# Content - ScriptCad Help

# Introduction

Program *ScriptCad* is drawing tool for 2D projects. Program work under Microsoft Windows (Windows 95/98/98SE/ME/2000/XP) and was programmed in Delphi 5 Professional.

*ScriptCad* has another style of work than other CAD programs. In other programs, there is prefered visual design. So, user work with mouse, he is clicking at buttons (with symbols of geometrical units) in toolbars. For all *ScriptCad* supported this way too (see section Visual design), it is not prefered way. The primary way for drawing in *ScriptCad* program is command line programming. Complete index of commands is in section Commands and their description.

# Menu of program

We write drawing commands (their complete description is in section Commands and their description) in left part of main window (script area). Then, we can click at *Refresh* button (or choose menu *Drawing-Refresh*) and all drawing is draw again.

**Menu Script**
Commands from left part of main window can be saved to text files (so called scripts). These scripts has *.vic* extension. By menu *Script,* you can save and open scripts and you can choose font for display scripts in script area too.

**Menu Drawing**
Next section of menu is *Drawing*. By this section of menu, you can refresh drawing (*Drawing-Refresh*), you can clear all drawing area, you can hide/show layers (*Drawing-Show layers*) or you can save drawing as picture (*Drawing-Save drawing as*) or you can print drawing on printer (*Drawing-Print drawing*).

You can save drawing as three types of pictures (BMP, JPG a GIF). Raster (resolution) of picture has the same size as size drawing area is. For enlarging pictures, you can resize drawing area first. If you enlarge pictures in some graphic editor, quality will be lower. For enlarging drawing area in *ScriptCad* program, you can use command *PixelsPerUnit* - see section Commands and their description. In addition, you can choose colour depth for saving pictures (see section Program setting-up - parameter *BitsPerPixel*).

If you print drawing on printer, drawing is stretched for all printing area (for example A4). With-height ratio is keeping, of course. If quality of printed drawing is unsatisfactory, you can experiment with parameter *Stretch*, see section Program setting-up.

**Menu Help**
By this section of menu, you can run help file for program (*Help-Content*) or you can find in this file (*Help-Find*). You can run *About* window (*Help-About*) or you can switch among languages (*Help-Language*).

The shortest way for open help file is by pressing key *F1.*

# Visual design

For visual design of drawing, there is context menu of drawing area (right part of main window). You can choose item from this menu by right mouse click above of drawing area. Each item has differential number of clicks (for building appropriate object). Some items (objects) do not need additional clicks - e.g. Comment. Some items (objects) needs one additional click - e.g. Point. Other items (objects) needs two additional clicks - e.g. Line. First click correspond begin of Line and second click correspond end of Line.

Context menu has these items:

- **Line**
Command for draw Line is added to script area (left part of main window). For complete action, you must click to drawing area twice times. First click correspond begin of Line and second click correspond end of Line. After it, command is assembled, written to script area and runed on drawing area.

- **Dimension Line**
Command for draw Dimension Line is added to script area. For complete action, you must click to drawing area twice times.

- **Text**
Command for writing Text is added to script area. For complete action, you must click to drawing area once time. The content of text may be changed in script area.

- **Point**
Command for draw Point is added to script area. For complete action, you must click to drawing area once time.

- **Fill Area**
Hatch enclosed area. For complete action, you must click to drawing area once time. Area is enclosed by points which have another colour.

- **Rectangle**
Command for draw Rectangle is added to script area. For complete action, you must click to drawing area twice times. First click correspond left-top corner of Rectangle and second click correspond right-bottom corner of Rectangle.

- **Circle**
Command for draw Circle is added to script area. For complete action, you must click to drawing area twice times. First click correspond circle-centre and second click correspond some point at circumference of circle.

- **Part of Circle**
Command for draw Part of Circle is added to script area. For complete action, you must click to drawing area twice times. First click correspond circle-centre and second click correspond some point at circumference of circle part.

- **Ellipse**
Command for draw Ellipse is added to script area. For complete action, you must click to drawing area twice times. First click correspond left-top corner and second click correspond right-bottom corner of rectangle - Ellipse is inscribed in this rectangle.

- **Picture**
Command for insert of picture is added to script area. Path to picture may be absolute path (e.g. "C:\SomeFolder\example.gif") or relative path (e.g. "example.gif"). If path is relative, then picture is searched

either in folder with program *ScrptCad.exe* (when script is not saved) or in folder with *.vic* script, which is executed.

- **Global variables**

Section of Global variables is added to script area. This section set up the most important parameters for (future) drawing. Every drawings may have this section on the beginning. For more information about Global variables, see section Global variables.

- **Layer**

Command for switch layer is added to script area. Number of layer must be enter into script. Valid numbers are integers between 1 and 9.

- **Procedure**

Command for definition of procedure is added to script area. Then, you can change Procedure name and number of variables and you can write procedure body. For more information about Procedures, see section Procedures as a definitions of new objects.

- **Comment**

Comment is added to script area. Comment is row, what has letter *%* (per cent) at the beginning.

- **Cancel drawing action**

Cancel semifinished drawing action.

**NOTE:** After creating object by above explained methods, you can modify parameters of commands in script area (left part of main window). In script area, you can delete commands or you can create commands, that is explained in section Commands and their description. After editing of commands in script area, you can re-draw drawing by *Drawing-Refresh.*

**NOTE:** Visual design of drawing is the easiest way to draw, but has some disadvantages. Problem is accuracy of drawing. In addition, direct writing of commands is more flexible. More information about direct writing of commands is in section Commands and their description.

# Commands and their description

Visual design of drawing (see section Visual design) is the easiest way to draw, but has some disadvantages. Problem is accuracy of drawing. In addition, direct writing of commands is more flexible.

**Co-ordinate system**
Program *ScriptCad* is tool for 2D drawing. So, co-ordinate system has two axises - *x* and *y*. **Initial point of co-ordinate system** (the point [0,0]) **is placed in left-top corner** of drawing area.

X-axis is (horizontal) top edge of drawing area and x-distance is growing from left to right (then maximal values of x-distance are in right (vertical) edge of drawing area and x-distance is 0 (zero) in left (vertical) edge of drawing area).

Y-axis is (vertical) left edge of drawing area and y-distance is growing from top to bottom (then maximal values of y-distance are in bottom (horizontal) edge of drawing area and y-distance is 0 (zero) in top (horizontal) edge of drawing area).

When mouse cursor is mooving through drawing area, the actual position of mouse (x,y) is displayed at the bottom part of main window. Co-ordinate values are measured in units, which are defined by global variable *PixelsPerUnit* (more informations about this variable and about other global variables are in section Global variables).

**Global variables**
Each drawing may have this section on the beginning because global variables set up the most important parameters for (future) drawing. These global variables are very important, then they have independent section Global variables.

**Comment**
Comment is row, what has letter *%* (per cent) at the beginning. When letter *%* is not at the beginning of row (line), then the first part of row is command and the second part of row is comment. All characters from letter *%* are ignored (for drawing).

Commands for geometrical units are

- Line - command Line
- Dimension Line - command DimLine
- Text - command Text
- Point - command Point
- Rectangle - command Rectangle
- Fill Area - command FillArea
- Circle - command Circle
- Part of Circle - command CirclePart
- Ellipse - command Ellipse

# Global variables

Global variables set up the most important parameters for drawing.There are variables for set up size of drawing area (variables *MaximumX*, *MaximumY*, *PixelsPerUnit*), colour of background (variable *BackgroundColor*), default values for colour of pen (variable *DefaultColor*), font of texts (variable *DefaultFontName*), font size (variable *DefaultFontSize*), size of angle of arrows and size of arrows in Dimension Line (variables *DefaultDimAngle* and *DefaultDimLength*) and feed of drawing (variables *ShiftX* and *ShiftY*).

**MaximumX**
It set width of drawing area in units, which are defined by global variable *PixelsPerUnit.*

**MaximumY**
It set height of drawing area in units, which are defined by global variable *PixelsPerUnit.*

**PixelsPerUnit**
It set, how many pixels are one unit. For example, *PixelsPerUnit=10* mean, that one unit is *10 px.* All co-ordinates of geometrical shapes are measured by these units. Variable *PixelsPerUnit* enable zooming of drawing at any time. F.g. after double of *PixelsPerUnit* are doubled all sizes of drawing and drawing area is enlarged.

**BackgroundColor**
It set colour of background of drawing (default colour is *White*).

**DefaultColor**
It set colour of pen for drawing (default colour is *Black*). By this colour, all geometrical shapes are drafting (if command of the object do not set another colour).

**DefaultFontName**
It set font name (default font is *"Arial"*). Font name must be in quotation marks.

**DefaultFontSize**
It set font size (default is 8). If *PixelsPerUnit=1*, then this setting for *DefaultFontSize* is compliant. If you set *PixelsPerUnit* to greater value, you can set *DefaultFontSize* to lower value!

**DefaultDimAngle**
It set angle of arrow in Dimension Line. Default value is 10.

**DefaultDimLength**
It set size of arrow in Dimension Line. Default value is 10 units. If *PixelsPerUnit=1*, then this setting for *DefaultDimLength* is compliant. If you set *PixelsPerUnit* to greater value, you can set *DefaultDimLength* to lower value!

**ShiftX**
It move all drawing in way of x-axis (units depended on *PixelsPerUnit*). By this way, you can move all drawing without re-creating co-ordinate values in all objects. **Attention: After using ShiftX and ShiftY, the actual position of mouse (x,y), which is displayed at the bottom part of main window, is wrong.**

**ShiftY**
It move all drawing in way of y-axis.

**Note:** Commands (such as Line, Circle etc.) may use global variables *DefaultColor*, *DefaultDimAngle*, *DefaultDimLength*, *DefaultFontName* and *DefaultFontSize*. For example, if *DefaultColor=Blue*, then commands *Point 124, 169, DefaultColor* and *Point 124, 169, Blue* are the same.

**Attention:** Variables *MaximumX*, *MaximumY*, *PixelsPerUnit* and *BackgroundColor* are very important. If you re-set any variable (one from these four), then drawing area is deleted! Accordingly this feature, you must set these variables at the beginning of drawing (beginning of script).

**Note:** X-size of drawing in pixels is *MaximumX* * *PixelsPerUnit*. Y-size of drawing in pixels is *MaximumY* * *PixelsPerUnit*.

**Note:** If size of drawing is small, then visual defect may occur, because resolution is low. Visual defects will be eliminated by enlargement of variable *PixelsPerUnit*).

**Note:** Variable *PixelsPerUnit* must be adjusted after variables *MaximumX* and *MaximumY*!! Otherwise, following situation may arise: for example

```
PixelsPerUnit = 1
MaximumX = 500
MaximumY = 400
```

Against recommendation, *PixelsPerUnit* is in first place. Whats hapened? First run of script set size of drawing area to *500 px* x *400 px*. Then we change units and we write

```
PixelsPerUnit = 100
MaximumX = 5
MaximumY = 4
```

Now, size of drawing area is *500 px* x *400 px* too, but this method is wrong. Script is executed command by command, then first command *PixelsPerUnit=100* make area *50000 px* x *40000 px* (because still *MaximumX = 500* and *MaximumY = 400*). Consecutively, area will be reduce, but previous commands spend all memory yet. Then program may occur memory error!

# Line - command Line

| | |
|---|---|
| *Universal form:* | Line x1, y1, x2, y2, LineType, ThickLine, LineColour |
| *Minimum form:* | Line x1, y1, x2, y2 |
| *Example:* | Line 87, 71, 320, 226, Solid, 1, Blue |

| | |
|---|---|
| (x1, y1) | ... initial point of line (*1234*) |
| (x2, y2) | ... terminal point of line (*1234*) |
| LineType | ... see Line Types |
| ThickLine | ... see Thick of Line |
| LineColour | ... see Colours |

If some parameters are missing (compare universal form with minimum form), then LineType is *Solid,* ThickLine is *1* and LineColour is *DefaultColor* from global variables.

# Dimension Line - command DimLine

*Universal form:* DimLine x1, y1, ArrowType1, x2, y2, ArrowType2, LineType, ThickLine, LineColour, ArrowAngle, ArrowLength
*Minimum form:* DimLine x1, y1, ArrowType1, x2, y2, ArrowType2
*Example:* DimLine 104, 232, Inner, 310, 52, Inner, Solid, 1, Black, 10, 8

(x1, y1) ... initial point of Dimension Line (*1234*)
ArrowType1 ... Type of arrow at initial point
(x2, y2) ... terminal point of Dimension Line (*1234*)
ArrowType2 ... Type of arrow at terminal point
LineType ... see Line Types
ThickLine ... see Thick of Line
LineColour ... see Colours
ArrowAngle ... is angle between main line and arrow (at terminal point)
ArrowLength ... is length of arrow (in units defined by variable *PixelsPerUnit*) (*1234*)

If some parameters are missing, then LineType is *Solid,* ThickLine is *1*, LineColour is global variable *DefaultColor*, ArrowAngle is global variable *DefaultDimAngle* and ArrowLength is global variable *DefaultDimLength.*

# Text - command Text

*Universal form:* Text x, y, CharacterSet, IsBold,IsItalic,IsUnderline,IsStrikeOut, FontName, TextColour, FontSize, AngleOfText
*Minimum form:* Text x, y, CharacterSet
*Example:* Text 179, 225, "Text is here", 0,0,0,0, "Arial", Red, 12, 90

(x, y) ... point, where is initial letter of text (*1234*)
CharacterSet ... text, which will be displayed (all text must be in quotation marks) (texts may be joined "he" + "llo")
IsBold ... text is bold (IsBold=1) or standard (IsBold=0)
IsItalic ... text is slanted (IsItalic=1) or standard (IsItalic=0)
IsUnderline ... text is underlined (IsUnderline=1) or standard (IsUnderline=0)
IsStrikeOut ... text is striked (IsStrikeOut=1) or standard (IsStrikeOut=0)
FontName ... Font Name. It must be in quotation marks (f.g. "MS Sans Serif"). Font must be in operating system (Windows). You can display index of all accessible fonts by menu *Script-Font.*
TextColour ... see Colours
FontSize ... size of font (in units defined by variable *PixelsPerUnit*) (*1234*)
AngleOfText ... angle of text. You can write horizontal (AngleOfText=0) or vertical (AngleOfText=90) or you can choose another angle (in degrees).

If some parameters are missing, then IsBold=0, IsItalic=0, IsUnderline=0, IsStrikeOut=0, FontName is global variable *DefaultFontName,* TextColour is global variable *DefaultColor*, FontSize is global variable *DefaultFontSize* and AngleOfText is 0 degrees.

# Point - command Point

*Universal form:* Point x, y, Colour
*Minimum form:* Point x, y
*Example:* Point 201, 182, Green

(x, y) ... co-ordinate of point (*1234*)
Colour ... colour of point

If parameter Colour is missing, then Colour is global variable *DefaultColor.*

# Rectangle - command Rectangle

| | |
|---|---|
| *Universal form:* | Rectangle x1, y1, x2, y2, LineType, ThickLine, LineColour |
| *Minimum form:* | Rectangle x1, y1, x2, y2 |
| *Example:* | Rectangle 101, 233, 185, 293, Dot, 1, Gray |

| | |
|---|---|
| (x1, y1) | ... left-top corner of rectangle (*1234*) |
| (x2, y2) | ... right-bottom corner of rectangle (*1234*) |
| LineType | ... see Line Types |
| ThickLine | ... see Thick of Line |
| LineColour | ... see Colours |

If some parameters are missing, then LineType is *Solid,* ThickLine is *1* and LineColour is *DefaultColor* from global variables.

# Fill Area - command FillArea

*Universal form:* FillArea x, y, TypeOfHatch, ColourOfHatch
*Minimum form:* FillArea x, y
*Example:* FillArea 179, 75, BDiagonal, Gray

TypeOfHatch ... see Types of hatch
ColourOfHatch ... see Colours
(x, y) ... co-ordinate of point, which is into some closed area. Area is closed by points, which have another colour then initial point (x, y) (*1234*).

If some parameters are missing, then TypeOfHatch is Solid and ColourOfHatch is *DefaultColor* from global variables.

# Circle - command Circle

| | |
|---|---|
| *Universal form:* | Circle x, y, Radius, LineType, ThickLine, LineColour |
| *Minimum form:* | Circle x, y, Radius |
| *Example:* | Circle 269, 293, 79, Solid, 1, NavyBlue |

| | |
|---|---|
| (x, y) | ... middle of circle (*1234*) |
| Radius | ... radius of circle (in units defined by variable *PixelsPerUnit*) (*1234*) |
| LineType | ... see Line Types |
| ThickLine | ... see Thick of Line |
| LineColour | ... see Colours |

If some parameters are missing, then LineType is *Solid,* ThickLine is *1* and LineColour is *DefaultColor* from global variables.

# Part of Circle - command CirclePart

| | |
|---|---|
| *Universal form:* | CirclePart x, y, Radius, FromAngle, ToAngle, LineType, ThickLine, LineColour |
| *Minimum form:* | CirclePart x, y, Radius, FromAngle, ToAngle |
| *Example:* | CirclePart 69, 239, 60, 90, 180, Solid, 1, Black |

| | |
|---|---|
| (x, y) | ... middle of circle (*1234*) |
| Radius | ... radius of circle (in units defined by variable *PixelsPerUnit*) (*1234*) |
| FromAngle, ToAngle | ... it define, how long part of circle will be drawn (in degrees) (*1234*).<br>1st quadrant is 0-90 degrees,<br>2nd quadrant is 90-180 degrees,<br>3rd quadrant is 180-270 degrees,<br>4th quadrant is 270-360 degrees.<br>Only integers are acceptable. |
| LineType | ... see Line Types |
| ThickLine | ... see Thick of Line |
| LineColour | ... see Colours |

If some parameters are missing, then LineType is *Solid,* ThickLine is *1* and LineColour is *DefaultColor* from global variables.

# Ellipse - command Ellipse

| | |
|---|---|
| *Universal form:* | Ellipse x1, y1, x2, y2, LineType, ThickLine, LineColour |
| *Minimum form:* | Ellipse x1, y1, x2, y2 |
| *Example:* | Ellipse 125, 197, 191, 232, Solid, 1, Blue |

| | |
|---|---|
| (x1, y1) | ... left-top corner of rectangle, where ellipse is inscribed (*1234*) |
| (x2, y2) | ... right-bottom corner of rectangle (*1234*) |
| LineType | ... see Line Types |
| ThickLine | ... see Thick of Line |
| LineColour | ... see Colours |

If some parameters are missing, then LineType is *Solid,* ThickLine is *1* and LineColour is *DefaultColor* from global variables.

# Colours

In *ScriptCad* program, following colours are enabled (for geometrical objects):

Aqua, Black, Blue, DarkGray, Fuchsia, Gray, Green, LimeGreen, LightGray, Maroon, NavyBlue, OliveGreen, Purple, Red, Silver, Teal, White, Yellow.

**Note:** You can use not only concrete colour (f.g. Black), but you can use also global variable *DefaultColor*. For example, if *DefaultColor=Blue*, then commands *Point 124, 169, DefaultColor* and *Point 124, 169, Blue* are the same.

# Thick of Line

Thick of Line is measured by pixels (NOT by units, which are defined by global variable *PixelsPerUnit)!* Primary thick is *1 px* - then all line types are enabled. When thick of line is greater than *1,* only *Solid* type of line is enabled.

# Line Types

Following types of line are enabled in *ScriptCad* program:

Solid
Dash
Dot
DashDot
DashDotDot

**Note:** When thick of line is greater than *1,* only *Solid* type of line is enabled.

# Types of hatch

You can use following types of hatch:

Solid ... full hatch
BDiagonal ... diagonal hatch (hatch are uprised)
FDiagonal ... diagonal hatch (hatch are subsided)
Cross ... tiles
DiagCross ... diagonal tiles
Horizontal ... horizontal hatch
Vertical ... vertical hatch

## Types of arrows

You can use following three types of arrows:

| | |
|---|---|
| Inner | ... inner arrow in Dimension Line |
| Outer | ... outer arrow in Dimension Line |
| None | ... without arrow |

# Program setting-up (ScrptCad.ini)

All settings are in file *ScrptCad.ini*. This file is in the same directory as program *ScrptCad.exe.* File must be writable (not read-only), otherwise settings will not be saved.

*ScrptCad.ini* has 4 sections.

In section **[MainForm]**, there is saved size and position of main window of program *ScriptCad* on Windows Desktop (variables *Height, Width, Left* and *Top*), also size of left part of main window (script area) is saved (variable *Splitter*). For language settings, there is variable *Language* (1000 for English and 2000 for Czech). Variable *DemoScript* indicate, whether demonstration script *demo.vic* was ran at first run of program *ScriptCad.*

In section **[ScriptFont]**, there are information about font, which display scripts at script area.

In section **[Layers]**, there are information about showing/hiding of layers of drawing.

In section **[Print]**, there you can set print parameters.
Variable *BitsPerPixel* set colour depth for saving pictures/drawings *(Drawing-Save drawing as).* Default value is *BitsPerPixel=4.* It mean, that colour depth is 4-bit (i.e. 16 colours). It is enought, because program *ScriptCad* support only these colours. If you want to have greater colour depth, you can set variable *BitsPerPixel* to another value:

*BitsPerPixel=8* for 256 colours,
*BitsPerPixel=16* for 65536 colours
*BitsPerPixel=24* for 16777216 colours and
*BitsPerPixel=32* for 4294967296 colours.

Variable *Stretch* set, how many times drawing will be stretched (zoom) during printing.
Situation will be explained as example: let x-size of drawing (in right part of main window of *ScriptCad* program) is 100 pixels and x-resolution of printer is 6000 pixels (it depend on size of print page - f.g. A4 - and it depend also on print quality (dpi)). Y-co-ordinate of drawing do not reflected (it is simple example). On default setting *Stretch=4*, the drawing will be enlarged in program *ScriptCad* from initial size 100 px to 1500 px - this enlargement is vector (without damage of quality - this type of enlargement is similar as growing of global variable *PixelsPerUnit* (see section Global variables). Value 1500 px is four times lesser (because *Stretch=4*) than x-resolution of printer (now 6000 px). Enlarged drawing (1500 px) will be stretch 4 times at printer, so x-resolution will be 6000 px. This method is optimal, but you can also change variable *Stretch.* Range is from 1 (smooth picture (hatch chiefly)) to 10 (rough draft).

Sections *[MainForm], [ScriptFont]* and *[Layers]* are autamatically used and changed by program *ScriptCad*. But section *[Print]* is only for manual changes, because author of this program think, that these settings are low-level (only for advanced users). New values in section *[Print]* will be forcible at next start of *ScriptCad* program.

# Registration

Program *ScriptCad* is shareware application. Test period is 30 days. If you decide to keep this program after the test period, you must register the program at e-mail address **vincze@ji.cz**. One license (license per one computer) cost 10 USD. Registration key is a new file *license.dll,* what will be sended to you. This file must be placed in directory, where program *ScrptCad.exe* is. Registration key will be valid for all future versions of *ScriptCad* program, so buying upgrade will not be necessary. New versions of *ScriptCad* program are enabled at web address of the author: **http://vincze.czweb.org**.

# License agreement

The owner of this software is Ing.Roman VINCZE. The software product is licensed, not sold. The software product is protected by copyright laws and international copyright treaties, as well as other property laws and treaties. You may not analyze, decompile, or disassemble the software product or any component thereof. You may make copies of the software product, but only for reserve.

NO WARRANTY.
No warranty of any kind is expressed or implied. *ScriptCad* software is distributed "as is". You use it at your own risk.

NO LIABILITY FOR DAMAGES.
The author is not liable for any data loss, damages, loss of profits or any other kind of loss while using or misusing this software.

# Layers

Program *ScriptCad* support drawing into layers. You may have 9 layers (max). If you want to draw (for example) into 2nd layer, you can use command *Layer = NumberOfLayer (Layer = 2). NumberOfLayer* must be between 1 and 9. Example:

*command (draw into 1st layer)*
*Layer = 2*
*command (draw into 2nd layer)*
*command (draw into 2nd layer)*
*command (draw into 2nd layer)*
*Layer = 3*
*command (draw into 3rd layer)*
*Layer = 1*
*command (draw into 1st layer)*
*command (draw into 1st layer)*

Commands for global variables are forcible for all layers. For example, if you set variable *MaximumX*, then this setting is valid for rest drawing (all layers). Commands as Line or Rectangle etc. refer only for one layer.

For displaying/hiding layers, there is menu *Drawing-Show layers* or you can use checkboxes *Layers* on toolbar. You can use any combination from 9 layers.

**Note:** For adding command Layer into script, you can use context menu above of drawing area too.

**Note:** If you use visual design (drawing by context menu above of drawing area), then objects are drawn without reference to layers (at first). Until you press *Drawing-Refresh*, objects are drawn into right layers.

# Picture - command Picture

*Universal form:* Picture x, y, Path, Width, Height
*Minimum form:* Picture x, y, Path
*Example:* Picture 15, 100, "C:\SomeFolder\example.gif", 19, 19

(x, y) ... left-top corner of picture (in units defined by variable *PixelsPerUnit*) (*1234*)
Path ... picture path on harddisk (it must be in quotation marks)
Width ... picture width (in units defined by variable *PixelsPerUnit*) (*1234*)
Height ... picture height (in units defined by variable *PixelsPerUnit*) (*1234*)

Parameter *Path* may be absolute path (e.g. "C:\SomeFolder\example.gif") or relative path (e.g. "example.gif"). If path is relative, then picture is searched either in folder with program *ScrptCad.exe* (when script is not saved) or in folder with *.vic* script, which is executed.

You can insert these pictures: *bmp, gif, jpg, jpeg.*

Parameters *Width* and *Height* may be missing. Then, the picture is inserted with it's true resolution and it's width and height will be fixed, independent from variable *PixelsPerUnit.* If parameters *Width* and *Height* are presented, then the picture is stretched into defined area.

**NOTE:** Drawing area in *ScriptCad* program has 4-bit colour depth. Then, if inserted picture has more colours, it's colours will be reduced. If you want to keep more colours, you can set variable *BitsPerPixel* in file *ScrptCad.ini* in section *[Print]* - more information about it is in section Program setting-up.

# Calculated terms

Calculated term is term, which may be calculated as number. It may be number (e.g. *5* or *-3.2*) or it may be composite of numbers and elementary mathematical operations (plus, minus, times, divide and parenthesis):

3 + 5
3 * 2.3
3 - 2 * 5
3 / (1 + 2)

**Example:** If you write
*Line 100 + 50, 50 * 2, 300 - 100, 500/10, Solid, 1, DefaultColor*
then it is the same as
*Line 150, 100, 200, 50, Solid, 1, DefaultColor*

Calculated terms are important for Procedures as a definitions of new objects. In this documentation, there is used symbol (*1234*) for calculated terms (see individual commands).

# Procedures as a definitions of new objects

By procedures, you can create own objects for drawing. After create of procedure, you can use it as well as native command of program *ScriptCad.*

**Declaration of procedure**
Procedure begin by key-word *Procedure*, then there is procedure name and variables. Variables are segregated by commas and **their names must begin by #.** Next lines contain body of procedure (commands, which are drawn in procedure use). Definition of procedure finish by key-word *End.*

**Example of declaration of procedure**
You can see script *demo2.vic* for using procedures. This script contain next procedure (draw battery):

```
Procedure Battery #x, #y
   Line #x, #y-1.5, #x, #y+1.5, Solid, 1, DefaultColor
   Line #x + 0.3, #y - 3, #x + 0.3, #y + 3, Solid, 1, DefaultColor
   Text #x + 0.2, #y - 4.9, "+", 0,0,0,0, DefaultFontName, DefaultColor, DefaultFontSize, 0
   Line #x + 0.3, #y, #x + 1.5, #y, Solid, 1, DefaultColor
End
```

**Application of procedure**
After create of procedure, you can use it as well as native drawing command, so e.g. commands ...

```
Battery 18, 7
Battery 19.5, 7
Battery 21, 7
Battery 22.5, 7
```

... draw battery, which has four cells. Variables #x and #y are co-ordinates. You can create special text file with declarations of procedures and you can insert this script into another scripts by command @include.

# Including of scripts to another scripts - command @include

In section Procedures as a definitions of new objects is described using of procedures. You can create special text file with declarations of procedures and you can insert this script into another scripts.

*Universal form:* @include Path
*Example:* @include "C:\SomeFolder\myfile.vic"

*Path* is path of file on harddisk (it must be in quotation marks). If you write @*include "myfile.vic",* then all content of file *"myfile.vic"* will be placed at this position.

Parameter *Path* may be absolute path (e.g. "C:\SomeFolder\myfile.vic") or relative path (e.g. "myfile.vic"). If path is relative, then file is searched either in folder with program *ScrptCad.exe* (when script is not saved) or in folder with *.vic* script, which is executed.

**Note:** Only 1st level for including is supported. So included (library) scripts cannot use @*include* command again.

# Printing log and how to print using scale

**Printing log LastPrint.log**
When you print drawing at a printer, program *ScriptCad* make text file with faithful information about print. This file is named *LastPrint.log* and it is created in folder with program *ScrptCad.exe.* This file is created again and again for each print (so older information about printing are not saved). There is example of the file *LastPrint.log* and explanation of its content:

| | | |
|---|---|---|
| ScriptName: | C:\MyFolder\demo.vic | ... *script name* |
| Date: | 10/10/2006 16:06:34 | ... *date* |
| Printer: | Samsung ML-1510_700 Series | ... *printer name* |
| PixelsPerInch: | 600 dpi x 600 dpi | ... *resolution (pixels per inch)* |
| PageWidth: | 4756 | ... *page width in pixels* |
| PageHeight: | 6810 | ... *page height in pixels* |
| PageRes: | 4756 x 6810 | ... *page width x height in pixels* |
| PageRes (cm): | 20.13 x 28.83 | ... *page width x height in centimetres* |
| PhysPageSize: | 4960 x 7014 | ... *page width x height with borders in pixels* |
| PhysPageSize (cm): | 21.00 x 29.69 | ... *page width x height with borders in centimetres* |
| Orientation: | Portrait | ... *orientation of print (Portrait or Landscape)* |

Program *ScriptCad* use whole printing area, which is defined by *PageRes* - drawing is stretched during printing. In section Program setting-up (ScrptCad.ini) you can read more about print settings (colour depth).

**Drawing and exact scale**
For example, we have some (constructive) drawing and we want to print it at exact scale 1:50. Because program *ScriptCad* use stretch for printing and because each printer have another resolution (300 dpi, 600 dpi, ...) and page size (A4, A3, ...) and size of borders (compare *PageRes* and *PhysPageSize*), user must calculate some specifications.

For example, we have drawing of house in a file *myhouse.vic.* Real house have (some) wall 8 metres and we want scale 1:50. So we want to have this length 8:50 = 0.16 metre = 16 cm at the printer. Our printer use A4 format and let height is smaller then width. Then printing area have width = 20.13 cm (width without borders - see *PageRes (cm)*). If you select *MaximumX = 20.13* and wall is ...

*Line 3, 5, 19, 5, Solid, 1, DefaultColor*

... then wall have 16 cm at the paper (because 19 - 3 = 16 - and it is length of wall).

**Change of printer**
If drawing is finihed, you cannot change *MaximumX* and *MaximumY* for printing. Then, you can calculate backward - from *MaximumX* and required scale you can calculate *PageRes.* But in this case, calculated *PageRes* is not standard (A4, A3, ...). Example: *MaximumX = 12* and we want width of drawing = 12 cm. Then, we must set (at printing) custom page size - width = 12 cm (**without borders**).