3 Returns

BYTE : The ASCII value of the first character of the string *character*.

5.4 Example

```
Program asciis;
Begin

    write(0,0, 0,0,asc("A"));
    write(0,0,10,0,asc("CAT"));

    Repeat
        frame;
    Until(key(_esc))

End
```

Used in example: write(), key()

Template:Funcbox

6 Asin

6.1 Syntax

FLOAT asin (<**FLOAT** value>)

6.2 Description

Returns the arcsine of a certain *value*.

This function performs an arcsine calculation on a certain value and returns an angle between and including -90000 and 90000 (-90-90º).

6.3 Parameters

FLOAT value - The value to be performed an arcsine calculation on.

6.4 Returns

FLOAT : The arcsine result of the specified value, an angle between and including -90000 and 90000 (-90-90º).

6.5 Notes

The angle value returned by this function is in thousandths of degrees, as most angles within Bennu are.

To read about all aspects of trigonometry, you can visit Wikipedia's Trigonometric function page.

Template:Moduledocbox

7 Atan

7.1 Syntax

FLOAT atan (<**FLOAT** value>)

7.2 Description

Returns the arctangent of a certain *value*.

This function performs an arctangent calculation on a certain value and returns an angle between but not including -90000 and 90000 (-90-90º).

7.3 Parameters

FLOAT value - The value to be performed an arctangent calculation on.

7.4 Returns

FLOAT : The arctangent result of the specified value, an angle between but not including -90000 and 90000 (-90-90º).

7.5 Notes

The angle value returned by this function is in thousandths of degrees, as most angles within Bennu are.

To read about all aspects of trigonometry, you can visit Wikipedia's Trigonometric function page.

Template:Moduledocbox

8 Atof

8.1 Definition

FLOAT atof (<**STRING** str>)

Returns the float value of the number specified in a certain string.

8.2 Parameters

STRING str - The string specifying a number of which the float value will be returned.

8.3 Returns

FLOAT : The float value of the number specified by the specified string.

Template:Funcbox

9 Atoi

9.1 Definition

INT atoi (<**STRING** str>)

Returns the int value of the number specified in a certain string.

9.2 Parameters

STRING str - The string specifying a number of which the int value will be returned.

9.3 Returns

INT : The int value of the number specified by the specified string.

Template:Funcbox

10 Blendop apply

10.1 Definition

INT blendop_apply (<**INT** fileID> , <**INT** graphID> , <**INT** blendTable>)

Applies a blend operation to the pixels of a graphic, as if it was rendered into a black (color 0) background.

This function actually changes the pixeldata of the graphic.

10.2 Parameters

INT fileID	- The fileID of the file holding the graphic.
INT graphID	- The graphID of the graphic to apply the blend operation to.
INT blendTable	- The blend table to apply.

10.3 Returns

INT : true

Template:Funcbox

11 Blendop assign

11.1 Definition

INT blendop_assign (<**INT** fileID> , <**INT** graphID> , <**INT** blendTable>)

Assigns a blend operation to a graphic. The graphic will be drawn using this blend operation hereafter. Only one blend operation can be assigned to one graphic at a given time; when another is assigned, the previous one is unassigned.

To deassigning a blend table, using NULL as the *blendopID*.

11.2 Parameters

INT fileID	- The fileID of the file holding the graphic.
INT graphID	- The graphID of the graphic to assign the blendop to.
INT blendTable	- The blend table to assign (NULL for none).

11.3 Returns

INT : true

Template:Funcbox

12 Blendop free

12.1 Definition

INT blendop_free (<**INT** blendTable>)

Frees the given blend table. Before doing this, make sure it is not assigned to a graphic.

12.2 Parameters

INT blendTable - The blend table to free.

12.3 Returns

INT : true

Template:Funcbox

13 Blendop grayscale

13.1 Definition

INT blendop_grayscale (<**INT** blendTable> , <**INT** mode>)

Modify the blend table by modifying the colours, so the compononts of one colour is the same (this makes them appear gray). This means that the graphic the blend operation is assigned to will appear gray.

The source section of the blend table will be modified; this will clear the destination section of the blend table.

13.2 Parameters

INT blendTable	- The blend table to modify.
INT mode	- The mode to perform the grayscaling (see notes).

13.3 Returns

INT : true

13.4 Notes

- Method 1

```
component = 0.3*r+0.59*g+0.11*b colour = rgb ( component , component , component )
```

- Method 2

```
max = r > g ? r > b ? r : g : g > b ? g : b ; min = r < g ? r < b ? r : g : g < b ? g : b ; component = ( max + min ) / 2 colour = rgb ( component , component , component )
```

- Method 3

```
max = r > g ? r > b ? r : g : g > b ? g : b ; colour = rgb ( max , max , max )
```

Template:Funcbox

14 Blendop identity

14.1 Definition

INT blendop_identity (<**INT** blendopTable>)

Reinitializes the blend table to default.

The source section of the blend table will be the normal object and the destination section will be cleared, removing translucency.

14.2 Parameters

INT blendTable - The blend table to reinitialize.

14.3 Returns

INT : true

Template:Funcbox

15 Blendop intensity

15.1 Definition

INT blendop_intensity (<**INT** blendTable> , <**FLOAT** amount>)

Modify the blend table by intensifying the colours. This means that the graphic the blend operation is assigned to will appear darker or lighter than normal.

This will modify the source section of the blend table, but leave the destination section as it was.

15.2 Parameters

INT blendTable	- The blend table to intensify.
FLOAT amount	- The amount to multiply the R,G,B values of the colours by.

15.3 Returns

INT : true

Template:Funcbox

16 Blendop new

16.1 Definition

INT blendop_new ( )

Creates a new blend table. This table will contain a blending effect, which you'll have to set afterwards, using the various functions. When that is done you can finally apply or assign the blend table to a graphic.

The source section of the blend table will be the normal object and the destination section will be cleared, removing translucency.

16.2 Returns

0	- Error: insufficient memory or the screen was not yet initialized.
!0	- Success (pointer to the blend table).

16.3 Notes

The right order of doing blending stuff: First create a new table with **blendop_new**(), then put a blending effect in it with for example blendop_tint(), and then assign it to a graphic with blendop_assign().

Blendops are not supported in 32bit mode.

16.4 Errors

Insufficient memory	- There is insufficient memory available. This error doesn't occur often.

16.5 Example

```
Program test;
Private
    int blendTable;
Begin

    set_mode(320,240,16);

    x = 160;
    y = 120;

    graph = new_map(100,100,16);
    map_clear(0,graph,RGB(255,255,255));

    blendTable = blendop_new();
    blendop_tint(blendTable,1,255,0,0);

    Repeat
        if (key(_space))
            blendop_assign(0,graph,blendTable);
        else
            blendop_assign(0,graph,NULL);
        end
        frame;
    Until(key(_ESC))

OnExit

    unload_map(0,graph);

End
```

Used in example: set_mode(), new_map(), map_clear(), **blendop_new**(), blendop_tint(), key(), blendop_assign(), unload_map()

This will result in something like:

File:Blendop.jpg

Template:Funcbox

17 Blendop swap

17.1 Definition

INT blendop_swap (<**INT** blendTable>)

Modify the blend table by swapping the source and destination sections. This means that the graphic the blend operation is assigned to will appear like what is behind the object and the background will look like the object.

This will swap the source and destination section of the blend table.

17.2 Parameters

INT blendTable - The blend table to swap the sections of.

17.3 Returns

INT : true

17.4 Notes

To understand what swapping actually does, consider the following:

- When done a blendop_swap() on a just initialized blend table, the graphic it is assigned to will 'disappear'.

Template:Funcbox

18 Blendop tint

18.1 Definition

INT blendop_tint (<**INT** blendTable> , <**FLOAT** amount> , <**BYTE** r> , <**BYTE** g> , <**BYTE** b>)

Modify the blend table by tinting the colours with the specified colour. This means that the graphic the blend operation is assigned to will appear more like the specified colour, depending on the amount.

This will modify the source section of the blend table, but leave the destination section as it was.

18.2 Parameters

INT blendTable	- The blend table to tint.
FLOAT amount	- The amount to use the specified colour (0-1).
BYTE r	- The red component of the colour to be used for the tinting.
BYTE g	- The green component of the colour to be used for the tinting.
BYTE b	- The blue component of the colour to be used for the tinting.

18.3 Returns

INT : true

Template:Funcbox

18.4 Notes

To further clarify what is done to the assigned graphic, there is this formula: `VisibleColour = graphicColour * (1-amount) + specifiedColour * amount` So it is clear that the result will be a mix of the specified colour and the original graph.

19 Blendop translucency

19.1 Definition

INT blendop_translucency (<**INT** blendTable> , <**FLOAT** amount>)

Modify the blend table by setting how much the object is visible and how much what is behind the source. This means that the graphic the blend operation is assigned to will appear translucent or transparent.

This will modify both the source and destination section of the blend table. The source is multiplied by *amount* and the destination is multiplied by *1-amount*.

19.2 Parameters

INT blendTable	- The blend table to modify.
FLOAT amount	- Opacity factor (1 (opaque) - 0 (transparent)).

19.3 Returns

INT : true

19.4 Notes

To set an amount of 0.5 is the same as doing using a blit flag of B_TRANSLUCENT.

Template:Funcbox

20 Blur

20.1 Definition

INT blur (<**INT** fileID> , <**INT** graphID> , <**BYTE** mode>)

This will make the specified graphic blurry by using a specified mode. It manipulates the graphic directly.

Only 16bit graphics are supported on 16bit video mode.

20.2 Parameters

INT fileID	- The fileID of the file that holds the graphics.
INT graphID	- The graphID of the graphic to convert.
BYTE mode	- The blur mode that is applied to the graphic (see below).

20.3 Returns

INT

0 - Invalid graphic or non 16bit graphic used.
1 - Success.

20.4 Blur modes

Constant	- *Value*	- *Description*
BLUR_NORMAL	- 0	- single pixel: considers pixels located to the left and above of each pixel.
BLUR_3X3	- 1	- 3x3: considers all 8 adjacent pixels.
BLUR_5X5	- 2	- 5x5: considers the 24 pixels that surrounds each pixel.
BLUR_5X5_MAP	- 3	- 5x5 with additional map: Just as the previous one but using a temporary map.

20.5 Examples

Template:Image

Template:Moduledocbox

21 Cd

21.1 Definition

STRING cd ([<**STRING** folder>])

Sets the current path of execution if *folder* was specified and returns it.

Note that it is **highly recommended** to use chdir() for changing the current path of execution, as cd() will make Bennu crash when a folder is specified and the returned path of execution is used in the Bennu program. Just using cd() without a folder is not a problem.

21.2 Parameters

STRING folder - The folder to be entered from the current path of execution or a new path of execution.

21.3 Returns

STRING : The current path of execution.

21.4 Example

```
import "mod_dir"
import "mod_say"
import "mod_key"

Process Main()
Begin

    say(cd());
    say(chdir(".."));
    say(cd());

    Repeat
        frame;
    Until(key(_ESC))

End
```

Used in example: **cd**(), chdir(), say(), key()

Template:Moduledocbox

22 Cd drives

22.1 Definition

INT cd_drives ( )

Returns the number of CD/DVD drives in the system.

22.2 Returns

INT : The number of CD/DVD drives in the system.

Template:Moduledocbox

23 Cd eject

23.1 Definition

INT cd_eject (<**INT** cdID>)

Ejects the CD/DVD (tray) from the drive.

23.2 Parameters

INT cdID - The cdID of the CD/DVD drive of which the CD/DVD (tray) is to be ejected.

23.3 Returns

INT

0 - Invalid CD/DVD drive.
1 - The CD/DVD (tray) is ejected.
Template:Moduledocbox

24 Cd getinfo

24.1 Definition

INT cd_getinfo (<**INT** cdID>)

Fills the global structure cdinfo with information about the specified CD/DVD drive.

Returns `1` if there is a valid CD in the drive and `0` otherwise.

24.2 Parameters

INT cdID - The cdID of the CD/DVD drive of which the info is wanted.

24.3 Returns

INT

0 - Invalid CD/DVD or no CD/DVD in drive.
1 - Valid CD (structure cdinfo is filled successfully with information about the CD).
Template:Moduledocbox

25 Cd name

25.1 Definition

STRING cd_name (<**INT** cdID>)

Returns the current name of the specified CD/DVD drive.

25.2 Parameters

INT cdID - The cdID of the CD/DVD drive of which the name is wanted.

25.3 Returns

STRING : The current name of the specified CD/DVD drive.

Template:Moduledocbox

26 Cd pause

26.1 Definition

INT cd_pause (<**INT** cdID>)

Pauses playback of the specified CD/DVD drive.

26.2 Parameters

INT cdID - The cdID of the CD/DVD drive to pause playback on.

26.3 Returns

INT

0 - Invalid CD/DVD drive.
1 - Playback of the CD/DVD drive is paused.
Template:Moduledocbox

27 Cd play

27.1 Definition

INT cd_play (<**INT** cdID> , <**INT** trackNumber> , [<**INT** tracks>])

Starts playing at the specified track on the specified CD/DVD drive and will keep on playing for the specified number of tracks or one if no number is specified.

27.2 Parameters

INT cdID	- The cdID of the CD/DVD drive with the CD to play.
INT trackNumber	- The track number to play.
INT tracks	- The number of tracks to play. Default is *1*.

27.3 Returns

INT

0 - Invalid CD/DVD drive or CD/DVD.

1 - The CD started playing.

Template:Moduledocbox

28 Cd resume

28.1 Definition

INT cd_resume (<**INT** cdID>)

Resumes playback of the specified CD/DVD drive.

28.2 Parameters

INT cdID - The cdID of the CD/DVD drive to resume playback on.

28.3 Returns

INT

0 - Invalid CD/DVD drive.
1 - Playback of the CD/DVD drive is resumed.
Template:Moduledocbox

29 Cd status

29.1 Definition

INT cd_status (<**INT** cdID>)

Returns the current status of the specified CD/DVD drive.

See CD statuscodes.

29.2 Parameters

INT cdID - The cdID of the CD/DVD drive of which the status is wanted.

29.3 Returns

INT : The current status of the specified CD/DVD drive.

Template:Moduledocbox

30 Cd stop

30.1 Definition

INT cd_stop (<**INT** cdID>)

Stops playback of the specified CD/DVD drive.

30.2 Parameters

INT cdID - The cdID of the CD/DVD drive to stop playback on.

30.3 Returns

INT

0 - Invalid CD/DVD drive.
1 - Playback of the CD/DVD drive is stopped.
Template:Moduledocbox

31 Center set

31.1 Definition

INT center_set (<**INT** FileID> , <**INT** GraphID> , <**INT** x> , <**INT** y>)

Allows you to change the center of a graphic.

This function changes a graphic's center pixel, which is the pixel that is located on screen at the graph's x and y coordinates. So changing this will influence the position on the screen.

Also called set_center().

31.2 Parameters

INT FileID	- The FileID of the file containing the graphic
INT GraphID	- The GraphID of the graphic to change the center of.
INT x	- The new horizontal center coordinate of the graphic.
INT y	- The new vertical center coordinate of the graphic.

31.3 Returns

INT : Successrate

-1 - Specified graphic is invalid.

1 - The center was set successfully.

31.4 Notes

You set the position of control point 0 in the graphic. This control point acts as the graphic's coordinate on screen. For example, if a graphic's center is set to 0,0 , then the upper left pixel of the graphic will be placed on screen at the graphic's coordinates. If no center is set, by default control point 0 is set to the graphics true geometric center.

Another key feature is that the graphic will rotate around its center, as set by this function, and any horizontal or vertical mirrors (set with flags) will flip at this point.

When a graphic is used as a mouse pointer, its center point is used for the mouse tip. Most mouse cursors in Operating Systems have a mouse with the tip in the upper left of the image. Therefore, for a standard mouse pointer in Bennu, you will have to set the center at 0,0 to enable mouse accuracy.

31.5 Example

```
import "mod_map"
import "mod_text"
import "mod_key"

Process Main()
Begin
    graph=png_load("set_center.png");    //Loads a 128x128 image as the graphic
    x=160;
    y=100;                               //Places the graphic in the centre of the screen
    write(0,0,0,0,"Press [1] to set center to 0,0");
    write(0,0,0,10,"Press [2] to set center to 64,64");
    Loop
        if(key(_1))
            center_set(0,graph,0,0);
        end
        if(key(_2))
            center_set(0,graph,64,64);
        end
        angle+=100;                      //Rotates the graphic constantly
        frame;
    End
End
```

Used in example: graph, png_load(), key(), write(), **center_set**()

File(s) used in example: set_center.png

Template:Moduledocbox

32 Chdir

32.1 Definition

INT chdir (<**STRING** directoryname>)

Sets the current path of execution.

32.2 Parameters

STRING directoryname - The name of the directory to be entered from the current path of execution or a new path of execution.

32.3 Returns

INT : Success

0 - Setting of current path of execution succeeded.
!0 - Setting of current path of execution failed.

32.4 Example

```
import "mod_dir"
import "mod_say"
import "mod_key"

Process Main()
Begin

    say(CD());
    ChDir("..");
    say(CD());

    Repeat
        frame;
    Until(key(_ESC))

End
```

Used in example: cd(), **chdir**(), say(), key()

Template:Moduledocbox

33 Chr

33.1 Definition

STRING chr (<**BYTE** ASCIIvalue>)

Returns the character associated with *ASCIIvalue*.

33.2 Parameters

BYTE ASCIIvalue - The ASCII value of which the character is wanted.

33.3 Returns

STRING : The character associated with *ASCIIvalue*.

33.4 Example

```
Program chars;
Begin

    write(0,0, 0,0,chr(65));
    write(0,0,10,0,chr(67));

    Repeat
        frame;
    Until(key(_esc))

End
```

Used in example: write(), key()

Template:Funcbox

34 Collision

34.1 Definition

INT collision (<**INT** processID|processTypeID>)

Checks if a process collided with the process calling Collision().

When a processTypeID is specified, there could be multiple fitting collisions. In this case, collision() returns a processID on each subsequent call, until it returns 0. This can be reset by use of the frame; statement, and in such case, frame(0); can be handy.

34.2 Parameters

INT processID|processTypeID - The ProcessID of the process or the ProcessTypeID of the type of processes to be checked.

34.3 Returns

INT : The ID of the collided process.

0 - No collision

>0 - The processID of the process colliding with the current process

34.4 Example

```
Process SpaceShip( int file , int graph , int x , int y , int angle , int maxspeed , int maxturnspeed )
Private
    int speed;
    int collisionID;
    string text;
Begin
        write_string(0,0,0,0,&text);
    Loop
        // Handle movement
        speed+=key(_up)*(speed<maxspeed)-key(_down)*(speed>-maxspeed);
        angle+=(key(_left)-key(_right))*maxturnspeed;
        advance(speed);
        // Handle collision
        if( (collisionID = collision(type main)))
            text = "collision with: " + collisionID;
        else
            text = "no collision";
        end
        frame;
    End
End

Process Main()
Private
    int map;
Begin

    // Create the graph for the ship
    map = new_map(20,20,8);
    map_clear(0,map,rgb(0,255,255));

    // Create the graph for the Main process
    graph = new_map(50,50,8);
    map_clear(0,graph,rgb(255,255,0));

    // Position the main process and create the ship
    x = y = z = 100;
    SpaceShip(0,map,100,100,0,20,5000);

    Repeat
        frame;
    Until(key(_ESC))

    let_me_alone();

End
```

Used in example: write_string(), key(), **collision**(), new_map(), map_clear(), advance(), let_me_alone(), graph, type

Template:Moduledocbox

35 Cos

35.1 Syntax

FLOAT cos (<**FLOAT** angle>)

35.2 Description

Returns the cosine of the specified angle.

This function performs a cosine calculation on a certain angle and returns a value between -1 and 1.

35.3 Parameters

FLOAT angle - Angle, in thousandths of degrees. i.e. 75000 = 75º

35.4 Returns

FLOAT : The cosine result of the specified angle.

35.5 Notes

The angle value used in this function should be in thousandths of degrees, as most angles within Bennu are.

To read about all aspects of trigonometry, you can visit Wikipedia's Trigonometric function page.

35.6 Example

```
Const
    screen_width  = 320;
    screen_height = 200;
    screen_border = 15;
End

Global
    float value;
End

Process Main()
Begin

    // Modes
    set_title("Cosine Graph");
    set_mode(screen_width,screen_height);

    // X axis
    for(x=1;x<=8;x++)
        write( 0,
               screen_border+x*(screen_width-screen_border)/8+3,
               screen_height-1,
               8,
               itoa(x*360/8 )+"^" );
    end
    draw_line(1,screen_height-screen_border,screen_width,screen_height-screen_border);

    // Y axis
    write(0,screen_border-1,20,5,"1");
    write(0,screen_border-1,screen_height/2,5,"0");
    write(0,screen_border-1,screen_height-20,5,"-1");
    draw_line(screen_border,1,screen_border,screen_height-1);

    // Draw tangent
    for(angle=0;angle<360;angle++)
        value=cos(angle*1000)*(screen_height/2-20);
        put_pixel( screen_border+angle*(screen_width-screen_border)/360,
                   screen_height/2-value,
                   rgb(255,255,255) );
        // screen_height/2-value because the screen's origin (0,0) is topleft instead of downleft.
    end

    Repeat
```

```
        frame;
    Until(key(_ESC))

End
```

Used in example: set_title(), set_mode(), write(), draw_line(), **cos**(), put_pixel(), key()

This will result in something like:
Template:Image

Template:Moduledocbox

36 Crypt decrypt

36.1 Definition

INT crypt_decrypt (<**INT POINTER** handle>, <**CHAR POINTER** in>, <**CHAR POINTER** out>, <**INT** blocks>,)

OR

INT crypt_decrypt (< **INT** method>, <**CHAR POINTER** key>, <**CHAR POINTER** in>, <**CHAR POINTER** out>, <**INT** blocks>,)

36.2 Description

Decrypt a selected data using a handle or creating a handle with the given method and key.

36.3 Parameters

INT method	- Method to decrypt. may be CRYPT_DES or CRYPT_3DES
INT POINTER key	- key used to the decryption.
INT POINTER handle	- handle to use to decrypt.
CHAR POINTER in	- data to encrypt.
CHAR POINTER out	- pointer for where to write decrypted data.
INT blocks	- length of the data to encrypt.

36.4 Returns

INT : Successrate

-1 - Error.
0 - Ok.
Template:Moduledocbox

37 Crypt del

37.1 Syntax

INT crypt_del (<**INT POINTER** handle>)

37.2 Description

Destroy a crypt handle.

37.3 Parameters

INT POINTER handle - handle to destroy.

37.4 Returns

INT : true

Template:Moduledocbox

38 Crypt encrypt

38.1 Syntax

INT crypt_encrypt (<**INT POINTER** handle>, <**CHAR POINTER** in>, <**CHAR POINTER** out>, <**INT** blocks>,)

OR

INT crypt_encrypt (< **INT** method>, <**CHAR POINTER** key>, <**CHAR POINTER** in>, <**CHAR POINTER** out>, <**INT** blocks>,)

38.2 Description

Encrypt a selected data using a handle or creating a handle with the given method and key.

38.3 Parameters

INT method	- Method to encrypt. may be CRYPT_DES or CRYPT_3DES
INT POINTER key	- key used to the encryption.
INT POINTER handle	- handle to use to encrypt.
CHAR POINTER in	- data to encrypt.
CHAR POINTER out	- pointer for where to write encrypted data.
INT blocks	- length of the data to encrypt.

38.4 Returns

INT : Successrate

-1 - Error.
0 - Ok.
Template:Moduledocbox

39 Crypt new

39.1 Syntax

INT POINTER crypt_new (<**INT** method> , <**CHAR POINTER** key>)

39.2 Description

Create a handle to start to encrypt information.

39.3 Parameters

INT method	- Method to encrypt. may be CRYPT_DES or CRYPT_3DES
CHAR POINTER key	- key used to the encryption.

39.4 Returns

INT POINTER : crypt handle identifier. It is 0 then the handle can not be created.

Template:Moduledocbox

40 Delete draw

40.1 Definition

INT delete_draw (<**INT** DrawID>)

Deletes a certain drawing from the screen.

40.2 Parameters

INT DrawID - DrawID of the drawing to be deleted.

40.3 Returns

INT : true

40.4 Notes

Delete_draw(0) deletes all drawings from the screen.

Template:Funcbox

41 Delete text

41.1 Definition

INT delete_text (<**INT** TextID>)

Deletes a certain text from the screen.

41.2 Parameters

INT TextID - TextID of the text to be deleted.

41.3 Returns

INT : true

41.4 Notes

Delete_text(ALL_TEXT) deletes all text from the screen.

41.5 Example

```
Program test;
Global
    my_text;
Begin
    my_text = write(0,320/2,200/2,4,"Press space to delete this.");
    Repeat
        if (key(_space))
            if(my_text>0)
                delete_text(my_text);
                my_text = 0;
            end
        end
        Frame;
    Until(key(_esc))
End
```

Used in example: write(), key(), textID

This will result in something like:
File:Delete text.jpg

42 Draw box

42.1 Definition

INT draw_box (<**INT** x0> , <**INT** y0> , <**INT** x1> , <**INT** y1>)

Draws a filled rectangle with corners *(x0,y0)*, *(x0,y1)*, *(x1,y0)* and *(x1,y1)*.

42.2 Parameters

INT x0 - The x coordinate of one corner of the filled rectangle.
INT y0 - The y coordinate of one corner of the filled rectangle.
INT x1 - The x coordinate of the diagonally opposite corner of the filled rectangle.
INT y1 - The y coordinate of the diagonally opposite corner of the filled rectangle.

42.3 Returns

INT : DrawID

-1 - Error.
1 - If drawn after drawing_map().
!-1&&!1 - The DrawID of the drawing created.
Template:Funcbox

43 Draw circle

43.1 Definition

INT draw_circle (<**INT** x> , <**INT** y> , <**INT** radius>)

Draws a non-filled circle with center *(x0,y0)* and radius *radius*.

43.2 Parameters

INT x	- The x coordinate of one center of the non-filled circle.
INT y	- The y coordinate of one center of the non-filled circle.
INT radius	- The radius of the non-filled circle.

43.3 Returns

INT : DrawID

-1	- Error.
1	- If drawn after drawing_map().
!-1&&!1	- The DrawID of the drawing created.

Template:Funcbox

44 Draw curve

44.1 Definition

INT draw_curve (<**INT** x0> , <**INT** y0> , <**INT** x1> , <**INT** y1> , <**INT** x2> , <**INT** y2> , <**INT** x3> , <**INT** y3> , <**INT** smoothness>)

Draws a curve starting at the point *(x0,y0)*, ending at the point *(x1,y1)* and influenced by the points *(x2,y2)* and *(x3,y3)* with a certain level of smoothness.

44.2 Parameters

INT x0 - The x coordinate of the starting point of the curve.
INT y0 - The y coordinate of the starting point of the curve.
INT x1 - The x coordinate of the first influence point of the curve.
INT y1 - The x coordinate of the first influence point of the curve.
INT x2 - The x coordinate of the second influence point of the curve.
INT y2 - The x coordinate of the second influence point of the curve.
INT x3 - The x coordinate of the end point of the curve.
INT y3 - The y coordinate of the end point of the curve.

44.3 Returns

INT : DrawID

-1 - Error.
1 - If drawn after drawing_map().
!-1&&!1 - The DrawID of the drawing created.
Template:Funcbox

45 Draw fcircle

45.1 Definition

INT draw_fcircle (<**INT** x> , <**INT** y> , <**INT** radius>)

Draws a filled circle with center *(x0,y0)* and radius *radius*.

45.2 Parameters

INT x	- The x coordinate of one center of the filled circle.
INT y	- The y coordinate of one center of the filled circle.
INT radius	- The radius of the filled circle.

45.3 Returns

INT : DrawID

-1	- Error.
1	- If drawn after drawing_map().
!-1&&!1	- The DrawID of the drawing created.

Template:Funcbox

46 Draw line

46.1 Definition

INT draw_line(<**INT** x0> , <**INT** y0> , <**INT** x1> , <**INT** y1>)

Draws a line from point *(x0,y0)* to point *(x1,y1)*.

46.2 Parameters

INT x0 - The x coordinate of one point of the line.
INT y0 - The y coordinate of one point of the line.
INT x1 - The x coordinate of the other point of the line.
INT y1 - The y coordinate of the other point of the line.

46.3 Returns

INT : DrawID

-1 - Error.
1 - If drawn after drawing_map().
!-1&&!1 - The DrawID of the drawing created.
Template:Funcbox

47 Draw rect

47.1 Definition

INT draw_rect (<**INT** x0> , <**INT** y0> , <**INT** x1> , <**INT** y1>)

Draws a non-filled rectangle with corners *(x0,y0)*, *(x0,y1)*, *(x1,y0)* and *(x1,y1)*.

47.2 Parameters

INT x0 - The x coordinate of one corner of the non-filled rectangle.
INT y0 - The y coordinate of one corner of the non-filled rectangle.
INT x1 - The x coordinate of the diagonally opposite corner of the non-filled rectangle.
INT y1 - The y coordinate of the diagonally opposite corner of the non-filled rectangle.

47.3 Returns

INT : DrawID

-1 - Error.
1 - If drawn after drawing_map().
!-1&&!1 - The DrawID of the drawing created.
Template:Funcbox

48 Drawing alpha

48.1 Definition

INT drawing_alpha (<**INT** alpha>)

Tells Bennu to draw the coming drawings with a certain alpha value

Note: there currently is a 'bug' that makes drawing objects (draw commands with no map as target) have their alpha value altered too. This is to be changed.

48.2 Parameters

INT alpha - The alpha value to be drawn with.

48.3 Returns

INT : true

Template:Funcbox

49 Drawing color

49.1 Definition

INT drawing_color (<**INT** color>)

Tells Bennu to draw the coming drawings in a certain color.

49.2 Parameters

INT color - The color to be drawn in (see rgb()).

49.3 Returns

INT : true

Template:Funcbox

50 Drawing map

50.1 Definition

INT drawing_map (<**INT** fileID> , <**INT** graphID>)

Tells Bennu to draw the coming drawings on a certain graphic.

In order to draw with a certain z value again, drawing_z() can be used.

50.2 Parameters

INT fileID	- The fileID of the file that holds the graphic.
INT graphID	- The graphID of the graphic to draw on.

50.3 Returns

INT : true

Template:Funcbox

51 Drawing stipple

51.1 Definition

INT drawing_stipple (<**INT** stipples>)

Tells Bennu which pixels to draw of the coming drawings.

This is done by passing a 32bit value, each bit representing a pixel. Bit 0 represents the first pixels drawn, bit 1 represents the second, etc. When a 33rd pixel is to be drawn or not, bit 0 is checked and the cycle starts over. This means a value of `0xFFFFFFFF` (=`2^32-1`) means normal operation, meaning all the pixels will be drawn.

Note that this works only for non-filled drawings. For draw_curve(), the pattern is not always visible for the higher smoothness levels.

51.2 Parameters

INT stipples - Which pixels to draw, repetitive 32bits.

51.3 Returns

INT : true

51.4 Example

```
Program example;
Private
//    int draw_id;
Begin

    // Draw in background
    drawing_map(0,background);

    // Set stipplemode to display every other pixel.
    // binary code 0101 0101 0101 0101 0101 0101 0101 0101
    // hex code 55555555h
    drawing_stipple(55555555h);

    // Draw two lines
    draw_line(10,10,190,10);
    draw_line(11,12,190,12);

    // Draw this funky pattern
    // binary code 0011 1100 0111 1100 1010 0001 1101 0011
    // hex code 3C7CA1D3h
    drawing_stipple(3C7CA1D3h);

    // Draw two lines
    draw_line(10,20,190,20);
    draw_line(11,22,190,22);

    // Draw a circle
    draw_circle(100,100,50);

    // Draw a rectangle
    draw_rect(50,50,150,150);

    // Draw some lines
    draw_line( 50, 50,100,150);
    draw_line(100,150,150, 50);
    draw_line( 50,150,100, 50);
    draw_line(100, 50,150,150);

    // Draw two curves: one with high smoothness (bit pattern not visible) and one with low smoothness
    draw_curve( 200,200,
                100,200,
                100,150,
                300,100,15);
    draw_curve( 200,200,
                100,200,
                100,150,
                300,100,1);

    // Draw a filled circle
    draw_fcircle(20,180,15);
```

```
    // Draw a filled rectangle
    draw_box(50,180,80,195);

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

End
```

Used in example: drawing_map(), draw_line(), draw_circle(), draw_rect(), draw_curve(), draw_fcircle(), draw_box(), key()

This will result in something like:
Template:Image

Template:Funcbox

52 Drawing z

52.1 Definition

INT drawing_z (<**INT** z>)

Tells Bennu to draw the coming drawings on a certain z value.

In order to draw on a certain graphic again, drawing_map() can be used.

52.2 Parameters

INT z - The z value to be drawn on.

52.3 Returns

INT : true

Template:Funcbox

53 Exec

53.1 Definition

INT exec (<**INT** mode> , <**STRING** executable>, <**INT** number_of_arguments> , <**STRING POINTER** arguments>)

Executes the specified executable with the specified arguments in the specified mode.

53.2 Parameters

INT mode	- The mode to call the executable (_P_WAIT/_P_NOWAIT).
STRING executable	- The executable to start.
INT number_of_arguments	- The number of arguments given with *arguments*
STRING POINTER arguments	- Pointer to an array of strings to be passed; *number_of_arguments* strings will be passed as arguments.

53.3 Returns

INT

-1	- Error.
mode==_P_WAIT:	- The exit status of the executable.
mode==_P_NOWAIT:	- The process ID of the executable. This is **not** a ProcessID, it is a process ID of the operating system.

53.4 Notes

The mode parameter can be two things:

_P_WAIT	- Wait for the executable to exit.
_P_NOWAIT	- Do not wait for the executable to exit.

53.5 Example

Open file.txt in notepad:

```
import "mod_sys"

Process Main()
Private
    string arg;
Begin
    arg = "file.txt";
    exec(_P_NOWAIT,"notepad.exe",1,&arg);
End
```

Used in example: **exec**(), pointer

Template:Moduledocbox

54 Exists

54.1 Definition

INT Exists (<**INT** processID|processTypeID>)

Checks if a process is alive, using its processID or checks if there is a process alive of a certain processType, using its processTypeID.

54.2 Parameters

INT processID|processTypeID - The ProcessID of the process or the ProcessTypeID of the type of processes to be checked.

54.3 Returns

INT : The result of the check

0 (false) - The process with given processID is not alive or there are no processes alive of the given processTypeID.
1 (true) - The process with given processID is alive or there is at least one process alive of the given processTypeID.

54.4 Example

```
import "mod_proc"
import "mod_say"

Process Main()
Begin

    Proc();

    if(exists(id))
        say("I exist! (id)");
    end

    if(exists(0))
        say("0 exists!");
    else
        say("0 doesn't exist!");
    end

    if(exists(type proc))
        say("1- Proc exists!");
    else
        say("1- Proc doesn't exist!");
    end

    let_me_alone();

    if(exists(type proc))
        say("2- Proc exists!");
    else
        say("2- Proc doesn't exist!");
    end

End

Process Proc()
Begin
    Loop
        frame;
    End
End
```

Used in example: **exists**(), say(), let_me_alone()

Template:Moduledocbox

55 Exit

55.1 Definition

INT exit ([<**STRING** message>] , [<**INT** code>])

Exits the program. It can optionally pass a message or an error code thing.

55.2 Parameters

[**STRING** message]	- A message to pass outside the program as it exits.
[**INT** code]	- A code to pass outside the program as it exits.

55.3 Returns

INT : true

Template:Funcbox

56 Fade

56.1 Definition

INT fade (<**INT** red> , <**INT** green> , <**INT** blue> , <**INT** speed>)

Fades the screen from the current setting to the specified setting (*red*,*green*,*blue*) at the specified speed.

56.2 Parameters

INT red	- Amount of red shown from 0 to 200. 100 is normal.
INT green	- Amount of red shown from 0 to 200. 100 is normal.
INT blue	- Amount of red shown from 0 to 200. 100 is normal.
INT speed	- The speed of the fade from 1 to 64.

56.3 Returns

INT : true

56.4 Notes

Standard RGB combinations:

(R,G,B)	- *Description*
(0,0,0)	- Black out.
(100,100,100)	- Normal.
(200,200,200)	- White out.

The number of frames the fading will take can be calculated like this:

```
frames = roundup( 64 / speed )
speed = roundup( 64 / frames )
```

So:

Speed	- *Description*
<0	- Takes 1 frame.
0	- Pointless.
1	- Takes 64 frames.
2	- Takes 32 frames.
3	- Takes 22 frames.
>=64	- Takes 1 frame.

See also fade_on() and fade_off().

57 Fade off

57.1 Definition

INT fade_off ( )

Fades the screen from the current setting to black out.

This call is equivalent to `fade(0,0,0,16)`.

57.2 Returns

INT : true

58 Fade on

58.1 Definition

INT fade_on ( )

Fades the screen from the current setting to normal.

This call is equivalent to `fade(100,100,100,16)`.

58.2 Returns

INT : true

59 Fclose

59.1 Definition

INT fclose (<**INT** filehandle>)

Unloads a file previously loaded with fopen().

59.2 Parameters

INT filehandle - The FileHandle of the file returned by fopen().

59.3 Returns

INT: true

59.4 Example

```
Process loadthing(STRING loadpath);
Private
    int handle;   // handle for the loaded file
    int druppels; // here's where the loaded data go
Begin

    handle=fopen(loadpath,O_READ); // opens the file in reading mode
    fread(handle,druppels);        // reads from the file and puts the data in druppels
    fclose(handle);                // zipping up after business is done
    write(0,0,0,0,druppels);       // shows the value of druppels
End
```

Used in example: fopen(), fread(), write()

60 Feof

60.1 Definition

INT feof (<**INT** filehandle>)

Checks if the end of a certain file is reached.

60.2 Parameters

INT filehandle - The FileHandle of the file returned by fopen().

60.3 Returns

INT: true/false: Whether the end of the specified file is reached.

61 Fexists

61.1 Syntax

INT fexists (<**STRING** filename>)

61.2 Description

Checks if a certain file exists.

Also called file_exists().

61.3 Parameters

STRING filename - The file to be checked for existence, including a possible path.

61.4 Returns

INT : true/false: the existence of the file.

true - The specified file exists.
false - The specified file doesn't exist.

61.5 Example

```
import "mod_file"
import "mod_say"

Process Main()
Begin

    say("> C:\Windows\notepad.exe > " + fexists("C:\Windows\notepad.exe")); // A filename with
                                                                           // absolute path
    say("> SDL.dll > " + fexists("SDL.dll")); // A filename without a path
    say("> SomeNonExistingFile > " + fexists("SomeNonExistingFile")); // A nonexisting file

End
```

Used in example: say(), **fexists**()

Template:Moduledocbox

62 Fget angle

62.1 Definition

INT fget_angle (<**INT** pointA-X> , <**INT** pointA-Y> , <**INT** pointB-X> , <**INT** pointB-Y>)

Returns the angle between two certain points. The returned angle will be ranging from 0 to 360000 (0-360º).

62.2 Parameters

INT pointA-X - The X-coordinate of point A.
INT pointA-Y - The Y-coordinate of point A.
INT pointB-X - The X-coordinate of point B.
INT pointB-Y - The Y-coordinate of point B.

62.3 Returns

INT : The angle between point A and point B.

62.4 Notes

The angle value returned by this function is in thousandths of degrees, as most angles within Bennu are.

62.5 Example

```
Const
    screen_width     = 320;
    screen_height    = 200;
    screen_depth     = 8;
    screen_fps       = 60;
    screen_frameskip = 0;
End

Global
    int distance;
    int tempID;
End

Process Main()
Begin

    // Set the screen mode
    set_mode(screen_width,screen_height,screen_depth);
    set_fps(screen_fps,screen_frameskip);

    // Change this to see what happens
    resolution = 100;

    // Create mouse graph, assign to mouse.graph
    mouse.graph = new_map(20,20,screen_depth);
    map_clear(0,mouse.graph,rgb(255,0,0));

    // Create arrow, assign to graph
    graph = new_map(30,30,screen_depth);
    drawing_map(0,graph);
    drawing_color(rgb(0,255,0));
    draw_line( 0,29,29,30/2);
    draw_line( 0, 0,30,30/2);

    // Set position
    x = screen_width /2 * resolution;
    y = screen_height/2 * resolution;

    // Display distance
    write(0,0,0,0,"Distance:");
    write_int(0,60,0,0,&distance);

    // Always point to the mouse
    Repeat
        // Get the angle and distance between this process' coordinates and those of the mouse.
        angle    = fget_angle(x,y,mouse.x*resolution,mouse.y*resolution);
        distance = fget_dist (x,y,mouse.x*resolution,mouse.y*resolution);
```

```
        frame;
    Until(key(_esc))

End
```

Used in example: set_mode(), new_map(), map_clear(), drawing_map(), drawing_color(), draw_line(), write(), write_int(), **fget_angle**(), fget_dist(), resolution, mouse, graph, x, y, angle

This example could also be done with get_angle(), but that would be more work.

It could look something like:
Template:Image

Template:Funcbox

63 Fget dist

63.1 Definition

INT fget_dist (<**INT** pointA-X> , <**INT** pointA-Y> , <**INT** pointB-X> , <**INT** pointB-Y>)

Returns the distance between two certain points.

63.2 Parameters

INT pointA-X - The X-coordinate of point A.
INT pointA-Y - The Y-coordinate of point A.
INT pointB-X - The X-coordinate of point B.
INT pointB-Y - The Y-coordinate of point B.

63.3 Returns

INT : The distance between point A and point B.

63.4 Example

```
Program angling;
Const
    screen_width     = 320;
    screen_height    = 200;
    screen_depth     = 8;
    screen_fps       = 60;
    screen_frameskip = 0;
Global
    int distance;
    int tempID;
Begin

    // Set the screen mode
    set_mode(screen_width,screen_height,screen_depth);
    set_fps(screen_fps,screen_frameskip);

    // Change this to see what happens
    resolution = 100;

    // Create mouse graph, assign to mouse.graph
    mouse.graph = new_map(20,20,screen_depth);
    map_clear(0,mouse.graph,rgb(255,0,0));

    // Create arrow, assign to graph
    graph = new_map(30,30,screen_depth);
    drawing_map(0,graph);
    drawing_color(rgb(0,255,0));
    draw_line( 0,29,29,30/2);
    draw_line( 0, 0,30,30/2);

    // Set position
    x = screen_width /2 * resolution;
    y = screen_height/2 * resolution;

    // Display distance
    write(0,0,0,0,"Distance:");
    write_int(0,60,0,0,&distance);

    // Always point to the mouse
    Repeat
        // Get the angle and distance between this process' coordinates and those of the mouse.
        angle    = fget_angle(x,y,mouse.x*resolution,mouse.y*resolution);
        distance = fget_dist (x,y,mouse.x*resolution,mouse.y*resolution);
        frame;
    Until(key(_esc))

End
```

Used in example: set_mode(), new_map(), map_clear(), drawing_map(), drawing_color(), draw_line(), write(), write_int(), fget_angle(), **fget_dist**(), resolution, mouse, graph, x, y, angle

This example could also be done with get_dist(), but that would be more work. It also gives a much less accurate distance when the resolution is >1.

Resolution is 100:

Template:Image VS Template:Image
Template:Funcbox

64 Fgets

64.1 Definition

STRING fgets (<**INT** filehandle>)

Reads a line from a certain file and returns it. Subsequent calls will return the next line, until the end of the file is reached.

64.2 Parameters

INT filehandle - The FileHandle of the file returned by fopen().

64.3 Returns

STRING: The line read.

64.4 Notes

The returned string normally does not contain the '\n' or '\r','\n' charactersets.

When a line ends with the character '\', the next line will be joined with the current one, changing the '\' character to a '\n' character.

65 Find

65.1 Definition

INT find (<**STRING** str> , <**STRING** searchstring> , [<**INT** startposition>])

Returns the position of the firstly found appearance of a *searchstring* in *str* or -1 if not found. The starting position can be specified optionally.

65.2 Parameters

STRING str	- The string in which to search.
STRING searchstring	- The string to search for.
[**INT** startposition]	- The position at which to start searching. Default is 0.

65.3 Returns

INT : The position of the firstly found appearance of *searchstring* in *str* or -1 if not found.

65.4 Notes

A first character of a string is at position 0.

Template:Moduledocbox

66 Flength

66.1 Definition

INT flength (<**INT** filehandle>)

Gives back the size of a certain file.

66.2 Parameters

INT filehandle - The FileHandle of the file returned by fopen().

66.3 Returns

INT: The size of the file in bytes.

67 Fnt load

67.1 Definition

INT fnt_load (<**STRING** filename>)

Loads a FNT file into memory as a font. A font is usually with the extension *.fnt*.

Also called load_fnt().

67.2 Parameters

STRING filename - The filename of the FNT file that you wish to load (including extension and possible path).

67.3 Returns

INT : FontID

-1 - Error: file does not exist.
0 - Filename could not be obtained from the specified string (doesn't happen usually).
>0 - The FontID.

67.4 Errors

Format not recognized - The format of the specified file could not be recognized.
Template:Moduledocbox

68 Fnt new

68.1 Definition

INT fnt_new (<**INT** depth>)

Creates a new font with a certain color depth.

Also called new_fnt().

68.2 Parameters

STRING depth - The color depth of the glyphs of the font.

68.3 Returns

INT : FontID

-1 - Error: could not create font.
>=0 - The FontID.

68.4 Errors

Insufficient memory - There is insufficient memory available. This error doesn't occur often.
Too many fonts - There are too many fonts loaded (255).
Template:Moduledocbox

69 Fnt save

69.1 Definition

INT fnt_save (<**INT** fontID> , <**STRING** filename>)

Saves a font as a file. A font is usually with the extension ".fnt".

If the specified filename contains no extension, ".fnt" is added to it.

Also called save_fnt().

69.2 Parameters

INT fontID	- The fontID of the font to be saved.
STRING filename	- The name of the font file to be saved, including a possible path.

69.3 Returns

INT : Successrate

false	- Filename could not be obtained from the specified string (doesn't happen usually) or one of the errors.
true	- Font successfully saved.

69.4 Errors

Invalid fontID	- The specified fontID was invalid.
Unable to create file	- The file could not be created.
Font corrupt	- The font trying to be saved is corrupt.
Insufficient memory	- There is insufficient memory available. This error doesn't occur often.

Template:Moduledocbox

70 Fnt unload

70.1 Definition

INT fnt_unload (<**INT** fontID>)

Unloads the specified font from memory.

Also called unload_fnt().

70.2 Parameters

INT fontID - The fontID of the file to unload.

70.3 Returns

INT : false

Template:Moduledocbox

71 Fopen

71.1 Definition

INT fopen (<**STRING** filename> , <**INT** mode>)

Opens a file on the hard drive for reading or writing.

71.2 Parameters

STRING filename	- The filename of the file you wish to open (including extension and possible path).
INT mode	- The mode with which to access the file (see Readwrite_modes).

71.3 Returns

INT : FileHandle

0 - Could not load.
!0 - The identifier of the file now opened for reading/writing.

71.4 Example

```
Process loadthing(STRING loadpath);
Private
    int handle;   // handle for the loaded file
    int druppels; // here's where the loaded data go
Begin

    handle=fopen(loadpath,O_READ); // opens the file in reading mode
    fread(handle,druppels);        // reads from the file and puts the data in druppels
    fclose(handle);                // zipping up after business is done
    write(0,0,0,0,druppels);       // shows the value of druppels

End
```

Used in example: fread(), fclose(), write()

72 Fpg add

72.1 Definition

INT fpg_add (<**INT** destFileID> , <**INT** destGraphID> , <**INT** origFileID> , <**INT** origGraphID>)

Copies a certain graphic in a certain file to a certain file with a certain graphID.

72.2 Parameters

INT destFileID	- The fileID of the destination file.
INT destGraphID	- The graphID in the destination file where to add the graphic.
INT origFileID	- The fileID of the original file where the graphic is located.
INT origGraphID	- The graphID of the graphic to add.

72.3 Returns

INT : graphID

-1	- Invalid destination file or graphID.
0	- Invalid original file or graphID.
>0	- The GraphID of the destination graphic.

Template:Moduledocbox

73 Fpg exists

73.1 Definition

INT fpg_exists (<**INT** FileID>)

Checks if an FPG exists with the specified FileID.

73.2 Parameters

INT FileID - The FileID of the File to check for existence.

73.3 Returns

INT : Whether the File exists

false - The specified File does not exist.
true - The specified File exists.
Template:Moduledocbox

74 Fpg load

74.1 Definition

INT fpg_load (<**STRING** filename>)

Loads a graphics library into your program.

This function loads the whole contents of an FPG graphics library into your project, enabling the contained graphics to be used in your program as process' graphs, screen backgrounds (put_screen()) or other graphics.

Also called load_fpg().

74.2 Parameters

STRING filename	- The filename of the FPG that you wish to load (including extension and possible path).

74.3 Returns

INT : FileID

-1	- The specified archive could not be loaded.
>=0	- The FileID assigned to the archive.

74.4 Errors

Archive not found	- The specified archive could not be found.

74.5 Notes

Using an FPG file to contain all or some of the graphics used in a Bennu program is convenient, but isn't always the best way to load graphics. Other methods of loading graphics into your program include load_map() and load_png() which load individual graphic files. Graphics loaded individually will be given FileID 0 and a GraphID starting at 1000. This is because mod_map reserves room for the first FPG loaded (FPG files can contain 999 different graphics max.), sometimes referred to as the system file or environment file.

The first FPG file loaded using mod_map returns and uses the FileID 0, which is reserved by mod_map for use as the system file. All extra FPG files loaded will have a different FileID, progressing from 1 upwards.

An FPG file holds all its contained graphs in memory simultaneously. Having a lot of large graphs being read from a single FPG file at once has been known to cause a slowdown.

Once an FPG file is no longer necessary to be held in the memory, its memory space can be released by using the function unload_fpg(). It is not necessary to unload files at the termination of a program, as Bennu always releases all used memory at the end of program execution.

74.6 Example

```
Program example;
Global
    int my_fpg;
Begin
    my_fpg=load_fpg("test.fpg");   //Loads the FPG file into memory
    put_screen(my_fpg,1);          //Puts graphic with code 1 onto screen
    Repeat
        frame;
    Until(key(_esc))
    unload_fpg(my_fpg);
End
```

Used in example: put_screen(), unload_fpg()

Template:Moduledocbox

75 Fpg new

75.1 Definition

INT fpg_new ( )

Creates and initializes a new file.

To add graphics to the created file, use the returned fileID in the function fpg_add(). The file can be saved with fpg_save(). To free a file, use fpg_unload().

Also called new_fpg().

75.2 Returns

INT : fileID

-1 - Too many files or insufficient memory.
>=0 - The fileID of the new file.

75.3 Errors

Insufficient memory - There is insufficient memory available. This error doesn't occur often.
Template:Moduledocbox

76 Fpg save

76.1 Definition

INT fpg_save(<**INT** fileID> , <**STRING** filename>)

Saves a certain file to disk.

Also called save_fpg().

76.2 Parameters

INT fileID	- The fileID of the file to save.
STRING filename	- The name of the file to be saved, including a possible path.

76.3 Returns

INT : Success

false	- Invalid fileID or filename; The errors under Errors.
true	- Success.

76.4 Errors

Insufficient memory	- There is insufficient memory available. This error doesn't occur often.
Empty library	- The specified file contains no graphics.
Unsupported color depth	- A graphic in the specified file has an unsupported color depth.
Differing color depths	- An FPG can't hold graphics of different color depths.

Template:Moduledocbox

77 Fpg unload

77.1 Definition

INT fpg_unload (<**INT** fileID>)

Unloads a certain file from memory.

Also called unload_fpg().

77.2 Parameters

INT fileID - The fileID of the file to unload.

77.3 Returns

INT : true

Template:Moduledocbox

78 Fputs

78.1 Definition

INT fputs (<**INT** filehandle> , <**STRING** line>)

Writes a line to a certain file.

78.2 Parameters

INT filehandle - The FileHandle of the file returned by fopen().

78.3 Returns

INT: Number of bytes written.

0 - There was an error.
>0 - Number of bytes written.

78.4 Notes

The character '\' will be put in front of every newline character, so that fgets() reads the lines like they were written.

79 Fread

79.1 Definition

INT fread (<**INT** filehandle> , <**VARSPACE** data>)

INT fread (<**VOID POINTER** data> , <**INT** length> , <**INT** filehandle>)

Reads the information from a file loaded with fopen() to a variable.

79.2 Parameters

INT filehandle	- The FileHandle of the file returned by fopen().
VARSPACE data	- The data to read from the file (any type of variable). It will be loaded into this variable.

79.3 Returns

INT : The number of bytes read from the file.

79.4 Example

```
Process loadthing(STRING loadpath);
Private
    int handle;   // handle for the loaded file
    int druppels; // here's where the loaded data go
Begin

    handle=fopen(loadpath,O_READ); // opens the file in reading mode
    fread(handle,druppels);        // reads from the file and puts the data in druppels
    fclose(handle);                // zipping up after business is done
    write(0,0,0,0,druppels);       // shows the value of druppels

End
```

Used in example: fopen(), fclose(), write()

80 Free

80.1 Definition

INT free (<**VOID POINTER** data>)

Frees a block of memory.

The pointer used must be a pointer to a previously allocated block of memory, else the behavior of free() is undefined.

80.2 Parameters

VOID POINTER data - Pointer to the block of memory to be freed.

80.3 Returns

INT : true

80.4 Example

```
Program example;
Private
    byte pointer pbyte;
    word pointer pword;
    int  pointer pint;
    int elements = 10;
    int i;
Begin

    // Allocate memory
    pbyte = alloc(elements);
    pword = alloc(elements*sizeof(word));
    pint  = alloc(elements*sizeof(int));

    // Reset memory to 0's
    memset (pbyte,0,elements);
    memsetw(pword,0,elements); // same as  memset(pword,0,elements*sizeof(word));
                               // because value-parameter is 0.
    memset (pint ,0,elements*sizeof(int)); // There isn't a "memseti()", so we need to
                                           // set the individual bytes to 0. To change
                                           // ints to nonzero values, memset() can't be
                                           // used easily

    // Write numbers to bytes and ints
    for(i=0; i<elements; i++)
        pbyte[i]  = 133; // pbyte[i] is the same as *(pbyte+i)
        *(pint+i) = 4555; // pint[i] is the same as *(pint+i)
    end

        // Write numbers to words
        memsetw(pword,345,elements);

    // Show numbers
    for(i=0; i<elements; i++)
        say("byte["+i+"] = " + *(pbyte+i));
        say("word["+i+"] = " + pword[i]);
        say("int ["+i+"] = " + pint[i]);
    end

    Repeat
        frame;
    Until(key(_esc))

    // Free the used memory
    free(pbyte);
    free(pword);
    free(pint);

End
```

Used in example: alloc(), memset(), memsetw(), sizeof(), say(), **free**(), pointer

Template:Funcbox

81 Fseek

81.1 Definition

INT fseek (<**INT** filehandle> , <**INT** position> , <**INT** seek_mode>)

Sets the byte offset (reading position) of a certain file. This means where a function will start reading in that file.

81.2 Parameters

INT filehandle	- The FileHandle of the file returned by fopen().
INT position	- Number of bytes from the point indicated by *seek_mode*.
INT seek_mode	- Set the offset relative to a certain point (see seek modes).

81.3 Returns

INT: The new reading position.

82 Ftell

82.1 Definition

INT ftell (<**INT** filehandle>)

Returns the current reading position of a certain file.

The reading position can be altered by fseek().

82.2 Parameters

INT filehandle - The FileHandle of the file returned by fopen().

82.3 Returns

INT: The current reading position of the specified file.

83 Ftime

83.1 Syntax

STRING ftime (<**STRING** format> , <**INT** time>)

83.2 Description

Puts a certain time in a certain format.

It returns the specified string, with certain keywords replaced with their corresponding values, according to the specified time (see Notes). The current time is fetched by use of the function time().

83.3 Parameters

STRING format	- The format wanted.
INT time	- The time to be put in the formatted string.

83.4 Returns

STRING : The formatted string.

83.5 Notes

A list of keywords:

Keyword	*- Replaced by*
%a	- the locale's abbreviated weekday name.
%A	- the locale's full weekday name.
%b	- the locale's abbreviated month name.
%B	- the locale's full month name.
%c	- the locale's appropriate date and time representation.
%C	- the century number (the year divided by 100 and truncated to an integer) as a decimal number [00-99].
%d	- the day of the month as a decimal number [01,31].
%D	- the same as %m/%d/%y.
%e	- the day of the month as a decimal number [1,31]; a single digit is preceded by a space.
%h	- the same as %b.
%H	- the hour (24-hour clock) as a decimal number [00,23].
%I	- the hour (12-hour clock) as a decimal number [01,12].
%j	- the day of the year as a decimal number [001,366].
%m	- the month as a decimal number [01,12].
%M	- the minute as a decimal number [00,59].
%n	- a newline character.
%p	- the locale's equivalent of either a.m. or p.m.
%r	- the time in a.m. and p.m. notation; in the POSIX locale this is equivalent to %I:%M:%S %p.
%R	- the time in 24 hour notation (%H:%M).
%S	- the second as a decimal number [00,61].
%t	- a tab character.
%T	- the time (%H:%M:%S).
%u	- the weekday as a decimal number [1,7], with 1 representing Monday.
%U	- the week number of the year (Sunday as the first day of the week) as a decimal number [00,53].
%V	- the week number of the year (Monday as the first day of the week) as a decimal number [01,53]. If the week containing 1 January has four or more days in the new year, then it is considered week 1. Otherwise, it is the last week of the previous year, and the next week is week 1.
%w	- the weekday as a decimal number [0,6], with 0 representing Sunday.
%W	- the week number of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Monday are considered to be in week 0.

%x - the locale's appropriate date representation.
%X - the locale's appropriate time representation.
%y - the year without century as a decimal number [00,99].
%Y - the year with century as a decimal number.
%Z - the timezone name or abbreviation, or by no bytes if no timezone information exists.
%% - %.

83.6 Example

```
import "mod_timer"
import "mod_time"
import "mod_text"
import "mod_key"

Process Main()
Private
    String timestring; // The string holding the formatted time
Begin

    write_string(0,0,0,0,&timestring); // Display the timestring
    timer = 100; // Make it so it updates the timestring immediately

    Repeat
        if(timer>100) // Update the timestring every 1 second
            timer = 0;
            timestring = ftime("%d-%m-%Y %H:%M:%S",time());
        end
        frame;
    Until(key(_esc))

End
```

Used in example: write_string(), time(), key(), timer

Template:Moduledocbox

84 Ftoa

84.1 Definition

STRING ftoa (<**FLOAT** value>)

Returns a string containing a certain float value.

84.2 Parameters

FLOAT value - The value the returned string will contain.

84.3 Returns

STRING : The string containing the specified value, including sign and decimal point.

Template:Funcbox

85 Function:File

Up to Files Functions

85.1 Syntax

STRING file (<**STRING** filename>)

85.2 Description

Returns the whole contents of a certain file.

85.3 Parameters

STRING filename - The filename of the file you wish to read from (including extension and possible path).

85.4 Returns

STRING: The contents of the file.

Template:Moduledocbox

86 Fwrite

86.1 Definition

INT fwrite (<**INT** filehandle> , <**VARSPACE** data>)

Writes data to a file loaded with fopen.

86.2 Parameters

INT filehandle	- Identifier of the file loaded with fopen.
VARSPACE data	- The data to write to the file (any type of variable).

86.3 Returns

INT : The number of bytes written to the file.

86.4 Example

```
Process writething(STRING loadpath);
Private
    handle;   // handle for the loaded file
    druppels; // the data to write to the file

Begin

    druppels=rand(1,10);

    handle=fopen(loadpath,O_WRITE); // opens the file in writing mode
    fwrite(handle,druppels);        // writes the druppels variable to the file
    fclose(handle);                 // zipping up after business is done

End
```

Used in example: fopen(), fclose(), rand()

87 Get angle

87.1 Definition

INT get_angle (<**INT** processID>)

Returns the angle between the coordinates of a certain process and the process calling **get_angle**().

87.2 Parameters

INT processID - The other process.

87.3 Returns

INT : The wanted angle.

-1 - An error *may* have occurred: invalid process.
!-1 - The wanted angle.

87.4 Example

```
Program angling;
Const
    screen_width     = 320;
    screen_height    = 200;
    screen_depth     = 8;
    screen_fps       = 60;
    screen_frameskip = 0;
Global
    int distance;
    int tempID;
Begin

    // Set the screen mode
    set_mode(screen_width,screen_height,screen_depth);
    set_fps(screen_fps,screen_frameskip);

    // Change this to see what happens
    resolution = 100;

    // Create mouse graph and start mousepointer
    x = new_map(20,20,screen_depth);
    map_clear(0,x,rgb(255,0,0));
    mousepointer(0,x);

    // Create arrow, assign to graph
    graph = new_map(30,30,screen_depth);
    drawing_map(0,graph);
    drawing_color(rgb(0,255,0));
    draw_line( 0,29,29,30/2);
    draw_line( 0, 0,30,30/2);

    // Set position
    x = screen_width /2 * resolution;
    y = screen_height/2 * resolution;

    // Display distance
    write(0,0,0,0,"Distance:");
    write_int(0,60,0,0,&distance);

    // Always point to the mouse
    Repeat
        // Get the angle and distance between this process' coordinates and those of mousegraph.
        // We can use TYPE and get_id() here, because usually there would only be one
        // mousepointer and one always.
        tempID = get_id(type mousepointer);
        angle = get_angle(tempID);
        distance = get_dist(tempID);
        frame;
    Until(key(_esc))

End

/**
 * Follows the mouse coordinates. x is always mouse.x and y is always mouse.y
 * for processes with priority <1. The graphic of this process will be a certain graphic.
```

```
 * int file     - The fileID of the file where the graphic is located
 * int graph    - The graphID of the graphic to be used for this process
 */
Process mousepointer(int file,int graph)
Begin
    // Set the priority to 1, because we first want to have the correct coordinates of
    // the mouse set in this process. Then after that other process can use those coordinates.
    priority = 1;
    // Obtain father's resolution
    resolution = father.resolution;
    // Loop
    Loop
        // Obtain X and Y coordinates of the mouse and adjust for resolution
        // (mouse.y and mouse.y have an unchangeable resolution of 1)
        x = mouse.x * resolution;
        y = mouse.y * resolution;
        frame;
    End
End
```

Used in example: set_mode(), new_map(), map_clear(), drawing_map(), drawing_color(), draw_line(), write(), write_int(), get_id(), **get_angle**(), get_dist(), resolution, mouse, graph, x, y, angle, priority

This example could also be done with fget_angle(), which is easier and doesn't require an extra process.

It could look something like:
http://wwwhome.cs.utwente.nl/~bergfi/fenix/wiki/get_angle.PNG

Template:Moduledocbox

88 Get desktop size

88.1 Syntax

INT get_desktop_size (<**INT POINTER** width>, <**INT POINTER** height>)

88.2 Description

Get the desktop size.

88.3 Parameters

INT POINTER width	- Pointer to where the desktop width will be written.
INT POINTER height	- Pointer to where the desktop height will be written.

88.4 Returns

INT : Successrate

false	- Error.
true	- Ok.

Template:Moduledocbox

89 Get dist

89.1 Definition

INT get_dist (<**INT** processID>)

Returns the distance between the coordinates of a certain process and the process calling **get_dist**(). The distance returned is converted to the resolution of the process calling **get_dist**().

89.2 Parameters

INT processID - The other process.

89.3 Returns

INT : The wanted distance.

-1 - An error occurred: invalid process.
!-1 - The wanted distance.

89.4 Example

```
Program angling;
Const
    screen_width     = 320;
    screen_height    = 200;
    screen_depth     = 8;
    screen_fps       = 60;
    screen_frameskip = 0;
Global
    int distance;
    int tempID;
Begin

    // Set the screen mode
    set_mode(screen_width,screen_height,screen_depth);
    set_fps(screen_fps,screen_frameskip);

    // Change this to see what happens
    resolution = 100;

    // Create mouse graph and start mousepointer
    x = new_map(20,20,screen_depth);
    map_clear(0,x,rgb(255,0,0));
    mousepointer(0,x);

    // Create arrow, assign to graph
    graph = new_map(30,30,screen_depth);
    drawing_map(0,graph);
    drawing_color(rgb(0,255,0));
    draw_line( 0,29,29,30/2);
    draw_line( 0, 0,30,30/2);

    // Set position
    x = screen_width /2 * resolution;
    y = screen_height/2 * resolution;

    // Display distance
    write(0,0,0,0,"Distance:");
    write_int(0,60,0,0,&distance);

    // Always point to the mouse
    Repeat
        // Get the angle and distance between this process' coordinates and those of mousegraph.
        // We can use TYPE and get_id() here, because usually there would only be one
        // mousepointer and one always.
        tempID = get_id(type mousepointer);
        angle = get_angle(tempID);
        distance = get_dist(tempID);
        frame;
    Until(key(_esc))

End

/**
 * Follows the mouse coordinates. x is always mouse.x and y is always mouse.y
```

```
 * for processes with priority <1. The graphic of this process will be a certain graphic.
 * int file     - The fileID of the file where the graphic is located
 * int graph    - The graphID of the graphic to be used for this process
 */
Process mousepointer(int file,int graph)
Begin
    // Set the priority to 1, because we first want to have the correct coordinates of
    // the mouse set in this process. Then after that other process can use those coordinates.
    priority = 1;
    // Obtain father's resolution
    resolution = father.resolution;
    // Loop
    Loop
        // Obtain X and Y coordinates of the mouse and adjust for resolution
        // (mouse.y and mouse.y have an unchangeable resolution of 1)
        x = mouse.x * resolution;
        y = mouse.y * resolution;
        frame;
    End
End
```

Used in example: set_mode(), new_map(), map_clear(), drawing_map(), drawing_color(), draw_line(), write(), write_int(), get_id(), get_angle(), **get_dist**(), resolution, mouse, graph, x, y, angle, priority

This example could also be done with fget_dist(), which is easier and doesn't require an extra process. It also give a much more accurate distance when the resolution is >1.

Resolution is 100:

http://wwwhome.cs.utwente.nl/~bergfi/fenix/wiki/get_angle.PNG	VS	http://wwwhome.cs.utwente.nl/~bergfi/fenix/wiki/fget_angle.PNG
get_angle() and **get_dist**() with a process		fget_angle() and fget_dist()

Template:Moduledocbox

90 Get distx

90.1 Definition

INT get_distx (<**INT** angle> , <**INT** distance>)

Returns the horizontal distance in pixels of a specified displacement.

This is the same as `cos(angle)*distance`.

90.2 Parameters

INT angle	- Angle, in thousandths of degrees (90° = 90000).
INT distance	- Length (in pixels) to measure.

90.3 Returns

INT : The horizontal distance, in pixels, of a specified displacement.

90.4 Notes

This function returns the width of an imaginary rectangle who's opposite corners are the specified distance apart, at the specified angle from each other.

Template:Image

90.5 Example

```
Global
    xdist;
    ydist;
    dist;
    ang;
    mydraw;
End

Process Main()
Begin

    set_mode(640,480,16);
    set_fps (50,0);
    graph = new_map(3,3,16);
    map_clear(0,graph,rgb(0,255,0));
    x = 320;
    y = 240;

    set_text_color(rgb(0,0,0));
    write    (0,60, 0,2,"X Diff: ");
    write_int(0,60, 0,0,&xdist);
    write    (0,60,10,2,"Y Diff: ");
    write_int(0,60,10,0,&ydist);
    write    (0,60,20,2,"Angle: ");
    write_int(0,60,20,0,&ang);
    write    (0,60,30,2,"Distance: ");
    write_int(0,60,30,0,&dist);

    write    (0,10,40,0,"Left/right rotates your angle, up/down changes your distance");

    put(0,graph,x,y);
    drawing_background();

    repeat
        if(key(_up))
            dist++;
        end

        if(key(_down))
            dist--;
        end

        if(key(_left))
            ang-=1000;
        end
```

```
        if(key(_right))
            ang+=1000;
        end

        xdist = get_distx(ang,dist);
        ydist = get_disty(ang,dist);

        x = 320 + xdist;
        y = 240 + ydist;

        frame;

    until(key(_esc))

    let_me_alone();
    exit();

End

Process drawing_background()
Begin
    graph = new_map(640,480,16);
    set_ceter   (0,graph,0,0);
    map_clear    (0,graph,rgb(64,0,0));
    drawing_map  (0,graph);
    drawing_color(rgb(0,0,0));
    loop
        map_clear(0,graph,rgb(255,255,255));
        mydraw = draw_line(320,240,father.x,father.y);
        frame;
        delete_draw(mydraw);
    end
OnExit
    unload_map(0,graph);
End
```

Used in example: set_mode(), set_fps(), new_map(), set_text_color(), write(), write_int(), put(), key(), **get_distx**(), get_disty(), let_me_alone(), exit(), set_center(), map_clear(), rgb(), drawing_map(), drawing_color(), draw_line(), delete_draw(), unload_map()

Template:Funcbox

91 Get disty

91.1 Definition

INT get_disty (<**INT** angle> , <**INT** distance>)

Returns the vertical distance in pixels of a specified displacement.

This is the same as `-sin(angle)*distance`.

91.2 Parameters

INT angle	- Angle, in thousandths of degrees (90° = 90000).
INT distance	- Length (in pixels) to measure.

91.3 Returns

INT : The vertical distance, in pixels, of a specified displacement.

91.4 Notes

This function returns the height of an imaginary rectangle who's opposite corners are the specified distance apart, at the specified angle from each other.

Template:Image

91.5 Example

```
Global
    xdist;
    ydist;
    dist;
    ang;
    mydraw;
End

Process Main()
Begin

    set_mode(640,480,16);
    set_fps (50,0);
    graph = new_map(3,3,16);
    map_clear(0,graph,rgb(0,255,0));
    x = 320;
    y = 240;

    set_text_color(rgb(0,0,0));
    write    (0,60, 0,2,"X Diff: ");
    write_int(0,60, 0,0,&xdist);
    write    (0,60,10,2,"Y Diff: ");
    write_int(0,60,10,0,&ydist);
    write    (0,60,20,2,"Angle: ");
    write_int(0,60,20,0,&ang);
    write    (0,60,30,2,"Distance: ");
    write_int(0,60,30,0,&dist);

    write    (0,10,40,0,"Left/right rotates your angle, up/down changes your distance");

    put(0,graph,x,y);
    drawing_background();

    repeat
        if(key(_up))
            dist++;
        end

        if(key(_down))
            dist--;
        end

        if(key(_left))
            ang-=1000;
        end
```

```
        if(key(_right))
            ang+=1000;
        end

        xdist = get_distx(ang,dist);
        ydist = get_disty(ang,dist);

        x = 320 + xdist;
        y = 240 + ydist;

        frame;

    until(key(_esc))

    let_me_alone();
    exit();

End

Process drawing_background()
Begin
    graph = new_map(640,480,16);
    set_ceter   (0,graph,0,0);
    map_clear    (0,graph,rgb(64,0,0));
    drawing_map  (0,graph);
    drawing_color(rgb(0,0,0));
    loop
        map_clear(0,graph,rgb(255,255,255));
        mydraw = draw_line(320,240,father.x,father.y);
        frame;
        delete_draw(mydraw);
    end
OnExit
    unload_map(0,graph);
End
```

Used in example: set_mode(), set_fps(), new_map(), set_text_color(), write(), write_int(), put(), key(), get_distx(), **get_disty**(), let_me_alone(), exit(), set_center(), map_clear(), rgb(), drawing_map(), drawing_color(), draw_line(), delete_draw(), unload_map()

Template:Funcbox

92 Get id

92.1 Definition

INT get_id (<**INT** processTypeID>)

Returns a ProcessID of a process of the specified ProcessType. On the next call of get_id() in the same process and in the same frame, the next process will be returned of the given type. After a frame statement, get_id() is reset and will return the first process of the given processType. When there are no more processes of a given type, which have not been returned, it will return 0.

get_id(0) returns processes of any type.

92.2 Parameters

INT processTypeID - The processTypeID of the processType to get the processes' processIDs of.

92.3 Returns

INT : The processID of a process of the given processType.

0 - There are no more processes of the given processType, which have not been returned.

>0 - The processID of a process of the given processType.

92.4 Example

```
Program example;
Begin
    signaltype(type Monkey,s_kill);
End

/**
 * Empty process
 */
Process Monkey()
Begin
End

/**
 * Signals every process of type 't' the signal 'signal'.
 */
Function int signaltype(int t, int signal)
Begin
    while( (x=get_id(t)) ) // while there is an unprocessed process left and store that in 'x'
        signal(x,signal); // signal the process with processID 'x'.
    end
End

// Of course, the very observant of you already noticed that signaltype(my_type,my_signal)
// does the same thing as the function signal(my_type,my_signal), but this is just to
// illustrate the workings.

/**
 * Signals every process the signal 'signal'.
 */
Function int signalall(int signal)
Begin
    while( (x=get_id(0)) ) // while there is an unprocessed process left and store that in 'x'
        signal(x,signal); // signal the process with processID 'x'.
    end
End

// get_id(0) returns a process of any type. This is a possible implementation of a
// function which signals all existing processes. Note that this can be dangerous to use,
// as in some cases you might want one or two processes to stay alive.
```

Used in example: signal()

Template:Funcbox

93 Get joy position

1. REDIRECT joy_getposition

94 Get modes

94.1 Syntax

POINTER get_modes (<**INT** depth>, <**INT** flags>)

94.2 Description

Returns a pointer to an array of available screen dimensions for the given depth and render flags, sorted largest to smallest.

Returns NULL if there are no dimensions available for a particular format, or -1 if any dimension is okay for the given format.

94.3 Parameters

INT depth	- Color depth of the screen. See color_depths.
INT flags	- Mode of rendering. See render flags.

94.4 Returns

POINTER : A pointer to an array of available screen dimensions

94.5 Example

Used in example:

Template:Moduledocbox

95 Get real point

95.1 Definition

INT get_real_point(<**INT** controlpoint> , <**INT POINTER** x> , <**INT POINTER** y> ,)

Finds the actual position on the screen of the calling process, given its graphic and the specified *controlpoint* on this graphic. All process-related variables are taken into account, like x, y, angle, even ctype. These coordinates are then assigned to the variables pointed to by *x* and *y*.

95.2 Parameters

INT controlpoint	- The controlpoint on the process' graphic of which the actual position is wanted.
INT POINTER x	- A pointer to an integer to which the X-coordinate will be assigned.
INT POINTER y	- A pointer to an integer to which the Y-coordinate will be assigned.

95.3 Returns

INT : Successrate

false	- Error: no graphic; invalid controlpoint;
true	- Success.

Template:Moduledocbox

96 Get text color

96.1 Definition

INT get_text_color()

Gets the current text color (the color where texts will be written in).

96.2 Parameters

None.

96.3 Returns

INT: color the text will be written in.

96.4 Notes

None.

96.5 Errors

<If someone knows, please edit!>

96.6 Example

```
Program test;
Global
    my_text;
    text_color;
Begin

    set_text_color( rgb(192,112,0) );
    text_color = get_text_color();

    write    (0,320/2    ,200/2,4,"The color of this text is:");
    write_int(0,320/2+100,200/2,4,&text_color);

    Repeat
        frame;
    Until(key(_ESC))

End
```

Used in example: set_text_color(), write(), write_int(), key()

This will result in something like:
File:Get text color.png

97 Get timer

97.1 Syntax

INT get_timer ( )

97.2 Description

Returns the time the program has been running in milliseconds.

97.3 Returns

INT : The time the program has been running in milliseconds.

Template:Moduledocbox

98 Get window pos

98.1 Syntax

INT get_window_pos (<**INT POINTER** x>, <**INT POINTER** y>)

98.2 Description

Get the X and Y position of the window.

98.3 Parameters

INT POINTER x - Pointer to where the X-coordinate of the window will be written.
INT POINTER y - Pointer to where the Y-coordinate of the window will be written.

98.4 Returns

INT : Successrate

false - The system is in fullscreen.
true - Success ok.
Template:Moduledocbox

99 Get window size

99.1 Syntax

INT get_window_size (<**INT POINTER** window_width> , <**INT POINTER** window_height> , <**INT POINTER** client_width> , <**INT POINTER** client_height>)

99.2 Description

Get the window and client size.

99.3 Parameters

INT POINTER window_width	- Pointer to where the window width will be written.
INT POINTER window_height	- Pointer to where the window height will be written.
INT POINTER client_width	- Pointer to where the client width of window will be written.
INT POINTER client_height	- Pointer to where the client height of window will be written.

99.4 Returns

INT : Successrate

false	- Error.
true	- Ok.

99.5 Example

```
import "mod_key"
import "mod_video"
import "mod_text"
import "mod_wm"
Global
    desktop_width = 640;
    desktop_height = 480;
    window_width = 0;
    window_height = 0;
    client_width = 0;
    client_height = 0;
End

Process Main()
Begin

    get_desktop_size(& desktop_width,& desktop_height);
    get_window_size ( & window_width,  &window_height , & client_width ,  & client_height );
    set_mode (desktop_width-window_width+client_width,desktop_height-window_height+client_height,32);
    set_window_pos(0,0);

    write(0,desktop_width/2,desktop_height/2+30,0,"ESC to exit");
    while (!key(_ESC) )
        frame;
    end
End
```

Used in example: get_desktop_size(), **get_window_size**(), set_mode(), set_window_pos(), write(), key()

Template:Moduledocbox

100 Getenv

100.1 Definition

STRING getenv (<**STRING** variablename>)

Returns the value of an environment variable (like PATH).

100.2 Parameters

STRING variablename - The name of the variable to get the value of.

100.3 Returns

STRING : The value of the variable.

"" - The variable is invalid or empty.
!"" - The value of the variable.
Template:Moduledocbox

101 Glob

101.1 Definition

STRING glob (<**STRING** criteria>)

Gets a single filename or directoryname matching the criteria. If the same criteria is specified, it keeps returning new items on subsequent calls until it can't find any more, in which case it returns "". When different criteria are specified, the search is 'reset'. To reset the search without returning an item, use `""` as criteria.

After a call to glob(), the global struct fileinfo is filled with information about the last file/directory entry returned.

101.2 Parameters

STRING criteria - The search criteria to which the returned filenames apply. `""` to reset search.

101.3 Returns

STRING : Filename or directoryname

"" - No (new) file/directory entries.

!"" - The name of a file/directory entry matching the search criteria.

101.4 Notes

The search criteria can hold many criteria: the folder in which you want to look, simple requirements for filenames, such as extensions, and directory names. A few examples:

`*`	- Returns the filename of any file/directory in the current directory.
`*.dat`	- Returns the filename of any file/directory with extension `.dat` in the current directory.
`MyDir/*`	- Returns the filename of any file/directory in `MyDir\` relative to the current directory.
`C:/Program Files/*`	- Returns the filename of any file/directory in `C:\Program Files\*`.

101.5 Example

```
import "mod_dir"

Process Main()
Private
    String filename;
Begin

    // method one:
    Loop
        filename = Glob("levels/*.tul"); // Looks for a file in the relative folder
                                         // "levels" that has the file extension "tul".
        if(filename!="")
            load_level(filename); // load_level() would load the level file
        else
            break;
        end

    End

    //Reset search
    glob("");

    // method two:
    While( (filename = Glob("levels/*.tul")) != "" )
        load_level(filename);
    End

End
```

Both methods result in the same, but the second one is less code.

Template:Moduledocbox

102 Glyph get

102.1 Definition

INT get_glyph (<**INT** fontID> , <**INT** glyphID>)

Creates a new graphic containing the specified glyph of the specified font.

102.2 Parameters

INT fontID	- The fontID of the font the glyph is wanted.
INT glyphID	- The glyphID of the glyph in the specified font.

102.3 Returns

INT : GraphID

0	- Invalid font; Invalid glyph; could not create graphic;
>0	- The graphID of the graphic containing the glyph.

102.4 See also

- set_glyph()

Template:Moduledocbox

103 Glyph set

103.1 Definition

INT glyph_set (<**INT** fontID> , <**INT** glyphID> , <**INT** fileID> , <**INT** graphID>)

Sets the specified glyph of the specified font. The new glyph will be a copy of the specified graphic and thus it may be freed after the call.

Also called set_glyph().

103.2 Parameters

INT fontID	- The fontID of the font the glyph is to be set.
INT glyphID	- The glyphID of the glyph in the specified font.
INT fileID	- The fileID of the file that holds the graphic.
INT graphID	- The graphID of the graphic to be copied.

103.3 Returns

INT : false

103.4 See also

- glyph_get()

Template:Moduledocbox

104 Graphic info

104.1 Definition

INT graphic_info (<**INT** fileID> , <**INT** graphID> , <**INT** infotype>)

Gets some information about the graph specified.

104.2 Parameters

INT fileID	- The file that holds the graph.
INT graphID	- The graph to get information from.
INT infotype	- What type of information you want.

104.3 Returns

INT : Returns the information you want.
If the specified graph was invalid it returns 0.
If the specified infotype was not recognized it returns 1.

104.4 Example

```
Program keuken;
Local
    gxc;
    gyc;

Begin

    set_mode(640,480,16);

    graph=new_map(rand(50,150),rand(50,150),16); //makes a randomly proportioned red rectangle
    map_clear(0,graph,rgb(255,0,0));
    x=320;
    y=240;

    gxc=graphic_info(0,graph,G_X_CENTER);
    gyc=graphic_info(0,graph,G_Y_CENTER);  //finds the graphic's center coordinates

    map_put_pixel(0,graph,gxc,gyc,rgb(255,255,255)); //puts a white pixel in the center of the graphic

    Loop

        frame;
    End
End
```

Used in example: set_mode(), new_map(), map_clear(), map_put_pixel()

Template:Funcbox

105 Grayscale

105.1 Definition

INT grayscale (<**INT** fileID> , <**INT** graphID> , <**BYTE** method>)

This will convert the specified graphic by using the specified method; see notes for the details.

105.2 Parameters

INT fileID	- The fileID of the file that holds the graphics.
INT graphID	- The graphID of the graphic to convert.
BYTE method	- The method used (see notes).

105.3 Returns

INT

-1 - Invalid graphic.
1 - Success.

105.4 Notes

The exact formula is:

```
c = 0.3 * oldpixel_r + 0.59 * oldpixel_g + 0.11 * oldpixel_b
```

Method `0`:

```
for every pixel:
    newpixel_rgb = (c,c,c)
```

Method `1`:

```
for every pixel:
    newpixel_rgb = (c,0,0)
```

Method `2`:

```
for every pixel:
    newpixel_rgb = (0,c,0)
```

Method `3`:

```
for every pixel:
    newpixel_rgb = (0,0,c)
```

Method `4`:

```
for every pixel:
    newpixel_rgb = (c,c,0)
```

Method `5`:

```
for every pixel:
    newpixel_rgb = (c,0,c)
```

Method `6`:

```
for every pixel:
    newpixel_rgb = (0,c,c)
```

Other methodnumbers:

```
for every pixel:
    newpixel_rgb = oldpixel_rgb
```

Note that `rgbscale(0,map,1,1,1) = grayscale(0,map,0)` for a valid graphic (0,map).

Template:Moduledocbox

106 Is playing song

106.1 Definition

INT is_playing_song ( )

Checks to see if Bennu is playing a song file, started with play_song().

106.2 Returns

INT : Whether Bennu is playing a song at the moment of calling.

true	- Bennu is playing a song.
false	- Bennu is not playing a song.

106.3 Example

```
program music_example;
global
    my_song;
    playing;
    paused;
    faded_in;
    v;
begin
    set_mode(640,480,16);

    my_song=load_song("beat.ogg");

    write(0,320,30,4,"Use the keyboard to control the music playback.");
    write(0,320,50,4,"Key [ENTER] starts / stops the song.");
    write(0,320,60,4,"Key [SPACE] pauses / resumes the song.");
    write(0,320,70,4,"Key [0] through key [9] changes the song volume.");
    write(0,320,80,4,"Key [F] fades the song in or out.");

    write(0,320,120,5,"Playing: ");
    write_int(0,320,120,3,&playing);

    write(0,320,140,5,"Paused: ");
    write_int(0,320,140,3,&paused);

    write(0,320,160,5,"Faded in: ");
    write_int(0,320,160,3,&faded_in);

    write(0,320,180,5,"Volume: ");
    write_int(0,320,180,3,&v);

    v=128;
    faded_in=true;


    repeat
        if(key(_enter))
            if(is_playing_song())
                stop_song();
                playing=false;
            else
                play_song(my_song,1);
                playing=true;
            end
            while(key(_enter))frame;end
        end

        if(key(_space))
            if(paused)
                paused=false;
                resume_song();
            else
                paused=true;
                pause_song();
            end
            while(key(_space))frame;end
        end

        if(key(_f))
            if(faded_in)
                faded_in=false;
                fade_music_off(100);
```

```
            else
                faded_in=true;
                fade_music_in(my_song,1,100);
            end
            while(key(_f))frame;end
        end

        if(key(_0))v=0;end
        if(key(_1))v=14;end
        if(key(_2))v=28;end
        if(key(_3))v=43;end
        if(key(_4))v=57;end
        if(key(_5))v=71;end
        if(key(_6))v=85;end
        if(key(_7))v=100;end
        if(key(_8))v=114;end
        if(key(_9))v=128;end

        set_song_volume(v);

    frame;
    until(key(_esc))

    exit();
end
```

Used in example: key(), set_mode(), load_song(), write(), write_int(), pause_song(), play_song(), stop_song(), resume_song(), fade_music_in(), fade_music_off(), set_song_volume().

Template:Funcbox

107 Itoa

107.1 Definition

STRING itoa (<**INT** value>)

Returns a string containing a certain int value.

107.2 Parameters

INT value - The value the returned string will contain.

107.3 Returns

STRING : The string containing the specified value, including sign.

Template:Funcbox

108 Joy axes

1. REDIRECT joy_numaxes

109 Joy buttons

1. REDIRECT joy_numbuttons

110 Joy getaxis

110.1 Syntax

INT joy_getaxis ([<**INT** joy>], <**INT** axis>)

110.2 Description

Returns the selected joystick state for the given axis.

The JoyID is optional, if it is not present, the function uses the selected joystick. You can change the selected joystick with joy_select().

110.3 Parameters

[**INT** JoyID]	- The JoyID of the joystick.
INT axis	- The axis of the joystick.

110.4 Returns

INT : state for the given axis.

Template:Moduledocbox

111 Joy getball

111.1 Syntax

INT joy_getball ([<**INT** JoyID>] , <**POINTER** dx> , <**POINTER** dy>)

111.2 Description

Returns the state of the specfied ball on the current selected joystick.

The JoyID is optional, if it is not present, the function uses the selected joystick. You can change the selected joystick with joy_select().

111.3 Parameters

[**INT** JoyID]	- The JoyID of the joystick.
POINTER dx	- A pointer to the variable X of the ball.
POINTER dy	- A pointer to the variable Y of the ball.

111.4 Returns

INT : The state of the specfied ball on the current selected joystick.

111.5 Example

Template:Moduledocbox

112 Joy getbutton

112.1 Syntax

INT joy_getbutton ([<**INT** joy>], <**INT** button>)

112.2 Description

Returns the selected joystick state for the given button.

The JoyID is optional, if it is not present, the function uses the selected joystick. You can change the selected joystick with joy_select().

112.3 Parameters

[**INT** JoyID]	- The JoyID of the joystick.
INT button	- The button on the joystick.

112.4 Returns

INT : state for the given button.

Template:Moduledocbox

113 Joy gethat

113.1 Syntax

INT joy_gethat ([<**INT** JoyID>] , <**INT** hat>)

113.2 Description

Returns the current position of the digital POV hat of the controller pad selected.

The return values are:

Constant	- *Value*	- *Description*
JOY_HAT_CENTERED	- 0	- The hat is centered.
JOY_HAT_UP	- 1	- The hat is moved up.
JOY_HAT_RIGHT	- 2	- The hat is moved right.
JOY_HAT_DOWN	- 4	- The hat is moved down.
JOY_HAT_LEFT	- 8	- The hat is moved left.
JOY_HAT_RIGHTUP	- 3	- The hat is moved right and up.
JOY_HAT_RIGHTDOWN	- 6	- The hat is moved right and down.
JOY_HAT_LEFTUP	- 9	- The hat is moved left and up.
JOY_HAT_LEFTDOWN	- 12	- The hat is moved left and down.

You may notice that some are combinations of others. For example JOY_HAT_RIGHTUP == (JOY_HAT_RIGHT | JOY_HAT_UP). This is because the returned value has bit flags indicating four directions: up, down, left, right. These can be combined to make diagonal directions.

A value of -1 is returned when there is no hat or joystick detected.

The JoyID is optional, if it is not present, the function uses the selected joystick. You can change the selected joystick with joy_select().

113.3 Parameters

[**INT** JoyID]	- The JoyID of the joystick.
INT hat	- The number of the hat, starting at `0`

113.4 Returns

INT : The position of the POV hat.

113.5 Example

Template:Moduledocbox

114 Joy name

114.1 Syntax

STRING joy_name (<**INT** JoyID>)

114.2 Description

Returns the name of the specified joystick. This is a string describing the specified joystick, mostly used for the brand and model of the joystick.

114.3 Parameters

INT JoyID - The JoyID of the joystick.

114.4 Returns

STRING : The name of the joystick.

Template:Moduledocbox

115 Joy numaxes

115.1 Syntax

INT joy_numaxes ([<**INT** JoyID>])

115.2 Description

Returns the number of axes on the specified joystick. If no joystick is specified, the number of axes on the currently selected joystick will be returned.

The JoyID is optional, if it is not present, the function uses the selected joystick. You can change the selected joystick with joy_select().

Also called joy_axes().

115.3 Parameters

[**INT** JoyID] - The JoyID of the joystick.

115.4 Returns

INT : The number of axes.

Template:Moduledocbox

116 Joy numballs

116.1 Syntax

INT joy_numballs ([<**INT** JoyID>])

116.2 Description

Returns the number of balls on the specified joystick. If no joystick is specified, the number of balls on the currently selected joystick will be returned.

The JoyID is optional, if it is not present, the function uses the selected joystick. You can change the selected joystick with joy_select().

116.3 Parameters

[**INT** JoyID] - The JoyID of the joystick.

116.4 Returns

INT : The number of balls.

Template:Moduledocbox

117 Joy number

117.1 Syntax

INT joy_number ( )

117.2 Description

Returns the number of joysticks present in the system.

Also called joy_numjoysticks().

117.3 Returns

INT : The number of joysticks present in the system.

Template:Moduledocbox

118 Joy numbuttons

118.1 Syntax

INT joy_numbuttons ([<**INT** JoyID>])

118.2 Description

Returns the number of buttons on the specified joystick. If no joystick is specified, the number of buttons on the currently selected joystick will be returned.

The JoyID is optional, if it is not present, the function uses the selected joystick. You can change the selected joystick with joy_select().

Also called joy_buttons().

118.3 Parameters

[**INT** JoyID] - The JoyID of the joystick.

118.4 Returns

INT : The number of buttons.

Template:Moduledocbox

119 Joy numhats

119.1 Syntax

INT joy_numhats ([<**INT** JoyID>])

119.2 Description

Returns the number of hats on the specified joystick. If no joystick is specified, the number of hats on the currently selected joystick will be returned.

The JoyID is optional, if it is not present, the function uses the selected joystick. You can change the selected joystick with joy_select().

119.3 Parameters

[**INT** JoyID] - The JoyID of the joystick.

119.4 Returns

INT : The number of hats.

Template:Moduledocbox

120 Joy numjoysticks

1. REDIRECT joy_number

121 Joy select

121.1 Syntax

INT joy_select (<**INT** JoyID>)

121.2 Description

Select the joystick with id equals to JoyID.

121.3 Parameters

INT JoyID - The JoyID of the joystick.

121.4 Returns

INT : The ID of the selected joystick number.

Template:Moduledocbox

122 Key

122.1 Definition

INT key(<**INT** scancode>)

Checks if a certain key is being pressed.

122.2 Parameters

INT scancode - The scancode of the key to be checked.

122.3 Returns

INT : true/false: Whether the key is being pressed.

122.4 Notes

Take a look at the scancodes for a complete list.

122.5 Example

```
Program input_test;
Begin

    While( !key(_esc) )

        delete_text(ALL_TEXT);

        if( key(_left) && !key(_right) )
            write(0,160,120,4, "LEFT");
        end;

        if( key(_right) && !key(_left) )
            write(0,160,120,4, "RIGHT");
        end;

        frame;

    End;

    exit();

End
```

Used in example: delete_text(), write(), exit(), ALL_TEXT

This will output the words LEFT or RIGHT according to the keys you press, or it will quit the program once ESCAPE is pressed.

Template:Moduledocbox

123 Ksort

123.1 Definition

INT ksort (<**VARSPACE** array> , <**VARSPACE** sort-variable> , [<**INT** datacount>])

Sorts a certain array according to a sort-variable within the elements and by sorting a certain number of elements. By default the whole array is sorted.

If the array elements contain only one variable each or the first one is the sort-variable, sort() can be used. For more advanced sorting, look at quicksort().

123.2 Parameters

VARSPACE array	- The array to be sorted.
VARSPACE sort-variable	- The variable within each element to be used for the sorting.
[**INT** datacount]	- Number of elements to sort.

123.3 Returns

INT: Successrate

true	- Sorting succeeded.
false	- Sorting failed, probably the type of sort-variable isn't supported.

123.4 Example

```
Program sorting;

Type _player
    String name;
    int score;
End

Const
    maxplayers = 5;
Global
    _player player[maxplayers-1];
Begin

    // Insert some values
    player[0].name = "That one bad looking dude";
    player[1].name = "Ah pretty lame guy";
    player[2].name = "Some cool dude";
    player[3].name = "OMG ZOMG guy";
    player[4].name = "This person is ok";

    player[0].score = 70;
    player[1].score = 30;
    player[2].score = 80;
    player[3].score = 90;
    player[4].score = 50;


    // Show array
    say("-------------------- unsorted");
    for(x=0; x<maxplayers; x++)
        say(player[x].name + " - " + player[x].score);
    end

/* Sort by name ( quicksort() can't be used to sort Strings,
   as a String in Fenix is a pointer to the actual String,
   so it would sort the pointer addresses */

    // sort()
    sort(player); // sorts by name because name is the first variable in each element

    // Show array
    say("-------------------- name - sort()");
    for(x=0; x<maxplayers; x++)
        say(player[x].name + " - " + player[x].score);
    end

    // ksort()
    ksort(player,player[0].name,maxplayers);
```

```
    // Show array
    say("-------------------- name - ksort()");
    for(x=0; x<maxplayers; x++)
        say(player[x].name + " - " + player[x].score);
    end

/* Sort by score (sort() cannot be used here, because score is not the first variable) */

    // ksort()
    ksort(player,player[0].score,maxplayers);

    // Show array
    say("-------------------- score - ksort()");
    for(x=0; x<maxplayers; x++)
        say(player[x].name + " - " + player[x].score);
    end

    // quicksort()
    quicksort(&player[0],sizeof(_player),maxplayers,sizeof(String),sizeof(int),0);

    // Show array
    say("-------------------- score - quicksort()");
    for(x=0; x<maxplayers; x++)
        say(player[x].name + " - " + player[x].score);
    end

    // Wait until ESC is pressed
    Repeat
         frame;
    Until(key(_esc))

End
```

Used in example: say(), sort(), **ksort**(), quicksort(), type, array, pointer

124 LCD About

Up to LCD.DLL Functions

124.1 Definition

INT LCD_About ( )

Tells the user about LCD.DLL, using the Fenix console.

124.2 Returns

INT : The current LCD.DLL version (integer).

124.3 Example

```
Program example;
    include "LCD.fh";
Begin

    LCD_About();

    Loop
        frame;
    End

End
```

Template:Lcdfuncbox

125 LCD Close

Up to LCD.DLL Functions

125.1 Definition

INT LCD_Close (<**INT** device>)

Closes a connection to an LCDscreen.

125.2 Parameters

INT device - Number of the device.

125.3 Returns

INT : LCD.DLL Errorcodes

LCD_ERROR_NONE	- 0	- No error.
LCD_ERROR_SERVICEINACTIVE	- -14	- The service is inactive.
LCD_ERROR_INVALIDPARAMETER	- -22	- Invalid parameter.

Template:Lcdfuncbox

126 LCD Devices

Up to LCD.DLL Functions

126.1 Definition

INT LCD_Devices ( )

Returns the number of devices available.

126.2 Returns

INT : The number of LCD devices available.

0 - No devices or error.
!0 - The number of LCD devices available.
Template:Lcdfuncbox

127 LCD GetDepth

Up to LCD.DLL Functions

127.1 Definition

INT LCD_GetDepth (<**INT** device>)

Returns the colour depth in pixels of the LCDscreen of the specified device.

127.2 Parameters

INT device - Number of the device.

127.3 Returns

INT : The colour depth of the LCDscreen in pixels.

LCD_ERROR_INVALIDDEVICE	- -20	- Invalid device.
>=0		- The colour depth of the LCDscreen in pixels.

Template:Lcdfuncbox

128 LCD GetHeight

Up to LCD.DLL Functions

128.1 Definition

INT LCD_GetHeight (<**INT** device>)

Returns the height in pixels of the LCDscreen of the specified device.

128.2 Parameters

INT device - Number of the device.

128.3 Returns

INT : The height of the LCDscreen in pixels.

LCD_ERROR_INVALIDDEVICE	- -20	- Invalid device.
>=0		- The height of the LCDscreen in pixels.

Template:Lcdfuncbox

129 LCD GetNumButtons

Up to LCD.DLL Functions

129.1 Definition

INT LCD_GetNumButtons (<**INT** device>)

Returns the number of soft buttons of the LCDscreen of the specified device.

129.2 Parameters

INT device - Number of the device.

129.3 Returns

INT : The number of soft buttons of the LCDscreen.

LCD_ERROR_INVALIDDEVICE	- -20	- Invalid device.
>=0		- The number of soft buttons of the LCDscreen.

Template:Lcdfuncbox

130 LCD GetWidth

Up to LCD.DLL Functions

130.1 Definition

INT LCD_GetWidth (<**INT** device>)

Returns the width in pixels of the LCDscreen of the specified device.

130.2 Parameters

INT device - Number of the device.

130.3 Returns

INT : The width of the LCDscreen in pixels.

LCD_ERROR_INVALIDDEVICE	- -20	- Invalid device.
>=0		- The width of the LCDscreen in pixels.

Template:Lcdfuncbox

131 LCD Init

Up to LCD.DLL Functions

131.1 Definition

INT LCD_Init(<**STRING** connectionname>)

Initializes LCD.DLL.

131.2 Parameters

STRING connectionname - The name of the connection. This will be displayed when browsing LCD connections on the LCD.

131.3 Returns

INT : LCD.DLL Errorcodes

LCD_ERROR_NONE	- 0	- No error.
LCD_ERROR_RPCSSERVERUNAVAILABLE	- -10	- The RPC Server is unavailable.
LCD_ERROR_RPCXWRONGPIPEVERSION	- -11	- RPC X: wrong pipe version
LCD_ERROR_OLDWINVERSION	- -12	- Old Windows Version (must be 2000 or higher)
LCD_ERROR_NOSYSTEMRESOURCES	- -13	- No system resources available.
LCD_ERROR_SERVICEINACTIVE	- -14	- The service is inactive.
LCD_ERROR_ALREADYINITIALIZED	- -16	- LCD.DLL was already initialized.
LCD_ERROR_INVALIDPARAMETER	- -22	- Invalid parameter.
LCD_ERROR_FILENOTFOUND	- -23	- File not found.
LCD_ERROR_ALREADYEXISTS	- -24	- File or connection already exists.

Template:Lcdfuncbox

132 LCD IntVersion

Up to LCD.DLL Functions

132.1 Definition

INT LCD_IntVersion ( )

Returns the current LCD.DLL version.

132.2 Returns

INT : The current LCD.DLL version (int).

132.3 Example

```
Program example;
    include "LCD.fh";
Private
    int lcdver;
Begin

    lcdver = LCD_IntVersion();

    Loop
        frame;
    End

End
```

Template:Lcdfuncbox

133 LCD Open

Up to LCD.DLL Functions

133.1 Definition

INT LCD_Open (<**INT** device> , <**INT POINTER** buttons>)

Opens a connection to an LCDscreen.

133.2 Parameters

INT device	- Number of the device.
INT POINTER buttons	- Pointer to the buttonstatus holding integer.

133.3 Returns

INT : LCD.DLL Errorcodes

LCD_ERROR_NONE	- 0	- No error.
LCD_ERROR_SERVICEINACTIVE	- -14	- The service is inactive.
LCD_ERROR_INVALIDPARAMETER	- -22	- Invalid parameter.

Template:Lcdfuncbox

134 LCD Quit

Up to LCD.DLL Functions

134.1 Definition

INT LCD_Quit ( )

Deinitializes LCD.DLL.

134.2 Returns

INT : LCD.DLL Errorcode

LCD_ERROR_NONE - No error.
Template:Lcdfuncbox

135 LCD ReadButton

Up to LCD.DLL Functions

135.1 Definition

INT LCD_ReadButton (<**INT** device> , <**INT** button>)

Returns whether a button is being pressed on a device.

This also updates the button integer specified at LCD_Open().

135.2 Parameters

INT device - Number of the device.
INT button - Number of the button.

135.3 Returns

INT : LCD.DLL Errorcodes

TRUE	- 1	- The button is being pressed.
FALSE	- 0	- The button is not being pressed.
LCD_ERROR_SERVICEINACTIVE	- -14	- The service is inactive.
LCD_ERROR_INVALIDPARAMETER	- -22	- Invalid parameter.

Template:Lcdfuncbox

136 LCD ReadButtons

Up to LCD.DLL Functions

136.1 Definition

INT LCD_ReadButtons (<**INT** device>)

Updates the integer holding the buttonstatus.

136.2 Parameters

INT device - Number of the device.

136.3 Returns

INT : LCD.DLL Errorcodes

LCD_ERROR_NONE	- 0	- No error.
LCD_ERROR_SERVICEINACTIVE	- -14	- The service is inactive.
LCD_ERROR_DEVICENOTCONNECTED	- -21	- Device not connected.
LCD_ERROR_INVALIDPARAMETER	- -22	- Invalid parameter.

Template:Lcdfuncbox

137 LCD SetBitmap

Up to LCD.DLL Functions

137.1 Definition

INT LCD_SetBitmap (<**INT** device> , <**INT** priority> , <**BYTE POINTER** map>)

Sets the bitmap of the device to a certain map with a certain priority.

137.2 Parameters

INT device	- Number of the device.
INT priority	- The priority of the bitmap (see notes).
BYTE POINTER map	- Pointer to a 160x43 byte array.

137.3 Returns

INT : LCD.DLL Errorcodes

LCD_ERROR_NONE	- 0	- No error.
LCD_ERROR_SERVICEINACTIVE	- -14	- The service is inactive.
LCD_ERROR_DEVICENOTCONNECTED	- -21	- Device not connected.
LCD_ERROR_INVALIDPARAMETER	- -22	- Invalid parameter.

137.4 Notes

- Multiple priorities can be used:

LCD_PRIORITY_IDLE_NO_SHOW	- Lowest priority, disable displaying. Use this priority when you don?t have anything to show.
LCD_PRIORITY_BACKGROUND	- Priority used for low priority items.
LCD_PRIORITY_NORMAL	- Normal priority, to be used by most applications most of the time.
LCD_PRIORITY_ALERT	- Highest priority. To be used only for critical screens, such as "your CPU temperature is too high"

- One can use map_buffer() to obtain a byte pointer to a map. Make sure this map is of size 160x43 and it has a color depth of 8.

Template:Lcdfuncbox

138 LCD Version

Up to LCD.DLL Functions

138.1 Definition

STRING LCD_Version ( )

Returns the current LCD.DLL version.

138.2 Returns

STRING : The current LCD.DLL version (string).

138.3 Example

```
Program example;
    include "LCD.fh";
Private
    string lcdver;
Begin

    lcdver = LCD_Version();

    Loop
        frame;
    End

End
```

Template:Lcdfuncbox

139 Lcase

139.1 Definition

STRING lcase (<**STRING** str>)

Returns a string identical to a certain string, with the exception that all uppercase characters are replaced by their lowercase counterparts.

139.2 Parameters

STRING str - The string in "normal"-form.

139.3 Returns

STRING : The string in "lowercase"-form.

Template:Funcbox

140 Len

140.1 Definition

INT len (<**STRING** str>)

Returns the length, the number of characters, of a certain string.

140.2 Parameters

STRING str - The string of which the length will be returned.

140.3 Returns

INT : The length of the specified string.

Template:Funcbox

141 Let me alone

141.1 Definition

INT let_me_alone ( )

Kills all processes except the calling one.

To kill only one process, use signal().

141.2 Returns

INT : true

141.3 Example

```
Const
    screen_width = 320;
    screen_height = 200;
    screen_depth = 8;
End

/**
 * Description
 *   Generates an error. Puts the error in the console and stdout.txt and shows it onscreen
 *   for certain time. Immediately kills all other processes and quits the program after a
 *   certain time.
 *
 * Parameters
 *   String message   - The error message.
 *   int delay        - The time to display the error onscreen and after which the program will quit.
 *                      In 1/100seconds.
 *
 * Returns
 *   0 - Success.
 */
Process error(String message,int delay)
Begin

    // Put the error message in the console and in stdout.txt
    say("[ERROR] " + message);

    // Show the error message onscreen, the size adjust for the screen width
    set_text_color(rgb(255,0,0));
    graph = write_in_map(0,message,4);
    size = 100*(screen_width-10)/graphic_info(0,graph,G_WIDTH);
    x = screen_width/2;
    y = screen_height/2;

    // Kill all other processes
    let_me_alone();

    // Wait the specified time
    timer[0] = 0;
    Repeat
        frame;
    Until(timer[0]>delay)

    // Unload the used graph
    unload_map(0,graph);

    return 0;

End

Process Main();
Begin

    // Set the screen mode
    set_mode(screen_width,screen_height,screen_depth);

    // Generate an error
    error("ERROR, QUITTING IN 2 SECONDS",200);

    Repeat
        frame;
    Until(key(_esc))
```

```
End
```

Template:Funcbox

142 Ln

Up to Log.dll functions

142.1 Definition

FLOAT ln (<**FLOAT** n>)

Returns the natural logarithm of number *n* (logarithm with base ? (Euler)).

142.2 Parameters

FLOAT n - The number that will be used for the logarithm.

142.3 Returns

FLOAT : The natural logarithm of *n*.

142.4 Example

```
Import "log.dll";

Global
    float logarithm=0.0;
End

Process main()
Begin
    write_float(0,160,100,4,&logarithm);

    While(not(key(_ESC)))
        If (key(_1)) logarithm=ln(13.37); End

        Frame;
    End
End
```

Used in example: import, write_float(), key(), **ln**()

Template:Logfuncbox

143 Load

143.1 Definition

INT load (<**STRING** filename> , <**VARSPACE** data>)

Loads the data read from the specified file into the specified variable.

143.2 Parameters

STRING filename	- The name of the file to be loaded.
VARSPACE data	- The variable (of any datatype) in which the data read from the file will be loaded.

143.3 Returns

INT : The number of bytes read from the file.

143.4 Notes

To check whether a file exists before it is loaded, file_exists() can be used.

143.5 Example

```
Program test;
Global
    struct My_struct
        Level_number;
        string Map_name;
    End
Begin
    If (file_exists("myfile.sav"))
        Load("myfile.sav",My_struct);            // Content from myfile.sav is loaded into My_struct
        Write(0,10,10,0,My_struct.level_number); // A variable from the loaded struct is shown on screen
        Write(0,10,20,0,My_struct.map_name);     // Another variable loaded is shown on screen
    Else
        Write(0,10,10,0,"File couldn't be loaded, it doesn't exist.");
    End

    While (!key(_esc))
        Frame;
    End
End
```

Used in example: file_exists(), **load**(), write(), key()

Template:Funcbox

144 Load song

144.1 Definition

INT load_song (<**STRING** filename>)

Loads a song for later use with play_song().

There are multiple filetypes you can load using this function. Included are:

- OGG Vorbis (.ogg). Good quality for songs and doesn't take too much space (it's similar to *.mp3).
- MIDI (.mid). Takes very low space, but it's limited to samples of the soundcard.
- Modules (.xm, .it, .s3m, .mod). Like MIDI, but Modules also contain the samples themselves.

144.2 Parameters

STRING filename - The music file to be loaded, including a possible path.

144.3 Returns

INT : SongID

-1 - Could not load music file (errormessage in console).
>=0 - The SongID.

144.4 Example

```
Program example;
Private
    int song;
Begin
    song = load_song("my_song.ogg");
    play_song(song,0);
    Repeat
        frame;
    Until(key(_ESC))
OnExit
    unload_song(song);
End
```

Used in example: play_song(), key(), unload_song()

Template:Funcbox

145 Load wav

145.1 Definition

INT load_wav (<**STRING** filename>)

Loads a sound effect in the WAV format for later use with play_wav().

145.2 Parameters

STRING filename - The WAV file to be loaded, including a possible path.

145.3 Returns

INT : WaveID

-1 - Error: sound inactive; opening file
0 - Could not load wave file.
>0 - The WaveID.

145.4 Example

```
Program
Private
    int wave;
Begin
    wave = load_wav("my_wav.wav");
    play_wav(wave,0);
    Loop
        frame;
    End
End
```

Used in example: play_wav()

Template:Funcbox

146 Log

Up to Log.dll functions

146.1 Definition

FLOAT log (<**FLOAT** n> , [<**FLOAT** b>])

Returns a logarithm of number *n* with base *b*. If *b* is not specified, the number 10 will be used as base.

146.2 Parameters

FLOAT n	- The number that will be used for the logarithm.
FLOAT base	- The base that will be used for the logarithm.

146.3 Returns

FLOAT : The logarithm of *n*.

146.4 Example

```
Import "log.dll";

Global
    float logarithm=0.0;
End

Process main()
Begin
    write_float(0,160,100,4,&logarithm);

    While(not(key(_esc)))
        If (key(_1)) logarithm=log(1000,9); End
        If (key(_2)) logarithm=log(1000); End

        Frame;
    End
End
```

Used in example: import, write_float(), key(), **log**()

Template:Logfuncbox

147 Log2

Up to Log.dll functions

147.1 Definition

FLOAT log2 (<**FLOAT** n>)

Returns a logarithm of number *n* with base 2.

147.2 Parameters

FLOAT n - The number that will be used for the logarithm.

147.3 Returns

FLOAT : The logarithm of *n* with base 2.

147.4 Example

```
Import "log.dll";

Global
    float logarithm=0.0;
End

Process main()
Begin
    write_float(0,160,100,4,&logarithm);

    While(not(key(_esc)))
        If (key(_1)) logarithm=log2(1024); End

        Frame;
    End
End
```

Used in example: import, write_float(), key(), **log2**()

Template:Logfuncbox

148 Lpad

148.1 Definition

STRING lpad(<**STRING** str> , <**INT** length>)

Returns the string *str*, padding (adding spaces to) the front of the string if needed to make *str* of length *length*. The original string will remain unchanged.

If *length* is smaller or equal to the length of *str*, the returned string is *str*.

148.2 Parameters

STRING str	- The string to pad (add spaces to).
INT length	- The minimal length of the returned string.

148.3 Returns

STRING: padded string

148.4 Example

```
import "mod_string"
import "mod_say"

Process Main()
Private
    string ABC = "ABC";
    string _ABC;
    string ABC__;
Begin

    ABC = lpad(ABC,2);
    _ABC = lpad(ABC,4);
    ABC__ = rpad(ABC,5);

    say('ABC = "' + ABC + '"');
    say('_ABC = "' + _ABC + '"');
    say('ABC__ = "' + ABC__ + '"');

End
```

Used in example: say(), **lpad**(), rpad()

Result:

```
ABC = "ABC"
_ABC = " ABC"
ABC__ = "ABC  "
```

Template:Moduledocbox

149 Map block copy

149.1 Definition

INT map_block_copy (<**INT** fileID> , <**INT** destinationGraphID> , <**INT** destinationX> , <**INT** destinationY> , <**INT** originGraphID> , <**INT** x> , <**INT** y> , <**INT** width> , <**INT** height>, <**INT** blitflags>)

Draws (blits) a rectangular block from one graphic onto another graphic.

If the entire graphic is to be blitted, map_put() or map_xput() can be used.

149.2 Parameters

INT fileID	- The fileID of the file that holds the destination and origin graphics.
INT destinationGraphID	- The graphID of the graphic to draw on.
INT destinationX	- Where on the destination graph's x-axis to put the block.
INT destinationY	- Where on the destination graph's y-axis to put the block.
INT originGraphID	- The graphID of the graphic to draw with.
INT x	- The x-coordinate of the upperleft corner of the origin block.
INT y	- The y-coordinate of the upperleft corner of the origin block.
INT width	- The width of the block in pixels.
INT height	- The height of the block in pixels.
INT blitflags	- What blit flags to draw the graphic with.

149.3 Returns

INT : true

149.4 Notes

Blit flags can be used to give the drawing (blitting) a special effect.

149.5 Errors

Invalid origin graph	- The origin graph is invalid.
Invalid destination graph	- The destination graph is invalid.
Unsupported color depth	- The origin graphic's color depth is greater than the destination graph's.

Template:Funcbox

150 Map clear

150.1 Definition

INT map_clear (<**INT** fileID> , <**INT** graphID> , <**WORD** color>)

Clears a certain graph to a certain color.

150.2 Parameters

INT fileID	- The file that holds the graphic.
INT graphID	- The graphic to clear.
WORD color	- The color used to clear.

150.3 Returns

INT : true

150.4 Notes

Instead of using **map_clear**() to clear the screen background, clear_screen() is recommended as it is a little bit faster.

150.5 Errors

Unsupported color depth	- The specified graph has a not supported color depth.

150.6 Example

```
Program a_map_is_born;
Private
    int map;
Begin

    // Create a new graph of size 100x100 and color depth of 8bit
    map = new_map(100,100,8);

    // Clear the map red
    map_clear(0,map,rgb(255,0,0));

    // Put it in the center of the screen
    put(0,map,160,100);

    Loop
        frame;
    End

End
```

Used in example: new_map(), put()

This will result in something like:
http://wwwhome.cs.utwente.nl/~bergfi/fenix/wiki/new_map.PNG

Template:Funcbox

151 Map clone

151.1 Definition

INT map_clone (<**INT** fileID> , <**INT** graphID>)

Clones a certain graphic and puts it in the system file.

151.2 Parameters

INT fileID	- The fileID of the file holding the graphic.
INT graphID	- The graphID of the graphic to be cloned.

151.3 Returns

INT : GraphID

0	- Invalid graphic specified;
>0	- The graphID of the clone graphic.

151.4 Errors

Unsupported color depth	- The specified graph has a not supported color depth. *(Console)*
Insufficient memory	- There is insufficient memory available. This error doesn't occur often. *(Console)*

Template:Funcbox

152 Map exists

152.1 Syntax

INT map_exists (<**INT** fileID> , <**INT** graphID>)

152.2 Description

Checks if a graphic exists in the specified File with the specified graphID.

152.3 Parameters

INT fileID	- The fileID of the file that holds the graphic (or not).
INT graphID	- The graphID.

152.4 Returns

INT : Whether the graphic exists

false	- The specified graphic does not exist.
true	- The specified graphic exists.

Template:Moduledocbox

153 Map get pixel

153.1 Definition

INT map_get_pixel (<**INT** fileID> , <**INT** graphID> , <**INT** x> , <**INT** y>)

Returns the color on the specified graphic of the specified pixel.

153.2 Parameters

INT fileID	- The fileID of the file that holds the graphic.
INT graphID	- The graphID of the graphic to get the pixel from.
INT x	- The X-coordinate of the pixel the color is wanted.
INT y	- The Y-coordinate of the pixel the color is wanted.

153.3 Returns

INT : The color

-1 - Error: invalid map; invalid; invalid pixel; invalid color depth of map. Note: in 32bit graphs this can be a color too.
!-1 - The color.
Template:Moduledocbox

154 Map load

154.1 Definition

INT map_load (<**STRING** filename>)

Creates a new graphic, using the specified MAP file as contents and puts it in the system file. Returns the graphID of the created graphic. The color depth of the created graphic will be the same as the loaded MAP file.

Also called load_map().

154.2 Parameters

STRING filename - The name of the MAP file to be loaded, including a possible path.

154.3 Returns

INT : graphID

0 - There was an error loading the file.
>0 - The graphID of the newly created graphic.
Template:Moduledocbox

155 Map name

155.1 Definition

STRING map_name (<**INT** fileID> , <**INT** graphID>)

Retrieves the name of an in-game graphic.

155.2 Parameters

INT fileID	- The fileID of the file that holds the graphic.
INT graphID	- The graphID of the graphic.

155.3 Returns

STRING : Name of the graphic

Template:Funcbox

156 Map new

156.1 Definition

INT map_new (<**INT** width> , <**INT** height> , <**INT** depth>)

Creates a new graphic, sets the color of all pixels to 0 (transparent) and puts it in the system file.

Also called new_map().

156.2 Parameters

INT width	- The width of the to be created graph in pixels.
INT height	- The height of the to be created graph in pixels.
INT depth	- The color depth of the to be created graph in bits

156.3 Returns

INT : GraphID

0	- There was an error.
>0	- The graphID of the newly created graphic.

156.4 Errors

Unsupported color depth	- The specified color depth is not supported. *(Console)*
Insufficient memory	- There is insufficient memory available. This error doesn't occur often. *(Console)*

156.5 Example

```
import "mod_map"
import "mod_screen"
import "mod_key"

Process Main()
Private
    int map;
Begin

    // Create a new graph of size 100x100 and color depth of 8bit
    map = map_new(100,100,8);

    // Clear the map red
    map_clear(0,map,rgb(255,0,0));

    // Put it in the center of the screen
    put(0,map,160,100);

    Repeat
        frame;
    Until(key(_ESC))

End
```

Used in example: **map_new**(), map_clear(), put(), key()

This will result in something like:
Template:Image

Template:Moduledocbox

157 Map put

157.1 Syntax

INT map_put (<**INT** fileID> , <**INT** destinationGraphID> , <**INT** originGraphID> , <**INT** x> , <**INT** y>)

157.2 Description

Draws (blits) a graph onto another graph.

If more advanced parameters are needed, map_xput() can be used. If a graph from one file needs to be drawn onto another graph in a different file, or a separate width and height scaling is needed, map_xputnp() can be used.

157.3 Parameters

INT fileID	- The fileID of the file that holds the graphics.
INT destinationGraphID	- The graphID of the graphic to draw on.
INT originGraphID	- The graphID of the graphic to draw with.
INT x	- Where on the destination graphic's x-axis to put the graph.
INT y	- Where on the destination graphic's y-axis to put the graph.

157.4 Returns

INT : true

157.5 Notes

The x and y parameters denote where to draw the graph, that is, where the center of the to be drawn graph will be.

157.6 Errors

Invalid origin graph	- The origin graph is invalid.
Invalid destination graph	- The destination graph is invalid.
Unsupported color depth	- The origin graph's color depth is greater than the destination graph's.

157.7 Example

```
import "mod_video"
import "mod_map"
import "mod_wm"

Process Main()
Private
    int destgraph;
    int origgraph1;
    int origgraph2;
Begin

    // Set the mode to 16bit and some resolution
    set_mode(320,200,16);

    // Makes the destination graphic a red square
    destgraph=new_map(100,100,16);
    map_clear(0,destgraph,rgb(255,0,0));

    // Makes the first origin graphic a green square
    origgraph1=new_map(100,100,16);
    map_clear(0,origgraph1,rgb(0,255,0));

    // Makes the second origin graphic a blue square
    origgraph2=new_map(100,100,16);
    map_clear(0,origgraph2,rgb(0,0,255));

    // Draws the blue and green squares at a random position on the red square
    map_put(0,destgraph,origgraph1,rand(0,100),rand(0,100));
    map_put(0,destgraph,origgraph2,rand(0,100),rand(0,100));
```

```
    // Shows the final graph
    put(0,destgraph,160,100);

    Repeat
        frame;
    Until(exit_status)

End
```

Used in example: set_mode(), new_map(), map_clear(), **map_put**(), put(), key(), exit_status

This will result in something like:
Template:Image

Template:Moduledocbox

158 Map put pixel

158.1 Definition

INT map_put_pixel (<**INT** fileID> , <**INT** graphID> , <**INT** x> , <**INT** y> , <**INT** color>)

Draws a single colored pixel on a graph.

158.2 Parameters

INT fileID	- The file that holds the graph.
INT graphID	- The graph to draw on.
INT x	- Where on the graph's x-axis to put the pixel.
INT y	- Where on the graph's y-axis to put the pixel.
INT color	- What color to draw.

158.3 Returns

INT : true

158.4 Errors

Invalid map	- Map doesn't exist.
Unsupported color depth	- Destination graph is of an unsupported colordepth.

158.5 Example

```
import "mod_video"
import "mod_map"
import "mod_screen"
import "mod_wm"
import "mod_draw"

Process Main()
Private
    int map;
    int i;
Begin

    // Set the mode to 16bit and some res
    set_mode(320,200,16);

    // Create a blue-ish square map
    map = new_map(100,100,16);
    map_clear(0,map,rgb(50,100,150));

    // Puts 100 yellow-ish pixels in random places in the graphic
    for(i=0; i<100; i++)
        map_put_pixel(0,map,rand(0,100),rand(0,100),rgb(255,255,55));
    end

    // Puts the map in the center of the screen
    put(0,map,160,100);

    Repeat
        frame;
    Until(exit_status)

End
```

Used in example: set_mode(), new_map(), map_clear(), **map_put_pixel**(), rand(), put()

Template:Moduledocbox

159 Map save

159.1 Definition

INT map_save (<**INT** fileID> , <**INT** graphID> , <**STRING** filename>)

Saves the specified graphic as *filename* with the format MAP.

Also called save_map().

159.2 Parameters

INT fileID	- The fileID of the file that holds the graphic.
INT graphID	- The graphID of the graphic to unload.
STRING filename	- The name of the MAP file to be saved, including a possible path.

159.3 Returns

INT : Successrate

false	- An error: Invalid graphic; could not open file.
true	- Graphic saved.

Template:Moduledocbox

160 Map unload

160.1 Definition

INT map_unload (<**INT** fileID> , <**INT** graphID>)

Frees the memory used by the specified graphic. The associated (fileID,graphID) combination is no longer valid afterwards.

Also called unload_map().

160.2 Parameters

INT fileID	- The fileID of the file that holds the graphic.
INT graphID	- The graphID of the graphic to unload.

160.3 Returns

INT : Successrate

false	- Invalid graphic.
true	- Graphic unloaded.

Template:Moduledocbox

161 Map xput

161.1 Definition

INT map_xput (<**INT** fileID> , <**INT** destinationGraphID> , <**INT** originGraphID> , <**INT** x> , <**INT** y> , <**INT** angle> , <**INT** size> , <**INT** blitflags>)

Draws (blits) a graphic onto another graphic.

If the advanced parameters aren't needed, map_put() can be used. If a graphic from one file needs to be drawn onto another graphic in a different file, or a separate width and height scaling is needed, map_xputnp() can be used.

161.2 Parameters

INT fileID	- The fileID of the file that holds the graphics.
INT destinationGraphID	- The graphID of the graphic to draw on.
INT originGraphID	- The graphID of the graphic to draw with.
INT x	- Where on the destination graphic's x-axis to put the graphic.
INT y	- Where on the destination graphic's y-axis to put the graphic.
INT angle	- What angle to draw the graphic at.
INT size	- What size to draw the graphic at.
INT blitflags	- What blit flags to draw the graphic with.

161.3 Returns

INT : true

161.4 Notes

The x and y parameters denote where to draw the graphic, that is, where the center of the to be drawn graphic will be. Blit flags can be used to give the drawing (blitting) a special effect.

When angle is 0 and size is 100, the speed is greater, because the graph doesn't need rotating or scaling.

161.5 Errors

Unsupported color depth	- The origin graphic's color depth is greater than the destination graphic's.

161.6 Example

```
import "mod_video"
import "mod_map"
import "mod_wm"

Process Main()
Private
    int destgraph;
    int origgraph;
Begin

    // Set the mode to 16bit and some resolution
    set_mode(320,200,16);

    // Makes the destination graphic a red square
    destgraph=new_map(100,100,16);
    map_clear(0,destgraph,rgb(255,0,0));

    // Makes the origin graphic a blue square
    origgraph=new_map(100,100,16);
    map_clear(0,origgraph,rgb(0,0,255));

    // Draws the blue square on the center of the red square transparently,
    // at a random angle and a random size
    map_xput(0,destgraph,origgraph,50,50,rand(-180000,180000),rand(50,150),4);
    map_xput(0,destgraph,origgraph,50,50,rand(-180000,180000),rand(50,150),4);

    // Shows the final graph
    put(0,destgraph,160,100);
```

```
    Repeat
        frame;
    Until(exit_status)

End
```

Used in example: set_mode(), new_map(), map_clear(), **map_xput**(), put(), key(), exit_status

This will result in something like: Template:Image

Template:Moduledocbox

162 Map xputnp

162.1 Definition

INT map_xputnp (<**INT** destinationFileID> , <**INT** destinationGraphID> , <**INT** originFileID> , <**INT** originGraphID> , <**INT** x> , <**INT** y> , <**INT** angle> , <**INT** scale_x> , <**INT** scale_y> , <**INT** blitflags>)

Draws (blits) a graphic onto another graphic.

If the advanced parameters aren't needed, map_put() or map_xput() can be used.

162.2 Parameters

INT destinationFileID	- The fileID of the file that holds the destination graphic.
INT destinationGraphID	- The graphID of the graphic to draw on.
INT originFileID	- The fileID of the file that holds the origin graphic.
INT originGraphID	- The graphID of the graphic to draw with.
INT x	- Where on the destination graph's x-axis to put the graphic.
INT y	- Where on the destination graph's y-axis to put the graphic.
INT angle	- What angle to draw the graphic at.
INT scale_x	- What size to draw the origin graphic's x-axis at (also see size_x)
INT scale_y	- What size to draw the origin graphic's y-axis at (also see size_y).
INT blitflags	- What blit flags to draw the graphic with.

162.3 Returns

INT : true

162.4 Notes

The x and y parameters denote where to draw the graphic, that is, where the center of the to be drawn graphic will be. Blit flags can be used to give the drawing (blitting) a special effect.

When angle is 0 and size is 100, the speed is greater, because the graphic doesn't need rotating or scaling.

162.5 Errors

Unsupported color depth	- The origin graphic's color depth is greater than the destination graph's.

162.6 Example

```
import "mod_video"
import "mod_map"
import "mod_wm"

Process Main()
Global
    int destgraph;
    int origgraph;
Begin

    // Set the mode to 16bit and some resolution
    set_mode(320,200,16);

    // Makes the destination graphic a red square
    destgraph=new_map(100,100,16);
    map_clear(0,destgraph,rgb(255,0,0));

    // Makes the origin graphic a blue square
    origgraph=new_map(100,100,16);
    map_clear(0,origgraph,rgb(0,0,255));

    // Draws the blue square on the center of the red square transparently,
    // at a random angle and a random size
    map_xputnp(0,destgraph,0,origgraph,50,50,rand(-180000,180000),rand(50,150),rand(50,150),4);
    map_xputnp(0,destgraph,0,origgraph,50,50,rand(-180000,180000),rand(50,150),rand(50,150),4);
```

```
    // Shows the final graph
    put(0,destgraph,160,100);

    Repeat
         frame;
    Until(exit_status)

End
```

Used in example: set_mode(), new_map(), map_clear(), **map_xputnp**(), put(), exit_status, blit flags

This will result in something like: Template:Image

Template:Moduledocbox

163 Memcmp

163.1 Definition

INT memcmp (<**VOID POINTER** ptr1> , <**VOID POINTER** ptr2> , <**INT** number>)

Compares the first *number* bytes of the block of memory pointed by *ptr1* to the first *number* bytes pointed by *ptr2*, returning zero if they all match or a value different from zero representing which is greater if they do not.

163.2 Parameters

VOID POINTER ptr1	- Pointer to a block of memory
VOID POINTER ptr2	- Pointer to a block of memory.
INT number	- The number of bytes to be checked.

163.3 Returns

INT : Difference

0	- The blocks of memory are identical.
>0	- The first differing byte in both memory blocks has a greater value in *ptr1*.
<0	- The first differing byte in both memory blocks has a greater value in *ptr2*.

A byte ranges from 0 to 255, meaning 189 is a greater value than 105.

163.4 Example

```
Program example;
Const
    elements = 10;
End
Private
    byte pointer pbyte1;
    byte pointer pbyte2;
    int result;
End
Begin

    // Allocate memory
    pbyte1 = alloc(elements);
    pbyte2 = alloc(elements);

    // Make the blocks the same and compare
    memset(pbyte1,3,elements);
    memset(pbyte2,3,elements);
    result = memcmp(pbyte1,pbyte2,elements); // You can also compare the first 5 bytes,
                                             // or the first elements/2 bytes, it
                                             // depends on what you want.
    say("Memcmp 1: " + result);

    // Make the first block have a greater value and compare
    pbyte1[0] = 4;
    result = memcmp(pbyte1,pbyte2,elements);
    say("Memcmp 2: " + result);

    // Make the blocks the same and compare
    pbyte2[0] = 4;
    result = memcmp(pbyte1,pbyte2,elements);
    say("Memcmp 3: " + result);

    // Make the first block have a greater value and compare
    pbyte2[1] = 5;
    result = memcmp(pbyte1,pbyte2,elements);
    say("Memcmp 4: " + result);

    Repeat
        frame;
    Until(key(_esc))

    // Free the used memory
    free(pbyte1);
    free(pbyte2);

End
```

Used in example: alloc(), memset(), **memcmp**(), say(), free(), pointer

Template:Funcbox

164 Memcopy

164.1 Syntax

INT memcopy (<**VOID POINTER** destination> , <**VOID POINTER** origin> , <**INT** size>)

164.2 Description

Copies a certain number of bytes from one point in memory to another.

Difference between memmove() and **memcopy**() is that the first one can be used if the destination section and origin section overlap. With **memcopy**(), this can go wrong, though some systems make **memcopy**() safe too.

164.3 Parameters

VOID POINTER destination	- Pointer to the first byte of the destination.
VOID POINTER origin	- Pointer to the first byte of the origin.
INT size	- The size of the to be copied memory in bytes.

164.4 Returns

INT : true

164.5 Example

```
import "mod_mem"
import "mod_say"

Const
    elements = 5;
End

Process Main()
Private
    byte bytearray[elements-1];
    byte* pbyte;
    int i;
End
Begin

    // Allocate memory
    pbyte = alloc(elements);

    // Set numbers
    for(i=0; i<elements; i++)
        bytearray[i] = i;
    end

    // Copy bytes to bytearray
    memcopy(pbyte,&bytearray[0],elements);

    // Show numbers
    for(i=0; i<elements; i++)
        say("byte["+i+"] = " + pbyte[i]);
    end

OnExit

    // Free the used memory
    free(pbyte);

End
```

Used in example: alloc(), **memcopy**(), say(), free(), array, pointer

Template:Moduledocbox

165 Memmove

165.1 Definition

INT memmove (<**VOID POINTER** destination> , <**VOID POINTER** origin> , <**INT** size>)

Copies a certain number of bytes from one point in memory to another.

Difference between **memmove**() and memcopy() is that the first one can be used if the destination section and origin section overlap. With memcopy(), this can go wrong, though some systems make memcopy() safe too.

165.2 Parameters

VOID POINTER destination	- Pointer to the first byte of the destination.
VOID POINTER origin	- Pointer to the first byte of the origin.
INT size	- The size of the to be copied memory in bytes.

165.3 Returns

INT : true

165.4 Example

```
import "mod_mem"
import "mod_say"

Const
    elements = 5;
End

Process Main()
Private
    byte bytearray[elements-1];
    byte* pbyte;
    int i;
End
Begin

    // Allocate memory
    pbyte = alloc(elements);

    // Set numbers
    for(i=0; i<elements; i++)
        bytearray[i] = i;
    end

    // Copy bytes to bytearray
    memmove(pbyte,&bytearray[0],elements);

    // Show numbers
    for(i=0; i<elements; i++)
        say("byte["+i+"] = " + pbyte[i]);
    end

OnExit

    // Free the used memory
    free(pbyte);

End
```

Used in example: alloc(), **memmove**(), say(), free(), array, pointer

Template:Moduledocbox

166 Memory free

166.1 Syntax

INT memory_free ( )

166.2 Description

Returns the free memory total in bytes.

166.3 Returns

INT : Free memory total in bytes.

166.4 Example

```
import "mod_mem"
import "mod_say"

Process Main()
Begin

    say("Total memory: " + memory_total());
    say("Free memory:  " + memory_free() );

End
```

Used in example: say(), memory_total(), **memory_free**()

Template:Moduledocbox

167 Memory total

167.1 Definition

INT memory_total ( )

Returns the memory total in bytes.

167.2 Returns

INT : Memory total in bytes.

167.3 Example

```
import "mod_mem"
import "mod_say"

Process Main()
Begin

    say("Total memory: " + memory_total());
    say("Free memory:  " + memory_free() );

End
```

Used in example: say(), **memory_total**(), memory_free()

Template:Moduledocbox

168 Memset

168.1 Syntax

INT memset (<**VOID POINTER** data> , <**BYTE** value> , <**INT** bytes>)

168.2 Description

Sets all bytes in the specified memory block to the specified value.

168.3 Parameters

VOID POINTER data	- Pointer to the block of bytes in memory
BYTE value	- Value to set all bytes to.
INT bytes	- Number of bytes to change the value of.

168.4 Returns

INT : true

168.5 Example

See alloc().

Also useful in conjunction with map_buffer() with 8bit maps. (*Example needed.*)

Template:Moduledocbox

169 Memseti

169.1 Syntax

INT memseti (<**VOID POINTER** data> , <**INT** value> , <**INT** ints>)

169.2 Description

Sets all ints in the specified memory block to the specified value.

169.3 Parameters

VOID POINTER data	- Pointer to the block of ints in memory.
INT value	- Value to set all ints to.
INT ints	- Number of ints to change the value of.

169.4 Returns

INT : true

169.5 Example

See alloc().

Also useful in conjunction with map_buffer() with 32bit maps. (*Example needed.*)

Template:Moduledocbox

170 Memsetw

170.1 Syntax

INT memsetw (<**VOID POINTER** data> , <**WORD** value> , <**INT** words>)

170.2 Description

Sets all words in the specified memory block to the specified value.

170.3 Parameters

VOID POINTER data	- Pointer to the block of words in memory.
WORD value	- Value to set all words to.
INT words	- Number of words to change the value of.

170.4 Returns

INT : true

170.5 Example

See alloc().

Also useful in conjunction with map_buffer() with 16bit maps. (*Example needed.*)

Template:Moduledocbox

171 Minimize

171.1 Definition

INT minimize ( )

Iconifies/minimizes the window.

171.2 Returns

INT : success

0 - If minimizing/iconification is not support or was refused by the window manager.
!0 - Success.

171.3 Example

```
import "mod_key"
import "mod_wm"

Process Main()
Begin

    Repeat
        if(key(_M))
                while(key(_M)) frame; end
                minimize();
        end
        frame;
    Until(key(_ESC)||exit_status)

End
```

Used in example: key(), **minimize**(), exit_status

Template:Moduledocbox

172 Mkdir

172.1 Definition

INT mkdir (<**STRING** directoryname>)

Creates a new directory in the current path of execution with a certain name.

172.2 Parameters

STRING directoryname - The name of the directory to be created.

172.3 Returns

INT : Success

0 (false) - Creating a new directory with the specified name failed.
!0 (true) - Creating a new directory with the specified name succeeded.
Template:Moduledocbox

173 Mode is ok

173.1 Definition

INT mode_is_ok (<**INT** width> , <**INT** height> , <**INT** depth>, <**INT** flags>)

Returns 0 if the requested mode is not supported under any bit depth,or returns the bits-per-pixel of the closest available mode with the given width, height and requested flags.

173.2 Parameters

INT width	- Width of the screen in pixels.
INT height	- Height of the screen in pixels.
INT depth	- Color depth of the screen. See color_depths.
INT flags	- Mode of rendering. See render flags.

173.3 Returns

INT : Whether the specified mode can be used.

0	- The specified mode cannot be used.
>0	- The bits-per-pixel of the closest available mode for the given width, height and flags.

173.4 Example

```
import "mod_video"
import "mod_key"
import "mod_wm"

Process Main()
Begin
    if (is_mode_ok(640,400,16,0))
     set_mode(640,400,16);
    else
     set_mode(640,480,16);
    end
    Repeat
        frame;
    Until(key(_ESC)||exit_status)
End
```

Used in example: is_mode_ok(),set_mode(), key(), exit_status

Template:Moduledocbox

174 Move draw

174.1 Definition

INT move_draw (<**INT** DrawID> , <**INT** x> , <**INT** y>)

Moves a certain drawing on the screen. Only drawings drawn with a certain z value can be moved like this, as other ones are drawn on a graphic and thus cannot be moved.

174.2 Parameters

INT DrawID	- DrawID of the drawing to be moved.
INT x	- The new x coordinate of the drawing.
INT y	- The new y coordinate of the drawing.

174.3 Returns

INT : true

Template:Funcbox

175 Move text

175.1 Definition

INT move_text (<**INT** TextID> , <**INT** x> , <**INT** y>)

Moves an existing text to another place on the screen.

175.2 Parameters

INT TextID	- Identifier of the text you want to move. This identifier should have been appointed to a text earlier on.
INT x	- The new horizontal coordinate (of the control point) of your text.
INT y	- The new vertical coordinate (of the control point) of your text.

175.3 Returns

<If someone knows, please edit!>

175.4 Notes

None.

175.5 Errors

<If someone knows, please edit!>

175.6 Example

```
Program test;
Global
    My_text;
Begin
    My_text=write(0,320/2,200/2,4,"Press space to move this.");
    Loop
        If (key(_space))
            Move_text(My_text,rand(0,319),rand(0,199));
        End
        Frame;
    End
End
```

Used in example: write(), key(), rand()

This will result in something like:
File:Move text.png

176 Move window

176.1 Definition

INT move_window (<**INT** x> , <**INT** y>)

Moves the Bennu window so that the top left of the window is on the specified (*x*,*y*).

Also called Set window pos.

176.2 Parameters

INT x - The new X coordinate of the top left of the window.
INT y - The new Y coordinate of the top left of the window.

176.3 Returns

INT : true

Template:Moduledocbox

177 NET About

Up to Network.DLL Functions

177.1 Definition

INT NET_About ( )

Tells the user about Network.DLL, using the console.

177.2 Returns

INT : The current Network.DLL version (integer).

177.3 Example

```
include "libnetwork.bh"

Process Main()
Begin

    NET_About();

End
```

Template:Netfuncbox

178 NET Accept

Up to Network.DLL Functions

178.1 Definition

INT NET_Accept (<**WORD** listen_connection>)

Waits for a new incoming connection on the specified listening connection. The connection identifier of the new incoming connection is returned. Waiting means that this function DOES NOT return until an incoming connection is made.

178.2 Parameters

WORD listen_connection - The connection identifier of the listening connection on which a new incoming connection is expected.

178.3 Returns

INT : Network.DLL Errorcode

NET_ERROR_INVALIDCONN	- The connection is invalid.
NET_ERROR_CONNINACTIVE	- The connection is inactive.
0 - n	- Connection identifier of the incoming connection on the specified listening connection. (Where n is the number of maximum allowed connections)

178.4 Example

As an example, here is the code for **NET_Accept**():

```
Function int NET_Accept(word listen_conn);
Begin

/* Checks */

        if(listen_conn>=NET.maxlistenports)
                if(NET.consolereports)
                        NET_ConsoleMessage(listen_conn,"Error: Connection " + listen_conn + " invalid");
                end
                return NET_ERROR_INVALIDCONN;
        end
        if(NET.Status[listen_conn]<=NET_STATUS_INACTIVE)
                if(NET.consolereports)
                        NET_ConsoleMessage(listen_conn,"Error: Connection " + listen_conn + " not active");
                end
                return NET_ERROR_CONNINACTIVE;
        end

/* Wait for new connection */

        While(NET.Incoming[listen_conn]!=NET_STATUS_ESTABLISHED)
                frame;
        End
        listen_conn = net_incoming_accept(listen_conn);

        return listen_conn;

End
```

Used in example: NET.Incoming, NET_Incoming_Accept()

Template:Netfuncbox

179 NET Connect

Up to Network.DLL Functions

179.1 Definition

INT NET_Connect (<**STRING** address> , <**WORD** port> , [<**BYTE** consolereports>])

Opens a connection to the specified address on the specified port. The opened connection can be disconnected with NET_Disconnect().

Also called NET_Open().

179.2 Parameters

STRING address	- The address to which will be connected.
WORD port	- The port on which will be connected.
[**BYTE** consolereports]	- true/false: specifies whether there will be console reports for this connection (like messages).

179.3 Returns

INT : Connection identifier.

NET_ERROR_INVALIDSOCKETSET	- The socketset is invalid.
NET_ERROR_SOCKETSETINACTIVE	- The socketset is inactive.
NET_ERROR_INVALIDPORTNUMBER	- The portnumber is invalid (<0 or >65535).
NET_ERROR_INVALIDTYPE	- The type specified is invalid.
NET_ERROR_TOOMANYCONNS	- There are too many connections
NET_ERROR_RESOLVINGHOST	- Could not resolve host.
NET_ERROR_CONNECTING	- Could not connect.
NET_ERROR_ADDINGSOCKET	- Could not add socket to socketset.
0 - n	- Connection identifier. (Where n is the number of maximum allowed connections)

179.4 Example

```
Program example;
    include "Network.fh";
Begin

    NET_Init(0,10,1);
    NET_Connect("www.google.com",80,true);

    Loop
        frame;
    End

End
```

Template:Netfuncbox

180 NET Disconnect

Up to Network.DLL Functions

180.1 Definition

INT NET_Disconnect (<**WORD** connection>)

Closes a the specified connection. This can be a listening connection and a normal connection.

Also called NET_Close().

180.2 Parameters

WORD connection - The connection identifier of the connection to be closed.

180.3 Returns

INT : Network.DLL Errorcode

NET_ERROR_INVALIDCONN	- The connection is invalid.
NET_ERROR_CONNINACTIVE	- The connection is inactive.
NET_ERROR_DELETINGSOCKET	- Could not delete the socket.
NET_ERROR_NONE	- No error.

180.4 Example

```
Program example;
    include "Network.fh";
Private
    int netid;
Begin

    NET_Init(0,10,1);
    netid = NET_Listen(4555,true);
    NET_Disconnect(netid);

    Loop
        frame;
    End

End
```

Template:Netfuncbox

181 NET DisconnectAll

Up to Network.DLL Functions

181.1 Definition

INT NET_DisconnectAll ( )

Closes all connections.

Also called NET_CloseAll().

181.2 Returns

INT : Network.DLL Errorcode

NET_ERROR_NONE - No error.

181.3 Example

```
Program example;
    include "Network.fh";
Begin

    NET_Init(0,10,1);
    NET_Connect("www.google.com",80,true);
    NET_DisconnectAll();

    Loop
        frame;
    End

End
```

Template:Netfuncbox

182 NET GetError

Up to Network.DLL Functions

182.1 Definition

STRING NET_GetError (<**INT** error_id>)

Returns a short description of the specified error.

182.2 Parameters

INT error_id - The errorcode.

182.3 Returns

STRING : The error.

Template:Netfuncbox

183 NET GetSeparator

183.1 Definition

STRING NET_GetSeparator (<**WORD** connection>)

Returns the current separation string for the specified connection.

NET_Separator(<**WORD** connection>) equals NET_GetSeparator(<**WORD** connection>).

183.2 Parameters

WORD connection - The connection identifier.

183.3 Returns

STRING : The separation string.

"" - There was an error or no separation string.
!"" - The separation string.
Template:Netfuncbox

184 NET GetSeparatorLength

Up to Network.DLL Functions

184.1 Definition

INT NET_GetSeparatorLength (<**WORD** connection>)

Returns the length of the current separation string for the specified connection.

184.2 Parameters

WORD connection - The connection identifier.

184.3 Returns

INT : The length of the separation string.

NET_ERROR_INVALIDCONN	- The connection is invalid.
>=0	- The length of the separation string.

Template:Netfuncbox

185 NET Hostname

Up to Network.DLL Functions

185.1 Definition

STRING NET_Hostname (<**WORD** connection>)

Gets the hostname of the other peer on a certain connection.

185.2 Parameters

WORD connection - The connection identifier of the connection of which the hostname of the other peer is wanted.

185.3 Returns

STRING : The hostname.

"" There was an error.
!"" The hostname (string).

185.4 Example

```
Program example;
    include "Network.fh";
Private
    int netid;
Begin

    NET_Init(0,10,1);
    netid = NET_Connect("www.google.com",80,true);
    say("Hostname: " + NET_Hostname(netid));

    Loop
        frame;
    End

End
```

Template:Netfuncbox

186 NET IPAddress

Up to Network.DLL Functions

186.1 Definition

STRING NET_IPAddress (<**WORD** connection>)

Gets the IP address of the other peer on a certain connection.

186.2 Parameters

WORD connection - The connection identifier of the connection of which the IP address of the other peer is wanted.

186.3 Returns

STRING : The IP address.

"" There was an error.
!"" The IP address (string).

186.4 Example

```
Program example;
    include "Network.fh";
Private
    int netid;
Begin

    NET_Init(0,10,1);
    netid = NET_Connect("www.google.com",80,true);
    say("IPAddress: " + NET_IPAddress(netid));

    Loop
        frame;
    End

End
```

Template:Netfuncbox

187 NET IPToInt

Up to Network.DLL Functions

187.1 Definition

STRING NET_IPToInt (<**STRING** IPAddress>)

Returns the int form of the IPAddress. If the string IPAddress is not of the form xxx.xxx.xxx.xxx, there is no guaranteed outcome.

187.2 Parameters

STRING IPAddress - The IPaddress.

187.3 Returns

INT : The int form of the IPaddress.

Template:Netfuncbox

188 NET Incoming Accept

Up to Network.DLL Functions

188.1 Definition

INT NET_Incoming_Accept (<**WORD** listen_connection>)

On the specified listening connection, accepts the last incoming connection. When the NET.Incoming[*listen_connection*] value becomes NET_STATUS_ESTABLISHED, this function can be used to accept the connection and retrieve the connectionID of the new incoming connection.

For a function that *waits* for a new connection, see NET_Accept().

188.2 Parameters

WORD listen_connection	- The connection identifier of the listening connection on which a new incoming connection is made.

188.3 Returns

INT : Network.DLL Errorcode

NET_ERROR_INVALIDCONN	- The connection is invalid.
NET_ERROR_CONNINACTIVE	- The connection is inactive.
NET_ERROR_TOOFEWCONNS	- There are no new incoming connections on the specified listening connection.
0 - n	- Connection identifier of the incoming connection on the specified listening connection. (Where n is the number of maximum allowed connections)

188.4 Example

As an example, here is the code for NET_Accept():

```
Function int NET_Accept(word listen_conn);
Begin

/* Checks */

        if(listen_conn>=NET.maxlistenports)
                if(NET.consolereports)
                        NET_ConsoleMessage(listen_conn,"Error: Connection " + listen_conn + " invalid");
                end
                return NET_ERROR_INVALIDCONN;
        end
        if(NET.Status[listen_conn]<=NET_STATUS_INACTIVE)
                if(NET.consolereports)
                        NET_ConsoleMessage(listen_conn,"Error: Connection " + listen_conn + " not active");
                end
                return NET_ERROR_CONNINACTIVE;
        end

/* Wait for new connection */

        While(NET.Incoming[listen_conn]!=NET_STATUS_ESTABLISHED)
                frame;
        End
        listen_conn = net_incoming_accept(listen_conn);

        return listen_conn;

End
```

Used in example: NET.Incoming, **NET_Incoming_Accept**()

Template:Netfuncbox

189 NET Init

Up to Network.DLL Functions

189.1 Definition

INT NET_Init (<**INT** timeout> , <**WORD** maxconnections> , <**WORD** maxlistenports>)

Initializes Network.DLL.

189.2 Parameters

INT timeout	- Milliseconds to wait for messages each frame. (0 is the best thing, really)
WORD maxconnections	- Maximum number of connections.
WORD maxlistenports	- Maximum number of ports on which can be listened for connections.

189.3 Returns

INT : Network.DLL Errorcode

NET_ERROR_ALREADYINIT	- Network.DLL already initialized.
NET_ERROR_INITIALIZATION	- Error during initialization.
NET_ERROR_TOOFEWCONNS	- Too small number of maximum connections
NET_ERROR_TOOMANYLISTENERS	- Too many listenports specified.
>=0	- No error (ProcessID of the NET process)

189.4 Notes

Also consider (initializing) the following Global variables (before calling NET_Init()):

- word NET.MaxConnections
- word NET.MaxListenPorts
- word NET.ActiveConnections
- word NET.ActiveListenPorts
- byte NET.ConsoleReports

189.5 Example

```
include "libnetwork.bh"

Process Main()
Begin

    NET_Init(0,10,1);
    NET_Quit();

End
```

Used in example: **NET_Init**(), NET_Quit()

Template:Netfuncbox

190 NET IntToIP

Up to Network.DLL Functions

190.1 Definition

STRING NET_IntToIP (<**INT** IP>)

Returns the normal xxx.xxx.xxx.xxx form of the IP.

190.2 Parameters

INT IP - The IP as an integer.

190.3 Returns

STRING : The xxx.xxx.xxx.xxx form of the IP.

"" - There was an error.
!"" - The xxx.xxx.xxx.xxx form of the IP.
Template:Netfuncbox

191 NET IntVersion

Up to Network.DLL Functions

191.1 Definition

INT NET_IntVersion ( )

Returns the current Network.DLL version.

191.2 Returns

INT : The current Network.DLL version (int).

191.3 Example

```
Program example;
    include "Network.fh";
Private
    int netver;
Begin

    netver = NET_IntVersion();

    Loop
        frame;
    End

End
```

Template:Netfuncbox

192 NET Listen

Up to Network.DLL Functions

192.1 Definition

INT NET_Listen (<**WORD** port> , [<**BYTE** consolereports>])

Opens a listenconnection on the specified port. The connections accepted will inherit certain variables from the listenport they were connected on. These variables are consolereports and separator. You only need to call this function once in order to accept multiple clients. The listening connection can be closed with NET_Disconnect().

192.2 Parameters

WORD port	- The port on which to listen.
[**BYTE** consolereports]	- true/false: specifies whether there will be console reports for this connection (like messages).

192.3 Returns

INT : Connection identifier.

NET_ERROR_INVALIDSOCKETSET	- The socketset is invalid.
NET_ERROR_SOCKETSETINACTIVE	- The socketset is inactive.
NET_ERROR_INVALIDPORTNUMBER	- The portnumber is invalid (<0 or >65535).
NET_ERROR_INVALIDTYPE	- The type specified is invalid.
NET_ERROR_TOOMANYCONNS	- There are too many connections
NET_ERROR_RESOLVINGHOST	- Could not resolve host.
NET_ERROR_LISTENINGONCONN	- Could not listen on connection.
NET_ERROR_ADDINGSOCKET	- Could not add socket to socketset.
0 - n	- Connection identifier. (Where n is the number of maximum allowed connections)

192.4 Example

```
Program example;
    include "Network.fh";
Begin

    NET_Init(0,10,1);
    NET_Listen(4555,true);

    Loop
        frame;
    End

End
```

Template:Netfuncbox

193 NET Port

Up to Network.DLL Functions

193.1 Definition

INT NET_Port (<**WORD** connection>)

Gets the port in use on a certain connection.

Both ListenConnections and normal connections can be specified.

193.2 Parameters

WORD connection	- The connection identifier of the connection of which the port in use is wanted.

193.3 Returns

INT : The port in use on the specified connection.

NET_ERROR_INVALIDCONN	- The connection is invalid.
0 - 65536	- The port in use on the specified connection.

193.4 Example

```
Program example;
    include "Network.fh";
Private
    int netid;
Begin

    NET_Init(0,10,1);
    netid = NET_Connect("www.google.com",80,true);
    say("Port: " + NET_Port(netid));

    Loop
        frame;
    End

End
```

Template:Netfuncbox

194 NET Quit

Up to Network.DLL Functions

194.1 Definition

INT NET_Quit ( )

Deinitializes Network.DLL.

194.2 Returns

INT : Network.DLL Errorcode

NET_ERROR_NONE - No error.

194.3 Example

```
include "libnetwork.bh"

Process Main()
Begin

    NET_Init(0,10,1);
    NET_Quit();

End
```

Used in example: NET_Init(), **NET_Quit**()

Template:Netfuncbox

195 NET Recv

Up to Network.DLL Functions

195.1 Definition

INT NET_Recv (<**WORD** connection> , [<**BYTE** includeseperator>])

Gets the waiting message on a certain connection.

The function will keep returning messages until it returns "", which indicates no more new messages.

195.2 Parameters

WORD connection - The connection identifier.
[**BYTE** includeseparator] - When true, the separator will be added to the message at the end. When false, it won't. Default is true.
Also called NET_GetMessage().

195.3 Returns

INT : Network.DLL Errorcode

"" - There was an error or no message.
!"" - The message (string). It appears this can also be a byte, word or integer. Just make sure the other peer sent it like it was received.

195.4 Notes

It is best to put this function in a frameless loop, like so:

```
msg = NET_Recv(conn_c);
Repeat
    say("Incoming(" + conn_c + "): " + msg);
    msg = NET_Recv(conn_c);
Until(len(msg)<=0)
```

This way it will still return "" if "" was indeed sent. If you don't need this or you have some other reason, you can change the loop to suit your needs, like:

```
while(len(msg = NET_Recv(conn_c))>0)
    say("Incoming(" + conn_c + "): " + msg);
End
```

195.5 Example

```
Program example;
    include "Network.fh";
Private
    int netid;
    string message;
Begin

    NET_Init(0,10,1);
    netid = NET_Connect("www.google.com",80,true);

    Loop
        If(NET.Incoming[netid])
            While( (message = NET_Recv(netid)) !="" )
                say("Incoming(" + netid + "): " + message);
            End
        End
        frame;
    End

End
```

Template:Netfuncbox

196 NET RecvFile

Up to Network.DLL Functions

196.1 Definition

INT NET_RecvFile (<**WORD** connection> , <**STRING** filename>)

Receives a file on the specified connection. The contents of the received file will be written to the file with specified filename.

196.2 Parameters

WORD connection	- The connection identifier.
STRING filename	- The filename of the file to write to, including a possible path.

196.3 Returns

INT : The size of the successfully received file. Without the separatorlength.

NET_ERROR_INVALIDCONN	- The connection is invalid.
NET_ERROR_CONNINACTIVE	- The connection is inactive.
NET_ERROR_OPENINGFILE	- Could not open file.
NET_ERROR_NONEWMESSAGE	- The incomingbuffer is empty.
NET_ERROR_UNFINISHEDMESSAGE	- The data in the incomingbuffer is incomplete.
NET_ERROR_SIZEMISMATCH	- Sizes do not match: tried to receive a non-existing file.
>=0	- The size of the successfully received file. Without the separatorlength.

196.4 Notes

This function is to receive files sent with NET_SendFile().

Template:Netfuncbox

197 NET RecvGraph

197.1 Definition

INT NET_RecvGraph (<**WORD** connection>)

Receives a graphic on the specified connection. With the data received on the connection, a new graphic will be created in the system file and its graphID returned.

197.2 Parameters

WORD connection - The connection identifier.

197.3 Returns

INT : The size of the successfully sent graph. With the separatorlength, if the separator was added.

NET_ERROR_INVALIDCONN	- The connection is invalid.
NET_ERROR_CONNINACTIVE	- The connection is inactive.
NET_ERROR_CREATINGGRAPH	- Could not create graphic.
NET_ERROR_NONEWMESSAGE	- The incomingbuffer is empty.
NET_ERROR_UNFINISHEDMESSAGE	- The message in the incomingbuffer is incomplete.
NET_ERROR_SIZEMISMATCH	- Sizes do not match: tried to receive a non-existing graph.
>=0	- The graphID of the graphic created using the received data.

197.4 Notes

This function is to receive graphics sent with NET_SendGraph().

Template:Netfuncbox

198 NET RecvVar

Up to Network.DLL Functions

198.1 Definition

INT NET_RecvVar (<**WORD** connection> , <**BYTE POINTER** data> , <**INT** length> , [<**BYTE** includeseparator>])

The receiving equivalent of NET_SendVar(): stores the data at the specified pointer, unlike NET_GetMessage(), which returns the message as a string, not allowing some variables.

198.2 Parameters

WORD connection	- The connection identifier.
BYTE pointer message	- The pointer the data/variable will be written to.
INT length	- The length/size of the data/variable.
[**BYTE** includeseparator]	- true/false: whether the connection's separator will be added to the received data at the end. Default is false.

198.3 Returns

INT : The size of the successfully received message. Without the separatorlength.

NET_ERROR_INVALIDCONN	- The connection is invalid.
NET_ERROR_CONNINACTIVE	- The connection is inactive.
NET_ERROR_NONEWMESSAGE	- The incomingbuffer is empty.
NET_ERROR_UNFINISHEDMESSAGE	- The data in the incomingbuffer is incomplete.
NET_ERROR_SIZEMISMATCH	- Sizes do not match (*length* and the length of the received data).
>=0	- The size of the successfully received data. Without the separatorlength.

198.4 Notes

This function is mainly to receive data sent with NET_SendVar().

Template:Netfuncbox

199 NET Resolve

Up to Network.DLL Functions

199.1 Definition

STRING NET_Resolve (<**STRING** IPAddress>)

Returns the hostname of the IPAddress specified. A hostname can also be specified.

199.2 Parameters

STRING IPAddress - The IPAddress (like "127.0.0.1") or hostname.

199.3 Returns

STRING : The hostname.

"" - There was an error. !"" - The hostname.
Template:Netfuncbox

200 NET Send

Up to Network.DLL Functions

200.1 Definition

INT NET_Send (<**WORD** connection> , <**STRING** message> , [<**BYTE** includeseparator>])

Sends a certain message to the other peer on a certain connection.

It appears the message can also be a byte, word or integer. Just make sure the other peer receives it like it was sent. For those variables NET_SendVar() is recommended though.

Also called NET_Message().

200.2 Parameters

WORD connection	- The connection identifier of the connection on which the other peer is to receive the message.
STRING message	- The message.
[**BYTE** includeseparator]	- true/false: whether the connection's separator will be added to the message at the end. Default is true.

200.3 Returns

INT : The size of the successfully sent message. With the separatorlength, if the separator was added.

NET_ERROR_INVALIDCONN	- The connection is invalid.
NET_ERROR_CONNINACTIVE	- The connection is inactive.
NET_ERROR_MESSAGETOOLONG	- The message is too long.
NET_ERROR_MESSAGETOOSHORT	- The message is too short.
NET_ERROR_SENDING	- Could not send.
>=0	- The size of the successfully sent message. With the separatorlength, if the separator was added.

200.4 Example

```
Program example;
    include "Network.fh";
Private
    int netid;
Begin

    NET_Init(0,10,1);
    netid = NET_Connect("www.google.com",80,true);
    NET_Send(netid,"HELLO!"+chr(13)+chr(10)); // This is not a valid HTTP protocol message

    Loop
        frame;
    End

End
```

Template:Netfuncbox

201 NET SendFile

Up to Network.DLL Functions

201.1 Definition

INT NET_SendFile (<**WORD** connection> , <**STRING** filename>)

Sends a file.

201.2 Parameters

WORD connection	- The connection identifier.
STRING filename	- The filename of the file to be sent, including a possible path.

201.3 Returns

INT : The size of the successfully sent file. With the separatorlength, if the separator was added.

NET_ERROR_INVALIDCONN	- The connection is invalid.
NET_ERROR_CONNINACTIVE	- The connection is inactive.
NET_ERROR_NONEXISTINGFILE	- The file does not exist.
NET_ERROR_OPENINGFILE	- Could not open file.
NET_ERROR_SENDING	- Could not send.
>=0	- The size of the successfully sent file. With the separatorlength, if the separator was added.

201.4 Notes

Make sure the file is received properly. NET_RecvFile() is helpful for this.

Template:Netfuncbox

202 NET SendGraph

Up to Network.DLL Functions

202.1 Definition

INT NET_SendGraph (<**WORD** connection> , <**INT** fileID> , <**INT** graphID>)

Sends a graphic on a connection.

202.2 Parameters

WORD connection	- The connection identifier.
INT fileID	- The FileID of the file the to be sent graphic is located in.
INT graphID	- The GraphID of the graphic to be sent.

202.3 Returns

INT : The size of the successfully sent graph. With the separatorlength, if the separator was added.

NET_ERROR_INVALIDCONN	- The connection is invalid.
NET_ERROR_CONNINACTIVE	- The connection is inactive.
NET_ERROR_NONEXISTINGGRAPH	- The graphic does not exist.
NET_ERROR_SENDING	- Could not send.
>=0	- The size of the successfully sent graph. With the separatorlength, if the separator was added.

202.4 Notes

Make sure the graphic is received properly. NET_RecvGraph() is helpful for this.

Template:Netfuncbox

203 NET SendRN

Up to Network.DLL Functions

203.1 Definition

INT NET_SendRN (<**WORD** connection> , <**STRING** message>)

Same as NET_Send() except there is added \r\n at the end of the message. The connection's separator is not added to the end.

Also called NET_MessageRN().

203.2 Parameters

WORD connection	- The connection identifier of the connection on which on which the other peer is to receive the message.
STRING message	- The message.

203.3 Returns

INT : The size of the successfully sent message. With the added \r\n.

NET_ERROR_INVALIDCONN	- The connection is invalid.
NET_ERROR_CONNINACTIVE	- The connection is inactive.
NET_ERROR_MESSAGETOOLONG	- The message is too long.
NET_ERROR_MESSAGETOOSHORT	- The message is too short.
NET_ERROR_SENDING	- Could not send.
>=0	- The size of the successfully sent message. With the added \r\n.

203.4 Example

```
Program example;
    include "Network.fh";
Private
    int netid;
Begin

    NET_Init(0,10,1);
    netid = NET_Connect("www.google.com",80,true);
    NET_SendRN(netid,"HELLO!"); // This is not a valid HTTP protocol message

    Loop
        frame;
    End

End
```

Template:Netfuncbox

204 NET SendVar

Up to Network.DLL Functions

204.1 Definition

INT NET_SendVar (<**WORD** connection> , <**BYTE POINTER** data> , <**INT** length> , [<**BYTE** includeseparator>])

Sends data in array form or a byte, word or int or whatever other variable.

A null character on the end is not needed:

```
NET_SendVar( netid , &data , sizeof(data_element)*elements );
```

Or send an integer:

```
NET_SendVar( netid , &my_int , sizeof(int) );
```

Or send a struct:

```
NET_SendVar( netid , mystruct , sizeof(mystruct) );
```

204.2 Parameters

WORD connection	- The connection identifier.
BYTE POINTER data	- The pointer to the data/variable.
INT length	- The length/size of the data/variable.
[**BYTE** includeseparator]	- true/false: whether the connection's separator will be added to the data at the end. Default is true.

204.3 Returns

INT : The size of the succesfully sent data. With the separatorlength, if the separator was added.

NET_ERROR_INVALIDCONN	- The connection is invalid.
NET_ERROR_CONNINACTIVE	- The connection is inactive.
NET_ERROR_MESSAGETOOLONG	- The data is too big.
NET_ERROR_SENDING	- Could not send.
>=0	- The size of the succesfully sent data. With the separatorlength, if the separator was added.

204.4 Notes

Make sure the data is received properly. If you send a struct, be sure the receiver knows he should be receiving a struct and then receive the struct like it should. NET_RecvVar() is helpful for this.

Template:Netfuncbox

205 NET Separator

Up to Network.DLL Functions

205.1 Definition

INT NET_Separator (<**WORD** connection> , <**STRING** separator> , <**BYTE** length>)

Specifies a separation string. Messages will be separated where this string is found. This is useful for various reasons. Like making an IRC client, you can do:

```
NET_Separator( netid , chr(13)+chr(10) , 2 );
```

IRC uses the ending string \r\n (carriagereturn and newline), so now the incoming messages will be automatically separated.

Maximum length is 255 characters.

NET_Separator(<**WORD** connection>) equals NET_GetSeparator(<**WORD** connection>).

205.2 Parameters

WORD connection	- The connection identifier.
STRING separator	- The separation string. Specify 0 to disable message separation.
BYTE length	- The length of the separation string. Specify 0 to disable message separation.

205.3 Returns

INT : Network.DLL Errorcode

NET_ERROR_INVALIDCONN	- The connection is invalid.
NET_ERROR_CONNINACTIVE	- The connection is inactive.
NET_ERROR_NONE	- No error.

Template:Netfuncbox

206 NET Stat Buffer

Up to Network.DLL Functions

206.1 Definition

INT NET_Stat_Buffer (<**WORD** connection>)

Gets the incoming messages buffer. This does not affect the buffer. This function is only useful in some debug cases.

206.2 Parameters

WORD connection - The connection identifier.

206.3 Returns

STRING : The incoming messages buffer.

"" - There was an error or the buffer is empty.
!"" - The incoming messages buffer.
Template:Netfuncbox

207 NET Version

Up to Network.DLL Functions

207.1 Definition

STRING NET_Version ( )

Returns the current Network.DLL version.

207.2 Returns

STRING : The current Network.DLL version (string).

207.3 Example

```
Program example;
    include "Network.fh";
Private
    string netver;
Begin

    netver = NET_Version();

    Loop
        frame;
    End

End
```

Template:Netfuncbox

208 Pal load

208.1 Syntax

INT pal_load (<**STRING** filename>)

208.2 Description

Loads a color palette from a file.

The current palette is switched to the loaded one. Note that one can load a palette from an 8bit FPG or MAP file (the remaining graphic data will not be loaded) or a PAL file.

Also called load_pal().

208.3 Parameters

STRING filename - The filename of the file that you wish to load the palette from (including extension and possible path).

208.4 Returns

INT : Error.

-1 - Error: could not open file; corrupt or truncated file; file doesn't contain palette information.
0 - Error: could not obtain filename; some FPL error.
1 - No error: palette was loaded with success.

208.5 Example

```
Program example;
Begin

    load_pal("example.pal");

    Loop
        frame;
    End

End
```

Template:Moduledocbox

209 Pango render

This function is part of the Pango library.

209.1 Syntax

INT pango_render (<**STRING** markup>)

209.2 Description

Returns a graphic with the specified markup rendered on it. The markup is expected to be a UTF-8 encoded string; markup accepted by the Pango BennuGD binding is the same as upstream Pango's and can be consulted here.

In case of failure, a 0x0 sized graphic will be returned.

After usage, unload the graphic with map_unload().

209.3 Parameters

STRING markup - The markup to render to a graphic.

209.4 Returns

INT : GraphID

0 - There was some error. Pango tends to drop error on stdout, so check your terminal for them.
>0 - The graphID of a new graphic with the specified markup rendered on it.

209.5 Example

```
import "mod_say";
import "mod_key";
import "mod_video";
import "mod_map";
import "mod_mouse";
import "mod_text";

import "pango";

Process main()
Private
    int retval=0;
    string markup = '<span foreground="blue" background="black">e=mc<sup>2</sup></span>';
Begin

    set_mode(640, 480, 32, MODE_WINDOW);

    // If given and argument, render that instead of the default text
    if(argc == 2)
        markup = '<span font_family="Serif" size="xx-large" foreground="blue" background="black"><u>'+argv[1]+'</u></span>';
    end;

    say("Going to render:");
    say(markup);

    // Render the markup and move it with the mouse
    graph = pango_render(markup);
    write_var(0, 0, 0, 0, x);
    write_var(0, 0, 10, 0, y);

    // Wait for escape key
    while(! key(_ESC))
        x = mouse.x;
        y = mouse.y;
        FRAME;
    End;

OnExit

    unload_map(0, graph);

End
```

Used in example: import(), set_mode(), say(), **pango_render**(), write_var(), key(), unload_map(), mouse

210 Pause song

210.1 Definition

INT pause_song ( )

Pauses the currently playing song.

210.2 Notes

The song pauses immediately, but can be resumed later by calling resume_song(). For a nicer effect, you may want to fade the music out before pausing. See fade_music_off().

210.3 Returns

INT : Error.

-1 - Error: sound inactive.
0 - No error.

210.4 Example

```
program music_example;
global
    my_song;
    playing;
    paused;
    faded_in;
    v;
begin
    set_mode(640,480,16);

    my_song=load_song("beat.ogg");


    write(0,320,30,4,"Use the keyboard to control the music playback.");
    write(0,320,50,4,"Key [ENTER] starts / stops the song.");
    write(0,320,60,4,"Key [SPACE] pauses / resumes the song.");
    write(0,320,70,4,"Key [0] through key [9] changes the song volume.");
    write(0,320,80,4,"Key [F] fades the song in or out.");

    write(0,320,120,5,"Playing: ");
    write_int(0,320,120,3,&playing);

    write(0,320,140,5,"Paused: ");
    write_int(0,320,140,3,&paused);

    write(0,320,160,5,"Faded in: ");
    write_int(0,320,160,3,&faded_in);

    write(0,320,180,5,"Volume: ");
    write_int(0,320,180,3,&v);

    v=128;
    faded_in=true;


    repeat
        if(key(_enter))
            if(is_playing_song())
                stop_song();
                playing=false;
            else
                play_song(my_song,1);
                playing=true;
            end
            while(key(_enter))frame;end
        end

        if(key(_space))
            if(paused)
                paused=false;
                resume_song();
            else
                paused=true;
```

```
                pause_song();
            end
            while(key(_space))frame;end
        end

        if(key(_f))
            if(faded_in)
                faded_in=false;
                fade_music_off(100);
            else
                faded_in=true;
                fade_music_in(my_song,1,100);
            end
            while(key(_f))frame;end
        end

        if(key(_0))v=0;end
        if(key(_1))v=14;end
        if(key(_2))v=28;end
        if(key(_3))v=43;end
        if(key(_4))v=57;end
        if(key(_5))v=71;end
        if(key(_6))v=85;end
        if(key(_7))v=100;end
        if(key(_8))v=114;end
        if(key(_9))v=128;end

        set_song_volume(v);

    frame;
    until(key(_esc))

    exit();
end
```

Used in example: key(), set_mode(), load_song(), write(), write_int(), **pause_song**(), play_song(), stop_song(), resume_song(), fade_music_in(), fade_music_off(), set_song_volume().

Template:Funcbox

211 Play song

211.1 Definition

INT play_song (<**INT** songID> , <**INT** repeats>)

Plays a song.

211.2 Parameters

INT songID	- SongID of the song loaded previously with load_song().
INT repeats	- Number of times to repeat the song. Use -1 for an infinite loop.

211.3 Returns

INT : Error.

-1	- Error: sound inactive; mixer error; invalid songID.
0	- No error.

211.4 Errors

Sound inactive	- The sound is inactive.
Invalid songID	- The songID was invalid.
Other	- Some Mixer error.

211.5 Example

```
Program example;
Private
    int song;
Begin
    song = load_song("my_song.ogg");
    play_song(song,0);
    Loop
        frame;
    End
End
```

Used in example: load_song()

Template:Funcbox

212 Play wav

212.1 Definition

INT play_wav (<**INT** waveID> , <**INT** repeats> , [<**INT** channel>])

Plays a sound effect previously loaded with load_wav().

212.2 Parameters

INT waveID	- The WaveID of the sound effect to be played.
INT repeats	- Number of times to *repeat* the sound effect. Use -1 for an infinite loop.
[**INT** channel]	- The sound channel the sound effect is to be played on (-1 for any, default).

212.3 Returns

INT : The sound channel the sound effect is now playing on.

-1	- Error: sound inactive; invalid waveID
>=0	- The sound channel the sound effect is now playing on.

212.4 Example

```
Program
Private
    int wave;
Begin
    wave = load_wav("my_wav.wav");
    play_wav(wave,0);
    Loop
        frame;
    End
End
```

Used in example: load_wav()

Template:Funcbox

213 Png load

213.1 Definition

INT png_load (<**STRING** filename>)

Creates a new graphic, using the specified PNG file as contents and puts it in the system file. Returns the graphID of the created graphic. The color depth of the created graphic will be the same as the loaded PNG file.

Also called load_png().

213.2 Parameters

STRING filename - The name of the PNG file to be loaded, including a possible path.

213.3 Returns

INT : graphID

0 - There was an error loading the file.
>0 - The graphID of the newly created graphic.

213.4 Example

Checkout the PNG_LoadDirectory tutorial.

Template:Moduledocbox

214 Png save

214.1 Definition

INT png_save (<**INT** fileID> , <**INT** graphID> , <**STRING** filename>)

Saves the specified graphic as *filename* with the format PNG.

Also called save_png().

214.2 Parameters

INT fileID	- The fileID of the file that holds the graphic.
INT graphID	- The graphID of the graphic to save.
STRING filename	- The name of the PNG file to be saved, including a possible path.

214.3 Returns

INT : Successrate

false	- Error.
true	- Success.

214.4 Example

```
//here's a cool thing to save a screenshot
import "mod_map"
import "mod_screen"
import "mod_key"

Global
    int takingscreenshot;
End

Process Main()
Begin

    Loop

        If (key(_F12))
            If (takingscreenshot==0)
                takingscreenshot=1;
                graph=screen_get(); // grabs the screen and sets it as the program graphic
                png_save(0,graph,"shot"+rand(0,9999)+".png"); // saves the graphic as a png with a
                                                              // random number in the filename to
                                                              // prevent overwriting
                map_unload(0,graph);  //frees the graphic
            Else
                takingscreenshot=0;
            End
            While(key(_F12)) Frame; End
       End

        frame;
    End
End
```

Used in example: key(), screen_get(), **png_save**(), map_unload()

Template:Moduledocbox

215 Point get

215.1 Definition

INT get_point (<**INT** fileID> , <**INT** graphID> , <**INT** controlpointID> , <**INT POINTER** x>, <**INT POINTER** y>)

Allows you to obtain a control point of a particular graph.

Any graph can contain up to 1000 control points (from 0 to 999). Control point 0 is the center of the graphic. This function allows you to know the location of any control point belonging to any graph.

To set a control point, use set_point() or, for only the center of a graph, set_center().

215.2 Parameters

INT fileID	- Number of the FPG library.
INT graphID	- Number of the graph inside the library which you want to use.
INT controlpointID	- Number of the control point.
INT POINTER x	- Pointer to where the X-coordinate of the control point will be written.
INT POINTER y	- Pointer to where the Y-coordinate of the control point will be written.

215.3 Returns

INT : Successrate

false	- One of the following: specified graph is invalid, specified control point is invalid, specified control point is undefined.
true	- The control point was defined or the center was used.

215.4 Example

```
Program cpoint;
Private
    int map;
    int cx,cy;
Begin

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

    // Set the center to a random point
    set_center(0,map,rand(-10,110),rand(-10,110));

    // Get the center
    get_point(0,map,0,&cx,&cy);

    // Show the center
    say("Center-X: " + cx);
    say("Center-Y: " + cy);

    // Assign the map to the graph variable
    graph = map;

    // Set the location of this process to the center of the screen
    x = 160;
    y = 100;

    Loop
        frame;
    End

End
```

Used in example: new_map(), map_clear(), set_center(), say(), pointer, graph

Notice that setting the center influences the position of the graph: Template:Image

Template:Moduledocbox

216 Point set

216.1 Definition

INT point_set (<**INT** fileID> , <**INT** graphID> , <**INT** controlpointID> , <**INT** x>, <**INT** y>)

Allows you to set a control point of a particular graphic.

Any graph can contain up to 1000 control points (from 0 to 999). Control point 0 is the center of the graphic. This function allows you to set the location of any control point belonging to any graph. The coordinates are relative to the upper left corner of the graphic.

To obtain the coordinates of a control point, use point_get().

Also called set_point().

216.2 Parameters

INT fileID	- FileID of the file containing the graphic.
INT graphID	- GraphID of the graphic of which to set a control point.
INT controlpointID	- Number of the control point.
INT x	- The new X-coordinate of the control point.
INT y	- The new Y-coordinate of the control point.

216.3 Returns

INT : Successrate

-1 - One of the following: specified graph is invalid, specified control point is invalid.
1 - The control point was set successfully.

216.4 Example

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

Process Main()
Private
    int map;
    int cx,cy;
Begin

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

    // Set the center to a random point
    point_set(0,map,0,rand(-10,110),rand(-10,110));

    // Get the center
    point_get(0,map,0,&cx,&cy);

    // Show the center
    say("Center-X: " + cx);
    say("Center-Y: " + cy);

    // Assign the map to the graph variable
    graph = map;

    // Set the location of this process to the center of the screen
    x = 160;
    y = 100;

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

End
```

Used in example: new_map(), map_clear(), **point_set**(), point_get(), say(), pointer, graph

Notice that setting the center influences the position of the graph: Template:Image

Template:Moduledocbox

217 Pow

217.1 Definition

FLOAT pow (<**FLOAT** base> , <**FLOAT** power>)

Returns *base* to the power of *power* (*base*^*power*).

217.2 Parameters

FLOAT base	- The base.
FLOAT power	- The power.

217.3 Returns

FLOAT : *base* to the power of *power* (*base*^*power*).

217.4 Example

```
Program powerful;
Global
    float value1;
    int   value2;
Begin

    write_float(0,0, 0,0,&value1);
    write_int  (0,0,10,0,&value2);

    value1 = pow(2.3,4.6);
    value2 = pow(2  ,3  );

    Repeat
        frame;
    Until(key(_ESC))

End
```

Used in example: write_float(), write_int(), **pow**(), key()

Template:Funcbox

218 Put

218.1 Definition

INT put (<**INT** fileID> , <**INT** graphID> , <**INT** x> , <**INT** y>)

Draws (blits) a graph onto the background.

For more advanced blitting, see:

- xput()
- map_put()
- map_xput()
- map_xputnp()

218.2 Parameters

INT fileID	- The file that holds the graph.
INT graphID	- The graph to draw with.
INT x	- Where on the background's x-axis to put the graph.
INT y	- Where on the background's y-axis to put the graph.

218.3 Returns

INT : true

218.4 Notes

The x and y parameters denote where to draw the graph, that is, where the center of the to be drawn graph will be.

218.5 Errors

Unsupported color depth	- The origin graph's color depth is greater than the destination graph's.

218.6 Example

```
import "mod_map"
import "mod_screen"
import "mod_key"

Process Main()
Private
    int map;
Begin

    // Create a new graph of size 100x100 and color depth of 8bit
    map = map_new(100,100,8);

    // Clear the map red
    map_clear(0,map,rgb(255,0,0));

    // Put it in the center of the screen
    put(0,map,160,100);

    Repeat
        frame;
    Until(key(_ESC))

End
```

Used in example: map_new(), map_clear(), **put**(), key()

This will result in something like:
Template:Image

Template:Moduledocbox

219 Quicksort

219.1 Definition

INT quicksort (<**VOID POINTER** array> , <**INT** elementsize> , <**INT** elements> , <**INT** dataoffset> , <**BYTE** datasize> , <**BYTE** datatype>)

Sorts an array by the Quicksort ordering algorithm.

This function is very handy for user defined types for elements in which a sort-variable is present. For simple arrays or arrays in which the first variable is the sort-variable, sort() can be used. For arrays in which the sort-variable is a String, ksort() can be used.

219.2 Parameters

VOID POINTER array	- Pointer to the first element of the array to be sorted.
INT elementsize	- The size of an element in the array in bytes.
INT elements	- The number of elements in the array.
INT dataoffset	- The number of bytes the sort-variable in each element is relative to the start of that element.
BYTE datasize	- The size of the sort-variable in bytes.
BYTE datatype	- The datatype of the sort-variable. (0:integer, 1:float)

219.3 Returns

INT: true

219.4 Example

```
Program sorting;

Type _player
    String name;
    int score;
End

Const
    maxplayers = 5;
Global
    _player player[maxplayers-1];
Begin

    // Insert some values
    player[0].name = "That one bad looking dude";
    player[1].name = "Ah pretty lame guy";
    player[2].name = "Some cool dude";
    player[3].name = "OMG ZOMG guy";
    player[4].name = "This person is ok";

    player[0].score = 70;
    player[1].score = 30;
    player[2].score = 80;
    player[3].score = 90;
    player[4].score = 50;


    // Show array
    say("-------------------- unsorted");
    for(x=0; x<maxplayers; x++)
        say(player[x].name + " - " + player[x].score);
    end

/* Sort by name ( quicksort() can't be used to sort Strings,
   as a String in Fenix is a pointer to the actual String,
   so it would sort the pointer addresses */

    // sort()
    sort(player); // sorts by name because name is the first variable in each element

    // Show array
    say("-------------------- name - sort()");
    for(x=0; x<maxplayers; x++)
        say(player[x].name + " - " + player[x].score);
    end

    // ksort()
```

```
    ksort(player,player[0].name,maxplayers);

    // Show array
    say("-------------------- name - ksort()");
    for(x=0; x<maxplayers; x++)
        say(player[x].name + " - " + player[x].score);
    end

/* Sort by score (sort() cannot be used here, because score is not the first variable) */

    // ksort()
    ksort(player,player[0].score,maxplayers);

    // Show array
    say("-------------------- score - ksort()");
    for(x=0; x<maxplayers; x++)
        say(player[x].name + " - " + player[x].score);
    end

    // quicksort()
    quicksort(&player[0],sizeof(_player),maxplayers,sizeof(String),sizeof(int),0);

    // Show array
    say("-------------------- score - quicksort()");
    for(x=0; x<maxplayers; x++)
        say(player[x].name + " - " + player[x].score);
    end

    // Wait until ESC is pressed
    Repeat
         frame;
    Until(key(_esc))

End
```

Used in example: say(), sort(), ksort(), **quicksort**(), type, array, pointer

220 Rand

220.1 Definition

INT rand (<**INT** lowerlimit> , <**INT** upperlimit>)

Returns a random number, ranging from a certain lower limit to a certain upper limit. The limits are within the range.

Make sure the difference between *lowerlimit* and *upperlimit* does not exceed 32767 (2^15-1). If that is needed, the function *rand2()* below can be used.

220.2 Parameters

INT lowerlimit - The lower limit for the random value.
INT upperlimit - The upper limit for the random value.

220.3 Returns

INT : A random value: lowerlimit <= result <= upperlimit

220.4 Notes

To synchronize rand() on different computers, the function rand_seed() can be used.

Rand() is not a very good function on itself. To counter this, the following *rand2()* can be used:

```
#define RAND_MAX 32767
#define DRAND_RANGE (1.0/((RAND_MAX + 1)*(RAND_MAX + 1)))
#define irand(x) ((unsigned int) ((x) * drand ()))
Function float drand ()
Private
    float f;
Begin
    Repeat
       f = (rand (0,RAND_MAX) * (RAND_MAX + 1.0)
          + rand (0,RAND_MAX)) * DRAND_RANGE;
    Until (f < 1); /* Round off */
    return f;
End
Function int rand2(int lowerlimit, int upperlimit)
Begin
    return (lowerlimit+irand(upperlimit-lowerlimit+1));
End
```

To understand this code, one can read its source.

Template:Moduledocbox

221 Rand seed

221.1 Definition

INT rand_seed (<**INT** seed>)

Seeds the random generator, used in rand().

This is useful for synchronizing the random generator on multiple machines, as when the same *seed* is used, calls to rand() with the same limits will return values in the same order on all the machines.

To reset the seeding to its original state, meaning the state before any call to rand() or **rand_seed**(), set *seed* to `1`.

221.2 Parameters

INT seed - The seed for the random generator used in rand(); 1 to reset.

221.3 Returns

INT : true

221.4 Example

```
import "mod_rand"
import "mod_say"
import "mod_time"

Process Main()
Begin
        say("First number: " + (rand(0,1000)%100));
        rand_seed(time());
        say("Random number: " + (rand(0,1000)%100));
        rand_seed(1);
        say("Again the first number: " + (rand(0,1000)%100));
End
```

Used in example: say(), rand(), **rand_seed**()

Template:Moduledocbox

222 Realloc

222.1 Syntax

VOID POINTER realloc (<**VOID POINTER** data> , <**INT** size>)

222.2 Description

Resizes the given block of memory.

It allocates a new block of memory, copying the old data. If the new size is smaller than the old size, the last part of the data is lost. If the new size of the block of memory requires movement of the block, the old memory block is freed.

222.3 Parameters

VOID POINTER data	- Pointer to the block of memory to be resized.
INT size	- The new size of the block of memory in bytes.

222.4 Returns

VOID POINTER : Pointer to (the first element of) the newly allocated memory block.

NULL	- There was are an error allocating the memory, like insufficient memory available.
!NULL	- Pointer to (the first element of) the newly allocated memory block.

222.5 Example

```
import "mod_mem"
import "mod_say"

Process Main()
Private
    byte pointer pbyte;
    byte pointer pbyte2;
    int elements = 10;
    int newelements = 15;
    int i;
Begin

    // Allocate memory
    pbyte = alloc(elements);

    // Set them to 13
    memset(pbyte,13,elements);

    // Relocate it to a larger, newly made memory block
    pbyte2 = realloc(pbyte,newelements);

    // Set the added part's elements to 16 (newelements > elements)
    memset(pbyte+elements,16,newelements-elements);

    // Show numbers
    for(i=0; i<newelements; i++)
        say("byte2["+i+"] = " + pbyte2[i]);
    end

OnExit

    // Free the used memory
    free(pbyte2);

End
```

Used in example: alloc(), memset(), **realloc**(), say(), free(), pointer

Template:Moduledocbox

223 Region define

223.1 Definition

INT region_define (<**INT** regionID> , <**INT** x> , <**INT** y> , <**INT** width> , <**INT** height>)

Defines the boundaries of a region.

There are 32 regions, range `0..31`. Region *0* is always the whole screen and cannot be changed. Defining regions can be useful for the function out_region(), the local variable region and using them with scrolls.

Also called define_region().

223.2 Parameters

INT regionID	- The regionID of the region to define.
INT x	- The x coordinate of the top left corner of the region.
INT y	- The y coordinate of the top left corner of the region.
INT width	- The width of the region.
INT height	- The height of the region.

223.3 Returns

INT : The regionID specified.

Template:Moduledocbox

224 Region out

224.1 Definition

INT out_region (<**INT** processID> , <**INT** regionID>)

Checks if the specified process is completely outside of the specified region.

The check is not pixel perfect, but uses the bounding box of the process.

Also called out_region().

224.2 Parameters

INT processID	- The processID of the process to check.
INT regionID	- The regionID of the region to check with.

224.3 Returns

INT : true/false: whether the process is completely outside the region.

true	- The process is completely outside the region.
false	- The process is (partly) inside the region or invalid region or process specified.

Template:Moduledocbox

225 Resume song

225.1 Definition

INT resume_song ( )

Resumes a song, after it has been paused with the function pause_song().

225.2 Returns

INT : Error.

-1 - Error: sound inactive.
0 - No error.

225.3 Notes

The song will instantly start playing again with this function. For a nicer effect, you may want to fade the music in as you resume it. See fade_music_in().

225.4 Example

```
program music_example;
global
    my_song;
    playing;
    paused;
    faded_in;
    v;
begin
    set_mode(640,480,16);

    my_song=load_song("beat.ogg");


    write(0,320,30,4,"Use the keyboard to control the music playback.");
    write(0,320,50,4,"Key [ENTER] starts / stops the song.");
    write(0,320,60,4,"Key [SPACE] pauses / resumes the song.");
    write(0,320,70,4,"Key [0] through key [9] changes the song volume.");
    write(0,320,80,4,"Key [F] fades the song in or out.");

    write(0,320,120,5,"Playing: ");
    write_int(0,320,120,3,&playing);

    write(0,320,140,5,"Paused: ");
    write_int(0,320,140,3,&paused);

    write(0,320,160,5,"Faded in: ");
    write_int(0,320,160,3,&faded_in);

    write(0,320,180,5,"Volume: ");
    write_int(0,320,180,3,&v);

    v=128;
    faded_in=true;


    repeat
        if(key(_enter))
            if(is_playing_song())
                stop_song();
                playing=false;
            else
                play_song(my_song,1);
                playing=true;
            end
            while(key(_enter))frame;end
        end

        if(key(_space))
            if(paused)
                paused=false;
                resume_song();
            else
                paused=true;
                pause_song();
            end
```

```
            while(key(_space))frame;end
        end

        if(key(_f))
            if(faded_in)
                faded_in=false;
                fade_music_off(100);
            else
                faded_in=true;
                fade_music_in(my_song,1,100);
            end
            while(key(_f))frame;end
        end

        if(key(_0))v=0;end
        if(key(_1))v=14;end
        if(key(_2))v=28;end
        if(key(_3))v=43;end
        if(key(_4))v=57;end
        if(key(_5))v=71;end
        if(key(_6))v=85;end
        if(key(_7))v=100;end
        if(key(_8))v=114;end
        if(key(_9))v=128;end

        set_song_volume(v);

    frame;
    until(key(_esc))

    exit();
end
```

Used in example: key(), set_mode(), load_song(), write(), write_int(), pause_song(), play_song(), stop_song(), **resume_song**(), fade_music_in(), fade_music_off(), set_song_volume().

Template:Funcbox

226 Rgb

226.1 Syntax

DWORD rgb (<**BYTE** red> , <**BYTE** green> , <**BYTE** blue>)

226.2 Description

Finds the single color in the current color mode closest to the combined red, green, and blue values specified. In 32bit mode, the alpha is set to 255.

Equal to `rgba(red,green,blue,255)`

226.3 Parameters

BYTE red	- Level of red in the desired color from 0 to 255.
BYTE green	- Level of green in the desired color from 0 to 255.
BYTE blue	- Level of blue in the desired color from 0 to 255.

226.4 Returns

DWORD : Returns the best matched color code.

226.5 Notes

Different color depths have different color codes, this is why rgb() is useful: it returns the appropriate code, based on the current color depth. When in 8bit mode, this code is 0..255 and when in 16bit mode 0..65535. In 32bit mode the code is 0xRRGGBBAA, two letters meaning one byte.

Using this function in different color depths can be tricky. This is because rgb() will return a different color code under different color depths. For example, when at 8bit, we do:

```
my_color = rgb(100,100,100);
```

my_color will most likely have the value *6* at this time. Suppose we change the color depth to 16bit now, using set_mode(). Now, my_color holds a totally different color than before, as *6* is nowhere near rgb(100,100,100) in 16bit mode. To counter this effect, my_color needs to be reinitialized:

```
my_color = rgb(100,100,100);
```

The same code, but rgb() now returns the proper code, as it always returns the code belonging to the current color depth. my_color will now be about *25388*.

226.6 Example

```
import "mod_map"
import "mod_text"
import "mod_key"
import "mod_wm"

Process Main()
Private
    byte red=0;
    byte green=255;
    byte blue=0;
Begin

    set_text_color(rgb(red,green,blue)); // rgb finds the color closest to pure green and
                                         // passes it to set_text_color

    write(0,1,1,0,"Green text for everybody!"); //this text will be green

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

End
```

Used in example: set_text_color(), **rgb**(), write(), key(), exit_status

Template:Moduledocbox

227 Rgba

227.1 Syntax

DWORD rgba (<**BYTE** red> , <**BYTE** green> , <**BYTE** blue>, <**BYTE** alpha>)

227.2 Description

Finds the single color in the current color mode closest to the combined red, green, blue and alpha values specified. This function is useful in 32 bpp modes.

227.3 Parameters

BYTE red	- Level of red in the desired color from 0 to 255.
BYTE green	- Level of green in the desired color from 0 to 255.
BYTE blue	- Level of blue in the desired color from 0 to 255.
BYTE alpha	- Level of alpha in the desired color from 0 to 255, 0 being completely transparent and 255 completely opaque.

227.4 Returns

DWORD : Returns the best matched color code.

227.5 Notes

Different color depths have different color codes, this is why rgba() is useful: it returns the appropriate code, based on the current color depth. When in 8bit mode, this code is 0..255 and when in 16bit mode 0..65535. In 32bit mode the code is 0xRRGGBBAA, two letters meaning one byte.

Using this function in different color depths can be tricky. This is because rgba() will return a different color code under different color depths. For example, when at 8bit, we do:

```
my_color = rgba(100,100,100,255);
```

my_color will most likely have the value *6* at this time. Suppose we change the color depth to 16bit now, using set_mode(). Now, my_color holds a totally different color value than before, as *6* is nowhere near rgba(100,100,100,255) in 16bit mode. To counter this effect, my_color needs to be reinitialized:

```
my_color = rgba(100,100,100,255);
```

The same code, but rgba() now returns the proper code, as it always returns the code belonging to the current color depth. my_color will now be about *25388*.

227.6 Example

```
import "mod_text"
import "mod_map"
import "mod_key"
import "mod_video"
import "mod_wm"

Const
    SCREEN_WIDTH  = 320;
    SCREEN_HEIGHT = 200;
    SCREEN_DEPTH  = 16;
End

Process Main()
Private
    int r=0;
    int g=255;
    int b=0;
    int a=0;
    int da=10;
Begin

    set_mode(SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_DEPTH);

    graph = map_new(100,100,SCREEN_DEPTH);
```

```
    map_clear(0,graph,rgba(r,g,b,a));

    x = y = 100;

    Repeat
        a += da;
        if(a<0)
            da = -da;
            a = 0;
        elseif(a>255)
            da = -da;
            a = 255;
        end
        map_clear(0,graph,rgba(r,g,b,a));
        frame;
    Until(key(_ESC)||exit_status)

End
```

Used in example: set_mode(), map_new(), map_clear(), key(), **rgba**(), graph, exit_status

Template:Moduledocbox

228 Rgbscale

228.1 Definition

INT rgbscale (<**INT** fileID> , <**INT** graphID> , <**FLOAT** r> , <**FLOAT** g> , <**FLOAT** b>)

This will convert the specified graphic by using the specified color as a reference. The converted graphic will have only the specified color and lighter/darker colors; see notes for the details.

228.2 Parameters

INT fileID	- The fileID of the file that holds the graphics.
INT graphID	- The graphID of the graphic to convert.
FLOAT r	- The red component of the color to be used for reference.
FLOAT g	- The green component of the color to be used for reference.
FLOAT b	- The blue component of the color to be used for reference.

228.3 Returns

INT

-1 - Invalid graphic.

1 - Success.

228.4 Notes

The exact formula is:

```
for every pixel:
    c = 0.3 * oldpixel_r + 0.59 * oldpixel_g + 0.11 * oldpixel_b
    newpixel_r = r * c;
    newpixel_g = g * c;
    newpixel_b = b * c;
```

where r,g,b are the specified r,g,b.

Note that **`rgbscale`**`(0,map,1,1,1) = grayscale(0,map,0)`, for a valid graphic (0,map).

Template:Moduledocbox

229 Rm

229.1 Definition

INT rm (<**STRING** filename>)

Removes (deletes) the file specified with *filename*.

229.2 Parameters

STRING filename - The name of the file to be removed (deleted).

229.3 Returns

INT : Success

0 (false) - Removing the file with the specified name failed.
!0 (true) - Removing the file with the specified name succeeded.
Template:Moduledocbox

230 Rmdir

230.1 Definition

INT rmdir (<**STRING** directoryname>)

Removes (deletes) the directory in the current path of execution with a certain name.

230.2 Parameters

STRING directoryname - The name of the directory to be removed (deleted).

230.3 Returns

INT : Success

0 (false) - Removing the directory with the specified name failed.
!0 (true) - Removing the directory with the specified name succeeded.
Template:Moduledocbox

231 Rpad

231.1 Definition

STRING rpad(<**STRING** str> , <**INT** length>)

Returns the string *str*, padding (adding spaces to) the back of the string if needed to make *str* of length *length*. The original string will remain unchanged.

If *length* is smaller or equal to the length of *str*, the returned string is *str*.

231.2 Parameters

STRING str	- The string to pad (add spaces to).
INT length	- The minimal length of the returned string.

231.3 Returns

STRING: padded string

231.4 Example

```
import "mod_string"
import "mod_say"

Process Main()
Private
    string ABC = "ABC";
    string _ABC;
    string ABC__;
Begin

    ABC = lpad(ABC,2);
    _ABC = lpad(ABC,4);
    ABC__ = rpad(ABC,5);

    say('ABC = "' + ABC + '"');
    say('_ABC = "' + _ABC + '"');
    say('ABC__ = "' + ABC__ + '"');

End
```

Used in example: say(), lpad(), **rpad**()

Result:

```
ABC = "ABC"
_ABC = " ABC"
ABC__ = "ABC  "
```

Template:Moduledocbox

232 Save

232.1 Definition

INT save (<**STRING** filename> , <**VARSPACE** data>)

Saves the data from the specified variable to the specified file.

232.2 Parameters

STRING filename	- The name of the file that will be saved.
VARSPACE data	- The variable (of any datatype) that will be saved in a file.

232.3 Returns

INT : The number of bytes written to the file.

232.4 Notes

Attempting to use "?","*","<",">" or "|" in a filename will result in no file at all on Windows, while using ":" in a filename results in everything from the ":" and beyond being cut off from the file name and the resulting file will be of size 0.

Using the characters "/" or "\" in the filename (without directories that can be accessed that way) results in everything from this character and before being cut off from the filename. The file will be saved successfully nonetheless.

232.5 Example

```
Program test;
Global
    struct My_struct
        Level_number="99";
        string Map_name="Amazing map";
    End
Begin
        Save("myfile.sav",My_struct); // The content of My_struct is saved in myfile.sav
        Write(0,10,10,0,"Data saved!");

    While (!key(_ESC))
        Frame;
    End
End
```

Used in example: **save**(), write(), key()

Template:Funcbox

233 Say

233.1 Definition

INT say (<**STRING** message>)

Prints *message* to stdout (console).

- Similar to `System.out.println(message)` in Java.
- Similar to `printf("%s\n",message)` in C

233.2 Parameters

STRING message - The message to print to stdout

233.3 Returns

INT - true

233.4 Example

```
import "mod_say"

Process Main()
Begin
    Say("Hello World!");
End
```

This will result in the output on console:

```
Hello World!
```

Template:Moduledocbox

234 Screen clear

234.1 Definition

INT screen_clear ( )

Clears the background of the screen, making it completely black.

Also called clear_screen().

234.2 Returns

INT : true

234.3 Notes

This is the same as **map_clear**(0,BACKGROUND). When the background is cleared in either way, Bennu knows the background is empty and will take advantage of this knowledge.

234.4 Errors

Unsupported color depth	- The specified graph has a not supported color depth.

Template:Moduledocbox

235 Screen get

235.1 Definition

INT screen_get ( )

Creates a new graphic containing a copy of the lastly rendered frame.

The map will contain everything, including background, processes, drawings and text. Just like map_new(), the newly created graphic will be located in the System file (fileID of *0*); the graphID will be returned. After the use of this graphic, it should be freed using map_unload().

Also called get_screen().

235.2 Returns

INT : GraphID

0	- Some error.
>0	- The GraphID of the graphic created.

235.3 Example

```
import "mod_key"
import "mod_screen"
import "mod_map"

Global
    int my_map;
End

Process Main()
Begin
    Repeat
        if(key(_F5))
            my_map = screen_get();
            png_save(0,my_map,"snapshot.PNG");
            map_unload(0,my_map);
            while(key(_F5)) frame; end
        end
        frame;
    Until(key(_ESC))
End
```

Used in example: key(), **screen_get**(), png_save(), map_unload()

Template:Moduledocbox

236 Screen put

236.1 Definition

INT screen_put (<**INT** fileID> , <**INT** graphID>)

Clears and draws (blits) a graph onto the background in the center.

For more advanced blitting, see:

- put()
- xput()

Also called put_screen().

236.2 Parameters

INT fileID	- The file that holds the graph.
INT graphID	- The graph to draw with.

236.3 Returns

INT

0	- Invalid map.
1	- Success.

236.4 Notes

The center of the specified graph influences its placement.

The following codes are equivalent:

```
screen_put(f,g);
```

```
screen_clear();
put(f,g,graphic_info(0,BACKGROUND,G_WIDTH)/2,graphic_info(0,BACKGROUND,G_HEIGHT)/2);
```

See screen_clear(), put() and map_info().

236.5 Errors

Invalid map	- The specified map is invalid.
Unsupported color depth	- The origin graph's color depth is greater than the destination graph's.

Template:Moduledocbox

237 Set Wav Volume

237.1 Syntax

INT set_wav_volume (<**INT** waveID> , <**INT** volume>)

237.2 Description

Change the reproduction volume of the wav track.

With this function, it is possible to set the volume of the sound effects, etc.

237.3 Parameters

INT waveID - waveID as returned by load_wav().
INT volume - New volume. (0..128)

237.4 Returns

(assumed) **INT** : Error.

-1 - Error: sound inactive.
0 - No error.

237.5 Notes

This function changes the reproduction volume of the wav track. The volume level can be set between 0 (silence) and 128 (original 100% volume of the track. The default volume is 128.

237.6 Example

```
global
    int my_wav;
    int v;
end

process main()
begin

    set_mode(640,480,16);

    my_wav = load_wav("beat.wav");

    write(0,320,30,4,"Use the keyboard to control the music playback.");
    write(0,320,50,4,"Key [ENTER] starts the wav.");
    write(0,320,60,4,"Key [0] through key [9] changes the song volume.");

    write(0,320,180,5,"Volume: ");
    write_int(0,320,180,3,&v);

    v = 128;

    repeat
        if(key(_ENTER))
            play_wav(my_wav,50);
            while(key(_ENTER))
                frame;
            end
        end


        if(key(_0)) v =   0; end
        if(key(_1)) v =  14; end
        if(key(_2)) v =  28; end
        if(key(_3)) v =  43; end
        if(key(_4)) v =  57; end
        if(key(_5)) v =  71; end
        if(key(_6)) v =  85; end
        if(key(_7)) v = 100; end
        if(key(_8)) v = 114; end
        if(key(_9)) v = 128; end
```

```
        set_wav_volume(v);

        frame;
    until(key(_ESC))

end
```

Used in example: key(), set_mode(), load_wav(), write(), write_int(), play_wav().

This example uses media: beat.wav

Template:Moduledocbox

238 Set fps

238.1 Definition

INT set_fps (<**INT** fps> , <**INT** skip>)

Sets the frames per second (framerate) your program aims to display. The more frames per second, the faster your program runs. Some computers might not be able to display the amount of frames you specified, and will show a lower fps. Therefore, it is important you choose a fps that is reasonable and can also be displayed by the somewhat slower computers. If you don't use this function then the default fps will be used (25 fps).

238.2 Parameters

INT fps	- Frames per second to use. The default is 25.
INT skip	- Frames the program is allowed to skip to keep up with the specified framerate if it's running low on processor time. The default is 0.

238.3 Returns

INT : The FPS entered.

238.4 Notes

If you use Set_fps(0,0), then your program will run at the maximum speed your computer can possibly handle.

The current FPS can be read from the global variable fps.

238.5 Errors

None.

238.6 Example

```
Program test;
Begin
    Set_fps(60,0);
    Loop
        Frame;
    End
End
```

Template:Funcbox

239 Set icon

239.1 Definition

INT set_icon (<**INT** fileID> , <**INT** graphID>)

Set the window icon to a certain graph.

The icon will only be updated after set_mode() is called, so call set_icon() before set_mode(). The map used for the icon must be 32x32 large, but it can have different depths. After set_icon() and set_mode() have been called, the map used for the icon can be freed from memory using unload_map().

239.2 Parameters

INT fileID	- The fileID of the file containing the graph.
INT graphID	- The graphID of the graph to be used as an icon.

239.3 Returns

INT : true

239.4 Example

```
import "mod_key"
import "mod_map"
import "mod_wm"
import "mod_video"

Const
    screen_width  = 320;
    screen_height = 200;
    screen_depth  =  16;
End

Process Main()
Private
    int map;
    int iconsize      =  32;
Begin

    set_mode(screen_width,screen_height,screen_depth);

    map = new_map(iconsize,iconsize,screen_depth);
    map_clear(0,map,rgb(0,255,255));

    set_title("<-- Look at the cyan block!");
    set_icon(0,map);

    unload_map(0,map);

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

End
```

Used in example: set_mode(), new_map(), map_clear(), rgb(), set_title(), **set_icon**(), unload_map(), key(), exit_status

Template:Moduledocbox

240 Set mode

240.1 Syntax

INT set_mode (<**INT** width> , <**INT** height> , [<**INT** depth>] , [<**INT** flags>])

240.2 Description

Sets the screen resolution of your program, and optionally the colordepth of the screen and any render flags for extra options. If this command is not used, the default settings will take effect (320x240 at 32 bpp).

Some much used resolutions are: 320x240, 640x480, 800x600, 1024x768 and 1280x1024.

240.3 Parameters

INT width	- Width of the screen in pixels.
INT height	- Height of the screen in pixels.
INT [depth]	- Color depth of the screen. See color_depths.
INT [flags]	- Mode of rendering. See render flags.

240.4 Returns

INT : true

240.5 Notes

Any fpg files you load must have the same or a lower colordepth as you set for the screen.

Uncommon resolutions can also be used, for example 399x10, which will be the actual size of the window if you run in windowed mode. At full screen black edges might appear.

There is another method of calling set_mode(): with one parameter. In this parameter you must use the 4 right digits for the height and the rest fot the width. For example to set a 320x200 mode you can use set_mode(3200200). The maximum height that allow this method is 9999 pixels. This method is **deprecated** and its use is disadvised.

240.6 Example

```
import "mod_video"
import "mod_key"
import "mod_wm"

Process Main()
Begin
    set_mode(640,480,16);
    Repeat
        frame;
    Until(key(_ESC)||exit_status)

End
```

Used in example: **set_mode**(), key(), exit_status

Template:Moduledocbox

241 Set song volume

241.1 Definition

INT set_song_volume (<**INT** volume>)

Change the reproduction volume of the music track.

With this function, it is possible to set the background music to a different volume than the sound effects, etc.

241.2 Parameters

INT volume - New volume. (0..128).

241.3 Returns

INT : Error.

-1 - Error: sound inactive.
0 - No error.

241.4 Notes

This function changes the reproduction volume of the music track. The volume level can be set between 0 (silence) and 128 (original 100% volume of the track. The default volume is 128.

241.5 Example

```
program music_example;
global
    my_song;
    playing;
    paused;
    faded_in;
    v;
begin
    set_mode(640,480,16);

    my_song=load_song("beat.ogg");


    write(0,320,30,4,"Use the keyboard to control the music playback.");
    write(0,320,50,4,"Key [ENTER] starts / stops the song.");
    write(0,320,60,4,"Key [SPACE] pauses / resumes the song.");
    write(0,320,70,4,"Key [0] through key [9] changes the song volume.");
    write(0,320,80,4,"Key [F] fades the song in or out.");

    write(0,320,120,5,"Playing: ");
    write_int(0,320,120,3,&playing);

    write(0,320,140,5,"Paused: ");
    write_int(0,320,140,3,&paused);

    write(0,320,160,5,"Faded in: ");
    write_int(0,320,160,3,&faded_in);

    write(0,320,180,5,"Volume: ");
    write_int(0,320,180,3,&v);

    v=128;
    faded_in=true;


    repeat
        if(key(_enter))
            if(is_playing_song())
                stop_song();
                playing=false;
            else
                play_song(my_song,1);
                playing=true;
            end
            while(key(_enter))frame;end
```

```
        end

        if(key(_space))
            if(paused)
                paused=false;
                resume_song();
            else
                paused=true;
                pause_song();
            end
            while(key(_space))frame;end
        end

        if(key(_f))
            if(faded_in)
                faded_in=false;
                fade_music_off(100);
            else
                faded_in=true;
                fade_music_in(my_song,1,100);
            end
            while(key(_f))frame;end
        end

        if(key(_0))v=0;end
        if(key(_1))v=14;end
        if(key(_2))v=28;end
        if(key(_3))v=43;end
        if(key(_4))v=57;end
        if(key(_5))v=71;end
        if(key(_6))v=85;end
        if(key(_7))v=100;end
        if(key(_8))v=114;end
        if(key(_9))v=128;end

        set_song_volume(v);

    frame;
    until(key(_esc))

    exit();
end
```

Used in example: key(), set_mode(), load_song(), write(), write_int(), pause_song(), play_song(), stop_song(), resume_song(), fade_music_in(), fade_music_off(), **set_song_volume**().

Template:Funcbox

242 Set text color

242.1 Definition

INT set_text_color (<**WORD** color>)

Sets the current text color (the color where texts will be written in). This only affects 1 bit (2 color) fonts, which can be loaded with load_font() or load_bdf(). 8 bit and 16 bit fonts already contain color information themselves and thus aren't affected.

242.2 Parameters

WORD color - The color to use for text.

242.3 Returns

INT : true if successful and false if failed. (*Needs confirmation.*)

242.4 Notes

Be warned that values returned by the Rgb() function differ with the video card. So, directly filling out color numbers as color parameter in 16 bit color mode without using Rgb() is a bad idea, as RGB returns the correct color code for every video card

242.5 Errors

<If someone knows, please edit!>

242.6 Example

```
Program awesome;
Global
    byte red=0;
    byte green=255;
    byte blue=0;

Begin

    set_text_color(rgb(red,green,blue));
    write(0,1,1,0,"Mijn potlood is bruin"); //this text will be green as an Irishman's ejecta

    set_text_color(rgb(255,0,0));
    write(0,1,11,0,"Je moeder"); //this text will be red

    Loop

        frame;
    End
End
```

Used in example: write(), rgb()

This results in something like this:
File:Set text color.png

243 Set title

243.1 Definition

INT set_title (<**STRING** title>)

Sets the title of the program's window.

The title will only be updated after set_mode() is called, so call set_title() before set_mode().

243.2 Parameters

STRING title - The new title for the program's window.

243.3 Returns

INT : true

243.4 Example

```
Program icon;
Private
    int map;
    int screen_width  = 320;
    int screen_height = 200;
    int screen_depth  =   8;
    int iconsize      =  32;
Begin

    map = new_map(iconsize,iconsize,screen_depth);
    map_clear(0,map,rgb(0,255,255));

    set_icon(0,map);
    set_title("<-- Look at the cyan block!");
    set_mode(screen_width,screen_height,screen_depth);

    unload_map(0,map);

    Repeat
        frame;
    Until(key(_esc))

End
```

Used in example: new_map(), map_clear(), set_icon(), set_mode, unload_map(), key()

Template:Funcbox

244 Signal

244.1 Definition

INT signal (<**INT** processID|processTypeID> , <**INT** signal>)

Allows a process/function or a range of processes/functions of certain processType to be controlled in a limited number of ways, by sending signals.

244.2 Parameters

INT processID|processTypeID	- The ProcessID of the process or the ProcessTypeID of the type of processes to send the signal to.
INT signal	- The signal that is to be sent to the target process(es).

244.3 Returns

INT :

false	- The specified processID|processTypeID was `ALL_PROCESS` or a processTypeID.
true	- The specified processID|processTypeID was a processID.

244.4 Errors

Invalid signal	- The specified signal is invalid.

244.5 Notes

To obtain the processType of a function, the operator Type can be used. For a process to send a signal to itself, the local variable id can be used. The parameter signal is passed by way of a set of signals which denote the signal to be sent. The parameter signal is one of the listed signals. To kill all processes at once, except the calling one, use let_me_alone().

To send a signal to all processes at once, except the calling one, use `signal(ALL_PROCESS,signal).`

To specify how a process reacts on an incoming signal, use signal_action().

A signal is carried out immediately, but this does not mean execution of processes is switched to the process, if you just woke it up. This can be achieved by using `frame(0);` after the signal. A frame will cause Bennu to continue execution with the process having the lowest frame-percentage done *and* it will only continue execution with the current process after all other processes have had their turn. Moreover, using a frame-percentage of *0* will make sure there is nothing added to the done frame-percentage of the process, meaning it doesn't affect the framerate of the process.

244.6 Example

```
signal( get_id(type enemy) , s_kill ); // Kills a process of type enemy.
                                       // Becareful! If there is no enemy process alive,
                                       // get_id() will return 0, which means signal()
                                       // will signal all processes.

signal( id , s_kill_tree );            // Kills the process that calls it, and all of its descendants

signal( Type player , s_freeze );      // Freezes all processes of type player so that they are
                                       //     still displayed, but do not execute any code.

signal(ALL_PROCESS,s_freeze);          // Freezes all processes except the one that called it.
                                       // Can be used to pause a game.

signal( someID , S_WAKEUP ); frame(0); // Wake up the process someID and let it execute once
                                       // (if not done this frame) before continuing.
```

Template:Funcbox

245 Signal action

245.1 Definition

INT signal_action ([<**INT** processID|processTypeID> ,] **INT** signal , **INT** action)

Sets the reaction of one or more processes when they receive a certain nonforceful-signal. Only *existing* processes are affected, processes created afterwards are not.

245.2 Parameters

INT processID|processTypeID	- A ProcessID,ProcessTypeID or `ALL_PROCESS`.
INT signal	- The code of a nonforceful-signal for which a reaction is to be specified.
INT action	- The reaction of the process when it receives a signal. (S_DFL/S_IGN)

245.3 Returns

INT : true

245.4 Notes

The reaction to an incoming forced signal (S_KILL_FORCE, S_SLEEP_FORCE, etc) cannot be changed and is **S_DFL** by default.

245.5 Example

```
// The current process ignores the kill signal from now on
signal_action(S_KILL,S_IGN);

// All currently existing processes ignore the kill signal from now on
signal_action(ALL_PROCESS,S_KILL,S_IGN);

// All currently existing processes of type 'Player' ignore the freeze signal from now on
signal_action(type Player,S_FREEZE,S_IGN);
```

Template:Moduledocbox

246 Sin

246.1 Definition

FLOAT sin (<**FLOAT** angle>)

Returns the sine of the specified angle.

This function performs a sine calculation on a certain angle and returns a value between -1 and 1.

246.2 Parameters

FLOAT angle - Angle, in thousandths of degrees. i.e. 75000 = 75º

246.3 Returns

FLOAT : The sine result of the specified angle.

246.4 Notes

The angle value used in this function should be in thousandths of degrees, as most angles within Bennu are.

To read about all aspects of trigonometry, you can click on Wikipedia's Trigonometric function page.

246.5 Example

```
Const
    screen_width  = 320;
    screen_height = 200;
    screen_border = 15;
End

Global
    float value;
End

Process Main()
Begin

    // Modes
    set_title("Sine Graph");
    set_mode(screen_width,screen_height);

    // X axis
    for(x=1;x<=8;x++)
        write( 0,
               screen_border+x*(screen_width-screen_border)/8+3,
               screen_height-1,
               8,
               itoa(x*360/8 )+"^" );
    end
    draw_line(1,screen_height-screen_border,screen_width,screen_height-screen_border);

    // Y axis
    write(0,screen_border-1,20,5,"1");
    write(0,screen_border-1,screen_height/2,5,"0");
    write(0,screen_border-1,screen_height-20,5,"-1");
    draw_line(screen_border,1,screen_border,screen_height-1);

    // Draw tangent
    for(angle=0;angle<360;angle++)
        value=sin(angle*1000)*(screen_height/2-20);
        put_pixel( screen_border+angle*(screen_width-screen_border)/360,
                   screen_height/2-value,
                   rgb(255,255,255) );
        // screen_height/2-value because the screen's origin (0,0) is topleft instead of downleft.
    end

    Repeat
        frame;
    Until(key(_ESC))

End
```

Used in example: set_title(), set_mode(), write(), draw_line(), **sin**(), put_pixel(), key()

This will result in something like:
Template:Image

Template:Funcbox

247 Sort

247.1 Syntax

INT sort (<**VARSPACE** array> , [<**INT** datacount>])

247.2 Description

Sorts an array by sorting a certain number of elements, by using the first variable in each element. By default the whole array is sorted.

If the elements contain multiple variables, ksort() can be used to select the variable to be used for sorting. For more advanced sorting, look at quicksort().

247.3 Parameters

VARSPACE array	- The array to be sorted.
[**INT** datacount]	- Number of elements to sort.

247.4 Returns

INT: Successrate

true	- Sorting succeeded.
false	- Sorting failed. Look in the output for the error.

247.5 Example

```
import "mod_sort";
import "mod_key";
import "mod_say";

Type _player
    String name;
    int score;
End

Const
    maxplayers = 5;
End;


Process main()
Private
    _player player[maxplayers-1];
    int i=0;
Begin
    // Insert some values
    player[0].name = "That one bad looking dude";
    player[1].name = "Ah pretty lame guy";
    player[2].name = "Some cool dude";
    player[3].name = "OMG ZOMG guy";
    player[4].name = "This person is ok";

    player[0].score = 70;
    player[1].score = 30;
    player[2].score = 80;
    player[3].score = 90;
    player[4].score = 50;


    // Show array
    say("-------------------- unsorted");
    for(i=0; i<maxplayers; i++)
        say(player[i].name + " - " + player[i].score);
    end

/* Sort by name ( quicksort() can't be used to sort Strings,
   as a String in Bennu is a pointer to the actual String,
   so it would sort the pointer addresses */

    // sort()
    sort(player); // sorts by name because name is the first variable in each element
```

```
    // Show array
    say("-------------------- name - sort()");
    for(i=0; i<maxplayers; i++)
        say(player[i].name + " - " + player[i].score);
    end

    // ksort()
    ksort(player,player[0].name,maxplayers);

    // Show array
    say("-------------------- name - ksort()");
    for(i=0; i<maxplayers; i++)
        say(player[i].name + " - " + player[i].score);
    end

/* Sort by score (sort() cannot be used here, because score is not the first variable) */

    // ksort()
    ksort(player,player[0].score,maxplayers);

    // Show array
    say("-------------------- score - ksort()");
    for(i=0; i<maxplayers; i++)
        say(player[i].name + " - " + player[i].score);
    end

    // quicksort()
    quicksort(&player[0],sizeof(_player),maxplayers,sizeof(String),sizeof(int),0);

    // Show array
    say("-------------------- score - quicksort()");
    for(i=0; i<maxplayers; i++)
        say(player[i].name + " - " + player[i].score);
    end
End
```

Used in example: say(), **sort**(), ksort(), quicksort(), type, array, pointer

248 Split

248.1 Syntax

INT split (<**STRING** delimiter> , <**STRING** str> , <**STRING POINTER** array> , <**INT** max_number>)

248.2 Description

Splits a string in several strings using a regular expression as delimiter.

The first piece will go to *array*[0], the second to *array*[1], and so forth, until either there are no more pieces left or *max_number* pieces are returned into the array. The number of pieces returned this way is returned by the function.

248.3 Parameters

STRING delimiter	- The regular expression used as delimiter to split.
STRING str	- The input string to split into multiple string.
STRING POINTER array	- Pointer to the string array where the pieces will be returned to.
INT max_number	- The maximum number of strings to return.

248.4 Returns

INT : The number of pieces returned into the array.

248.5 Example

```
import "mod_say"
import "mod_regex"

Process Main()
Private
        string str = "A,B,C,D,E";
        string a[9];
        int n;
        int i;
Begin

        // Split
        n = split(",",str,&a,10);

        // Display result
        say("Number of pieces: " + n);
        for(i=0; i<n; i++)
                say("[" + i + "] = " + a[i]);
        end

End
```

Used in example: **split**(), say()

Template:Moduledocbox

249 Sqrt

249.1 Definition

FLOAT sqrt (<**FLOAT** value>)

Returns the square root of a certain value.

249.2 Parameters

FLOAT value - The value of which the square root will be returned.

249.3 Returns

FLOAT : The square root of *value*.

Template:Funcbox

250 Start scroll

250.1 Definition

INT Start_scroll (<**INT** scrollnumber> , <**INT** fileID> , <**INT** graphID> , <**INT** backgroundgraphID> , <**INT** regionnumber> , <**INT** lockindicator>)

This creates a scroll window in which it will perform a view against a background graphic. That is, by using a graphic bigger than the display window, a part of this graphic can be shown and shifted in any direction. After this function, the use of the struct scroll makes sense.

250.2 Parameters

INT scrollnumber	- The ID for the new scroll window, so it can be referenced to later
INT fileID	- The fileID of the file containing the scroll graphics
INT graphID	- The graphID of the graphic of the main graphic to be scrolled
INT backgroundgraphID	- The graphID of the graphic for the background of the scroll window
INT regionnumber	- The region in which to put the scroll window
INT lockindicator	- A bit flag defining whether each of the two scroll planes is horizontally/vertically cyclical

250.3 Returns

INT : true

250.4 Notes

The locking indicator can be combinations of the following flags:

1 - The foreground will be displayed horizontally cyclical
2 - The foreground will be displayed vertically cyclical
4 - The background will be displayed horizontally cyclical
8 - The background will be displayed vertically cyclical

Combine them using the bitwise OR operator.

250.5 Using Scrolling

For each process that you want to be part of a scroll window, you must set the local variable ctype to value C_SCROLL. It should also be noted that the local variable c_number is used for selecting in which scroll a process should be displayed. Additionally, you must set the camera property of the scroll structure to the processID of the process you wish to be followed.

251 Strcasecmp

251.1 Definition

INT strcasecmp(<**STRING** str1> , <**STRING** str2>)

Compares two strings case-insensitive and returns the result.

251.2 Parameters

STRING str1 - The first string.
STRING str2 - The second string, to compare with the first string.

251.3 Returns

INT: difference

0 - The strings are equal.
>0 - The ASCII value of the first differing characters is higher for *str1*.
<0 - The ASCII value of the first differing characters is higher for *str2*.

251.4 Notes

If the strings differ, the ASCII difference between the first differing characters of the strings is actually returned. Let *i* be the index of the differing characters, then what is returned: *str1*`[i]`-*str2*`[i]`.

251.5 Example

```
import "mod_string"
import "mod_say"

Const
    AB = "AB";
    ABC = "ABC";
    CD = "CD";
    CD2 = "CD";
End

Process Main()
Begin

    say("strcasecmp(AB,ABC) = " + strcasecmp(AB,ABC));
    say("strcasecmp(AB,CD) = " + strcasecmp(AB,CD));
    say("strcasecmp(CD,AB) = " + strcasecmp(CD,AB));
    say("strcasecmp(CD,CD2) = " + strcasecmp(CD,CD2));

End
```

Used in example: say(), **strcasecmp**()

Result:

```
strcasecmp(AB,ABC) = -67
strcasecmp(AB,CD) = -2
strcasecmp(CD,AB) = 2
strcasecmp(CD,CD2) = 0
```

Template:Moduledocbox

252 Strrev

252.1 Definition

STRING strrev (<**STRING** str>)

Returns a reversed version of a certain string, meaning the characters are in reversed order.

252.2 Parameters

STRING str - The non reversed string.

252.3 Returns

STRING : The reversed string.

Template:Funcbox

253 Substr

253.1 Syntax

INT substr (<**STRING** str> , <**INT** startposition> , [<**INT** characters>])

253.2 Description

Returns a subsection of a certain string.

253.3 Parameters

STRING str	- The string of which a subsection will be returned.
INT startposition	- The position of the first character to be in the subsection.
[**INT** characters]	- The number of characters the subsection will hold. Negative values are special; see Notes.

253.4 Returns

STRING : The subsection.

253.5 Notes

If the number of characters is a negative value, the following applies: the start of the subsection will be *startposition*; the end of the subsection will be the length of the string minus the absolute value of *characters*.

253.6 Example

```
Private
    String str = "This is my string.";
Begin

    // No specified number of characters
    say( substr(str,2)     + "<" ); // "is is my string."
    say( substr(str,-7)    + "<" ); // "string."

    // Specified number of characters
    say( substr(str,5,2)   + "<" ); // "is"

    // Number of characters greater than length of string
    say( substr(str,2,50)  + "<" ); // "is my string."
    say( substr(str,-7,50) + "<" ); // "string."

    // Negative number of characters
    say( substr(str,5,-5)  + "<" ); // "is my st"

    // Special case
    say( substr(str,0,0)   + "<" ); // "", but pre 0.92: "This is my string."

    Repeat
        frame;
    Until(key(_ESC))

End
```

Used in example: say(), key()

Template:Funcbox

254 Tan

254.1 Definition

FLOAT tan (<**FLOAT** angle>)

Returns the tangent of a certain angle.

This function performs a tangent calculation on a certain angle and returns a value.

254.2 Parameters

FLOAT angle - Angle, in thousandths of degrees. i.e. 75000 = 75º

254.3 Returns

FLOAT : The tangent result of the specified angle.

254.4 Notes

The angle value used in this function should be in thousandths of degrees, as most angles within Bennu are.

To read about all aspects of trigonometry, you can visit Wikipedia's Trigonometric function page.

254.5 Example

```
Const
    screen_width  = 320;
    screen_height = 200;
    screen_border = 15;
End

Global
    float value;
End

Process Main()
Begin

    // Modes
    set_title("Tangent Graph");
    set_mode(screen_width,screen_height);

    // X axis
    for(x=1;x<=8;x++)
        write( 0,
               screen_border+x*(screen_width-screen_border)/8+3,
               screen_height-1,
               8,
               itoa(x*360/8 )+"^" );
    end
    draw_line(1,screen_height-screen_border,screen_width,screen_height-screen_border);

    // Y axis
    write(0,screen_border-1,20,5,"1");
    write(0,screen_border-1,screen_height/2,5,"0");
    write(0,screen_border-1,screen_height-20,5,"-1");
    draw_line(screen_border,1,screen_border,screen_height-1);

    // Draw tangent
    for(angle=0;angle<360;angle++)
        value=tan(angle*1000)*(screen_height/2-20);
        put_pixel( screen_border+angle*(screen_width-screen_border)/360,
                   screen_height/2-value,
                   rgb(255,255,255) );
        // screen_height/2-value because the screen's origin (0,0) is topleft instead of downleft.
    end

    Repeat
        frame;
    Until(key(_ESC))

End
```

Used in example: set_title(), set_mode(), write(), draw_line(), **tan**(), put_pixel(), key()

This will result in something like:
Template:Image

Template:Funcbox

255 Text height

255.1 Definition

INT text_height (<**INT** fontID> , <**STRING** text>)

Calculates the height in pixels of the specified text in the specified font.

255.2 Parameters

INT FontID	- FontID of the font for which the height of the text will be the calculated.
STRING text	- The text of which the height will be calculated.

255.3 Returns

INT : The height in pixels of the text in the font.

0	- Invalid or no text; invalid font.
>0	- The height in pixels of the text in the font.

255.4 See also

- text_width()

256 Text width

256.1 Definition

INT text_width (<**INT** fontID> , <**STRING** text>)

Calculates the width in pixels of the specified text in the specified font.

256.2 Parameters

INT FontID	- FontID of the font for which the width of the text will be the calculated.
STRING text	- The text of which the width will be calculated.

256.3 Returns

INT : The width in pixels of the text in the font.

0	- Invalid or no text; invalid font.
>0	- The width in pixels of the text in the font.

256.4 See also

- text_height()

257 Time

257.1 Syntax

INT time ( )

257.2 Description

Returns the current time, in seconds from January 1st, 1970.

This function is mostly useful for the function ftime(), to display time and date in a particular format. It is also useful in rand_seed(), to have 'more randomness'.

257.3 Returns

INT : The current time, in seconds from January 1st, 1970.

257.4 Example

```
import "mod_time"
import "mod_timer"
import "mod_text"
import "mod_key"

Process Main();
Private
    String timestring; // The string holding the formatted time
Begin

    write_string(0,0,0,0,&timestring); // Display the timestring
    timer = 100; // Make it so it updates the timestring immediately

    Repeat
        if(timer>100) // Update the timestring every 1 second
            timer = 0;
            timestring = ftime("%d-%m-%Y %H:%M:%S",time());
        end
        frame;
    Until(key(_esc))

End
```

Used in example: write_string(), ftime(), key(), timer

Template:Moduledocbox

258 Trim

258.1 Definition

STRING trim (<**STRING** str>)

Returns a copy of *str* without leading or ending whitespace (spaces, tabs, linefeeds, carriage returns).

258.2 Parameters

STRING str - The string to trim.

258.3 Returns

STRING: trimmed string

258.4 Example

```
import "mod_string"
import "mod_say"

Const
    _ABC_ = " ABC ";
End

Process Main()
Private
    string ABC;
Begin

    ABC = trim(_ABC_);
    say('_ABC_ = "' + _ABC_ + '"');
    say('ABC = "' + ABC + '"');

End
```

Used in example: say(), **trim**()

Result:

```
_ABC_ = " ABC "
ABC = "ABC"
```

Template:Moduledocbox

259 Ttf load

259.1 Syntax

INT ttf_load (<**STRING** filename> , <**INT** height>)

259.2 Description

Loads a TTF file as a font into memory.

Also called **load_ttf**().

259.3 Parameters

STRING filename	- The filename of the TTF file that you wish to load (including extension and possible path).
INT height	- The height in pixels of the to be created font, a size indication.

259.4 Returns

INT : FontID

-1	- Error: file does not exist; insufficient memory; failed to init freetype.dll; error while loading file; error creating new font.
0	- Invalid filename.
>0	- The FontID.

259.5 Errors

Insufficient memory	- There is insufficient memory available. This error doesn't occur often.
Failed to init freetype.dll	- There was an error initializing freetype.dll
Error loading file	- There occurred an error while trying to load the file.
Error creating new font	- There occurred an error while trying to create a new font.

259.6 Notes

This function gets a TrueType font and creates a new font with generated glyphs based on the recovered font in the standard characters set (ISO-8859-1). These glyphs are generated with two colours, being colour 0 (transparent) and the color last set by set_text_color(). The first one is the background colour and the last one is the colour of the characters self.

It's possible to save the loaded font as FNT with save_fnt(). This way the font can be reused on platforms not having mod_ttf. Note that a FNT file only has information about the font for one size, while a TTF file has information for any size.

The specified size is only an indication; some characters may be a little higher in fact.

259.7 Example

```
import "mod_key"
import "mod_proc"
import "mod_map"
import "mod_video"
import "mod_ttf"
import "mod_text"
global
  int id_ttfs[2];
Begin
    set_mode(320,240,16);
    id_ttfs[0] = load_ttf("resources/presdntn.ttf",48);
    id_ttfs[1] = load_ttfaa("resources/presdntn.ttf",48,16,RGB(255,0,0),RGB(0,255,0));
    write(id_ttfs[0],20,10,0,"Hello");
    set_text_color(RGB(255,0,255));
    write(id_ttfs[0],150,10,0,"World");
    write(id_ttfs[1],20,100,0,"Bye");
    Repeat
        frame;
    Until(key(_ESC))
```

```
    unload_fnt(id_ttfs[0]);
    unload_fnt(id_ttfs[1]);

    exit();
End
```

Used in example: set_mode(),set_text_color(), rgb(), load_ttf(),load_ttfaa(), write(), key(), unload_fnt()

Template:Moduledocbox

260 Ttf loadaa

260.1 Syntax

INT ttf_loadaa (<**STRING** filename> , <**INT** height> , <**INT** colordepth> , <**INT** backgroundcolor> , <**INT** textcolor>)

260.2 Description

Loads a TTF file as a font into memory.

Also called **load_ttf**().

260.3 Parameters

STRING filename	- The filename of the TTF file that you wish to load (including extension and possible path).
INT height	- The height in pixels of the to be created font, a size indication.
INT colordepth	- The colordepth of the to be created font (1,8 or 16).
INT backgroundcolor	- The background color of the to be created font.
INT textcolor	- The text color of the to be created font.

260.4 Returns

INT : FontID

-1	- Error: file does not exist; insufficient memory; failed to init freetype.dll; error while loading file; error creating new font.
0	- Invalid filename.
>0	- The FontID.

260.5 Errors

Insufficient memory	- There is insufficient memory available. This error doesn't occur often.
Failed to init freetype.dll	- There was an error initializing freetype.dll
Error loading file	- There occurred an error while trying to load the file.
Error creating new font	- There occurred an error while trying to create a new font.

260.6 Notes

This function gets a TrueType font and creates a new font with generated glyphs based on the recovered font in the standard characters set (ISO-8859-1). These glyphs are generated with two colours, being the specified colours.

It's possible to save the loaded font as FNT with save_fnt(). This way the font can be reused on platforms not supporting mod_ttf. Note that a FNT file only has information about the font for one size, while a TTF file has information for any size.

The specified size is only an indication; some characters may be a little higher in fact.

The parameters *backgroundcolor* and *textcolor* only matter for 8 and 16 bit colordepths. If you specify a colordepth of 1, then the returned font will be black (background) and white (textcolour). The advantage of this, is that you can use change the colour any time, even after loading, with set_text_color().

260.7 Example

```
import "mod_key"
import "mod_map"
import "mod_video"
import "mod_ttf"
import "mod_text"
global
  int id_ttf;
Begin
    set_mode(320,240,16);
    id_ttf = load_ttfaa("resources/presdntn.ttf",48,16,RGB(255,0,0),RGB(0,255,0));
    write(id_ttf,20,10,0,"Hello World");
    Repeat
```

```
        frame;
    Until(key(_ESC))
    unload_fnt(id_ttf);
End
```

Used in example: set_mode(),**ttf_loadaa**(), rgb(), write(), key(), unload_fnt()

Template:Moduledocbox

261 Ttf loadx

261.1 Syntax

INT ttf_loadx (<**STRING** filename> , <**INT** height> , <**INT** colordepth> , <**INT** backgroundcolor> , <**INT** textcolor> , <**INT** gradient_start>)

261.2 Description

Loads a TTF file as a font into memory.

261.3 Parameters

STRING filename	- The filename of the TTF file that you wish to load (including extension and possible path).
INT height	- The height in pixels of the to be created font, a size indication.
INT colordepth	- The colordepth of the to be created font (1,8 or 16).
INT backgroundcolor	- The background color of the to be created font.
INT textcolor	- The text color of the to be created font.
INT gradient_start	- At which gradient the drawing starts. Higher *gradient_start* means 'thinner' font and less anti-aliased. Range `0..255`.

261.4 Returns

INT : FontID

-1	- Error: file does not exist; insufficient memory; failed to init freetype.dll; error while loading file; error creating new font.
0	- Invalid filename.
>0	- The FontID.

261.5 Errors

Insufficient memory	- There is insufficient memory available. This error doesn't occur often.
Failed to init freetype.dll	- There was an error initializing freetype.dll
Error loading file	- There occurred an error while trying to load the file.
Error creating new font	- There occurred an error while trying to create a new font.

261.6 Notes

This function gets a TrueType font and creates a new font with generated glyphs based on the recovered font in the standard characters set (ISO-8859-1). These glyphs are generated with two colours, being the specified colours.

It's possible to save the loaded font as FNT with save_fnt(). This way the font can be reused on platforms not supporting mod_ttf. Note that a FNT file only has information about the font for one size, while a TTF file has information for any size.

The specified size is only an indication; some characters may be a little higher in fact.

The parameters *backgroundcolor* and *textcolor* only matter for 8 and 16 bit colordepths. If you specify a colordepth of 1, then the returned font will be black (background) and white (textcolour). The advantage of this, is that you can use change the colour any time, even after loading, with set_text_color().

261.7 Example

```
import "mod_key"
import "mod_map"
import "mod_video"
import "mod_ttf"
import "mod_text"
global
  int id_ttf;
Begin
    set_mode(320,240,16);
    id_ttf = load_ttfx("resources/presdntn.ttf",48,16,RGB(255,0,0),RGB(0,255,0),1);
    write(id_ttf,20,10,0,"Hello World");
    Repeat
        frame;
```

```
    Until(key(_ESC))
    unload_fnt(id_ttf);
End
```

Used in example: set_mode(),**ttf_loadx**(), rgb(), write(), key(), unload_fnt()

Template:Moduledocbox

262 Ucase

262.1 Definition

STRING ucase (<**STRING** str>)

Returns a string identical to a certain string, with the exception that all lowercase characters are replaced by their uppercase counterparts.

262.2 Parameters

STRING str - The string in "normal"-form.

262.3 Returns

STRING : The string in "uppercase"-form.

Template:Funcbox

263 Unload song

263.1 Definition

INT unload_song (<**INT** SongID>)

Frees the memory occupied by the song file, previously loaded with load_song().

263.2 Parameters

INT SongID - The SongID of the song to unload.

263.3 Returns

INT : Error.

-1 - Error: sound inactive; invalid songID.
0 - No error.
Template:Funcbox

264 Write

264.1 Definition

INT write (<**INT** fontID> , <**INT** x> , <**INT** y> , <**INT** alignment> , <**STRING** text>)

Puts a dynamic text with a certain font on certain coordinates on the screen with a certain alignment.

264.2 Parameters

INT fontID	- The FontID of the font to be used for the text.
INT x	- The X coordinate of the text.
INT y	- The Y coordinate of the text.
INT alignment	- The type of alignment.
STRING text	- The text to be used.

264.3 Returns

INT : TextID

-1	- Error. The text could not be obtained or was empty.
>=0	- The TextID of the text.

264.4 Notes

There is a limit of 511 texts to simultaneously exist on the screen. The program will crash with an error when this number is reached.

The text depth can be changed by adjusting the global variable text_z.

To write variables to the screen, rather use write_int(), write_string(), write_float() or write_var() than this command.

To write text on a map you can use the command write_in_map().

If you write texts with a font and you change any symbol of this font after, all written texts will be updated using the new changed symbols.

264.5 Example

```
Program texts;
Const
    maxtexts = 10;
Private
    int textid[maxtexts-1];
    string str;
    float flt;
Begin

    // Set FPS
    set_fps(60,0);

    // Write some texts
    textid[0] = write(0,0,0,0,"FPS:");
    textid[1] = write_int(0,30,0,0,&fps);
    textid[2] = write_string(0,160,95,1,&str);
    textid[3] = write_float(0,160,105,0,&flt);

    // Update the texts until ESC is pressed
    Repeat
        // Notice the texts get updated as the values of str and flt changes.
        // The same goes for the value of fps.
        str = "This program is running for " + timer/100 + " seconds.";
        flt = (float)timer/100;
        frame;
    Until(key(_esc));

    // Delete the texts
    for(x=0; x<maxtexts; x++)
        if(textid[x]!=0)
            delete_text(textid[x]);
        end
    end
```

```
End
```

Used in example: set_fps(), write_int(), write_string(), write_float(), key(), delete_text(), array, fps, TextID

This will result in something like:
Template:Image

265 Write float

265.1 Definition

INT write_float (<**INT** fontID> , <**INT** x> , <**INT** y> , <**INT** alignment> , <**FLOAT POINTER** var>)

Writes a floating point variable to the screen, which will be automatically updated when the value of the variable changes. The floating point variable will remain on the screen until deleted with delete_text().

265.2 Parameters

INT fontID	- The FontID of the font to be used for the text.
INT x	- The X coordinate of the text.
INT y	- The Y coordinate of the text.
INT alignment	- The type of alignment.
FLOAT POINTER var	- A pointer to a floating point variable.

265.3 Returns

INT : TextID

-1	- Error.
>=0	- The TextID of the text.

265.4 Notes

There is a limit of 511 texts to simultaneously exist on the screen. The program will crash with an error when this number is reached.

The text depth can be changed by adjusting the global variable text_z.

Instead of **write_float**(), write_var() can be used for the same purpose, which is a more general function that allows you to write variables of any type to the screen.

265.5 Errors

Too many texts onscreen	- There are too many texts on the screen.

265.6 Example

```
Program test;
Private
    float my_float=3.14159265;
Begin

    write_float(0,320/2,200/2,4,&my_float);

    Repeat
        Frame;
    Until(key(_ESC))

End
```

This will result in something like:
File:Write float.png

266 Write in map

266.1 Definition

INT write_in_map (<**INT** fontID> , <**STRING** text> , <**INT** alignment>)

Creates a new graphic in memory with the given text on it (without borders around the text) and puts it in the system file.

266.2 Parameters

INT fontID	- The FontID of the font to be used for the text.
STRING text	- The text to be used.
INT alignment	- The type of alignment.

266.3 Returns

INT : GraphID

0	- Error. The text could not be obtained or was empty.
!0	- The GraphID of the graphic of the generated graphic in the system file.

266.4 Notes

This function creates a graphic containing the specified font, with a width and height determined by the physical size of the text; the graphic's size will fit the text exactly to the pixel. The graphic will be stored in memory with FileID 0 (using the system file), and can be obtained at any time by calling its GraphID. The graphic can also be unloaded from memory by using unload_map().

The centre of the graph (control point 0) is given according to the given alignment. This gives added functionality of being able to place the graph like texts, yet also using flags, alpha, rotation, collision(), etc.

Processes can adopt the graphic containing the text, or it can be displayed with some maps functions, creating a very handy function.

266.5 Errors

Invalid font	- The specified font does not exist or is invalid.

266.6 Example

```
Program example;
Begin
    Set_text_color(rgb(222,195,140));
    graph=write_in_map(0,"Game programming is awesome!",4);
    repeat
        x=mouse.x;
        y=mouse.y;
    frame;
    until(key(_esc))
End
```

Used in example: set_text_color(), rgb(), key(), x, y, mouse

This will result in something like:
File:Write in map.png

267 Write int

267.1 Definition

INT write_int (<**INT** fontID> , <**INT** x> , <**INT** y> , <**INT** alignment> , <**INT POINTER** var>)

Writes an integer to the screen, which will be automatically updated when the value of the integer changes. The integer will remain on the screen until deleted with delete_text().

267.2 Parameters

INT fontID	- The FontID of the font to be used for the text.
INT x	- The X coordinate of the text.
INT y	- The Y coordinate of the text.
INT alignment	- The type of alignment.
INT POINTER var	- A pointer to an integer.

267.3 Returns

INT : TextID

-1	- Error.
>=0	- The TextID of the text.

267.4 Notes

There is a limit of 511 texts to simultaneously exist on the screen. The program will crash with an error when this number is reached.

The text depth can be changed by adjusting the global variable text_z.

Instead of **write_int**(), write_var() can be used for the same purpose, which is a more general function that allows you to write variables of any type to the screen.

267.5 Errors

Too many texts onscreen	- There are too many texts on the screen.

267.6 Example

```
Program test;
Private
    my_integer=0;
Begin

    write_int(0,320/2,200/2,4,my_integer);

    Repeat
        my_integer=rand(1,1000);
        frame;
    Until(key(_ESC))

End
```

Used in example: rand(), key()

This will result in something like:
File:Write int.png

268 Write string

268.1 Syntax

INT write_string (<**INT** fontID> , <**INT** x> , <**INT** y> , <**INT** alignment> , <**STRING POINTER** var>)

268.2 Description

Writes a string to the screen, which will be automatically updated when the value of the string changes. The string will remain on the screen until deleted with delete_text().

268.3 Parameters

INT fontID	- The FontID of the font to be used for the text.
INT x	- The X coordinate of the text.
INT y	- The Y coordinate of the text.
INT alignment	- The type of alignment.
STRING POINTER var	- A pointer to a string.

268.4 Returns

INT : TextID

-1	- Error. The text could not be obtained or was empty.
>=0	- The TextID of the text.

268.5 Notes

There is a limit of 511 texts to simultaneously exist on the screen. The program will crash with an error when this number is reached.

The text depth can be changed by adjusting the global variable text_z.

Instead of **write_string**(), write_var() can be used for the same purpose, which is a more general function that allows you to write variables of any type to the screen.

268.6 Errors

Too many texts onscreen	- There are too many texts on the screen.

268.7 Example

```
import "mod_text"
import "mod_key"

Global
    string my_string="Bennu Game Development";
End
Process Main()
Begin

    write_string(0,320/2,200/2,4,&my_string);

    Repeat
        frame;
    Until(key(_ESC))

End
```

Used in example: **write_string**(), key()

This will result in something like:
File:Write string.png

269 Write var

269.1 Syntax

INT write_var (<**INT** fontID> , <**INT** x> , <**INT** y> , <**INT** alignment> , <**VARSPACE** var>)

269.2 Description

Writes a variable of any kind to the screen, which will be automatically updated when the value of the variable changes. The variable will remain on the screen until deleted with delete_text().

269.3 Parameters

INT fontID	- The FontID of the font to be used for the text.
INT x	- The X coordinate of the text.
INT y	- The Y coordinate of the text.
INT alignment	- The type of alignment.
VARSPACE var	- The name of any variable.

269.4 Returns

INT : TextID

-1	- Error.
>=0	- The TextID of the text.

269.5 Notes

Please note that for the varspace **var** parameter *no* pointer should be filled out, while write_int()/write_string()/write_float() *do* require a pointer.

There is a limit of 511 texts to simultaneously exist on the screen. The program will crash with an error when this number is reached.

The text depth can be changed by adjusting the global variable text_z.

Instead of **write_var**(), write_int() could be used to write an int to the screen, write_string() for a string, write_float() for a float.

269.6 Errors

Too many texts onscreen	- There are too many texts on the screen.

269.7 Example

```
import "mod_text"
import "mod_key"

Global
    my_integer=0;
    string my_string="Bennu Game Development";
End

Process Main()
Begin

    write_var(0,0,0,0,my_string);
    write_var(0,320/2,200/2,4,my_integer);

    Repeat
        my_integer=rand(1,1000);
        frame;
    Until(key(_ESC))

End
```

Used in example: **write_var**(), rand(), key()

This will result in something like:
File:Write var.png

270 Xadvance

270.1 Definition

INT xadvance (<**INT** angle> , <**INT** distance>)

Moves a process a certain distance in a certain direction.

270.2 Parameters

INT angle	- The angle in which to move the process, in thousandths of a degree.
INT distance	- The distance to move the process, in pixels.

270.3 Returns

INT : Successrate

true	- Success.
false	- Error.

270.4 Example

```
Program example;
Global
    myproc;

Begin

    myproc=proc();

    Loop
        frame;
    End
End

Process proc();
Begin

    x=100;
    y=100;

    Loop
        xadvance(90000,10); //moves the process vertically (90 degrees) 10 pixels
        frame;
    End

End
```

Template:Moduledocbox

271 Xput

271.1 Definition

INT xput (<**INT** fileID> , <**INT** GraphID> , <**INT** x> , <**INT** y> , <**INT** angle> , <**INT** size> , <**INT** blitflags> , <**INT** region>)

Draws (blits) a graphic onto the background.

If the advanced parameters aren't needed, put() can be used.

271.2 Parameters

INT fileID	- The fileID of the file that holds the graphics.
INT graphID	- The graphID of the graphic to draw with.
INT x	- Where on the background graphic's x-axis to put the graphic.
INT y	- Where on the background graphic's y-axis to put the graphic.
INT angle	- What angle to draw the graphic at.
INT size	- What size to draw the graphic at.
INT blitflags	- What blit flags to draw the graphic with.
INT regionID	- The regionID of the region in which the graphic is only allowed to be drawn.

271.3 Returns

INT : true

271.4 Notes

The x and y parameters denote where to draw the graphic, that is, where the center of the to be drawn graphic will be. Blit flags can be used to give the drawing (blitting) a special effect.

When angle is 0 and size is 100, the speed is greater, because the graph doesn't need rotating or scaling.

271.5 Errors

Unsupported color depth	- The graphic's color depth is greater than the background graphic's.

Template:Moduledocbox