

# g2 Reference Manual

## 0.7x

Generated by Doxygen 1.3.7

Tue Jun 14 23:33:58 2005



# Contents

<b>1</b>	<b>g2 Main Page</b>	<b>1</b>
1.1	License Notice . . . . .	1
1.2	Introduction . . . . .	1
1.3	Getting Started . . . . .	2
1.4	Contact . . . . .	5
<b>2</b>	<b>g2 Module Index</b>	<b>7</b>
2.1	g2 Modules . . . . .	7
<b>3</b>	<b>g2 Page Index</b>	<b>9</b>
3.1	g2 Related Pages . . . . .	9
<b>4</b>	<b>g2 Module Documentation</b>	<b>11</b>
4.1	FIG . . . . .	11
4.2	color manipulations . . . . .	12
4.3	output control . . . . .	14
4.4	devices control . . . . .	16
4.5	graphical output . . . . .	18
4.6	virtual device related functions . . . . .	25
4.7	GD . . . . .	26
4.8	PostScript . . . . .	27
4.9	MS Windows . . . . .	30
4.10	X11 . . . . .	31
4.11	g2 User Interface . . . . .	32
4.12	g2 Physical devices . . . . .	33
<b>5</b>	<b>g2 Page Documentation</b>	<b>35</b>
5.1	PS paper sizes . . . . .	35



# Chapter 1

## g2 Main Page

### 1.1 License Notice

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA \*

Copyright (C) 1998-2004 Ljubomir Milanovic & Horst Wagner.

### 1.2 Introduction

#### 1.2.1 What is g2 ?

##### 1.2.1.1 Short version (if you are in hurry)

- 2D graphic library
- Simple to use
- Supports several types of output devices (currently X11, PostScript, devices supported by gd <http://www.boutell.com/gd/> (PNG, JPEG), FIG (<http://www.xfig.org>) and MS Windows windows)
- Concept allows easy implementation of new device types
- Virtual devices allow to send output simultaneously to several devices
- User definable coordinate system
- Written in ANSI-C
- Tested under Digital Unix, AIX, Linux, VMS and Windows NT
- Perl support
- Fortran interface

### 1.2.1.2 Long version

g2 is a simple to use graphics library for 2D graphical applications written in Ansi-C. This library provides a comprehensive set of functions for simultaneous generation of graphical output on different types of devices. Presently, following devices are currently supported by g2: X11, gd (PNG and JPEG), PostScript and FIG (xfig). One major feature of the g2\_library is the concept of virtual devices. An arbitrary number of physical devices (such as PostScript, or X11) can be grouped to create a so-called virtual device. Commands sent to such a virtual devices will automatically issued to all attached physical devices. This allows for example simultaneous output to a PNG file and a Postscript file. A virtual device in turn can be attached to another virtual device, allowing to construct trees of devices. Virtual devices can also be useful when using different user-coordinate systems. E.g. one X11 window showing an overview of a graphical output, and a second window showing a zoom of a more detailed area of the graphic. Drawing in both windows is performed by one single command to the virtual device.

```

                                /-----> PNG:   g2_attach(id_PNG,...
-----
g2_plot----> | Virtual device: id |-----> X11:   g2_attach(id_X11,...
-----
                                \-----> PS:    g2_attach(id_PS,...

```

If you don't need or like the concept of virtual devices, simply ignore it.

## 1.3 Getting Started

### 1.3.1 Preinstallation tasks:

PNG and JPEG support

g2 uses the gd library by Thomas Boutell to generate PNG files. This package is freeware (however not GPL) and can be downloaded at <http://www.boutell.com/gd/>. Linux users might prefer to install a pre-compiled gd rpm package which should be available at your local RedHat mirrorsite. NT users should install the gd source package in a subdirectory named "gd" which should be located in the same directory as the g2 subdirectory (but not in the g2 directory itself). Otherwise file locations for gd must be modified in the g2 project workspace. Unix and VMS users will have to build and install gd according to the instructions found in the gd distribution.

### 1.3.2 Installation

LINUX

1. Either install RPM packet with binaries, or compile as described in the UNIX section

UNIX

1. Extract package with `gzip -dc g2-xxxx.tar.gz | tar xvf -`
2. Run `'./configure'`
3. Optionally run `'make depend'`
4. Run `'make'`

5. Run 'make install' or copy libg2.a and **g2.h**(p. ??), **g2\_X11.h**(p. ??), **g2\_gd.h**(p. ??), and **g2\_PS.h**(p. ??) to the default locations for library and include files.
6. Optional: cd to demo directory and run 'make demo' to compile demo applications

## WINDOWS NT

1. Extract package using either the .tar.gz or the .zip distribution
2. MS Visual C++ users can build both library and demos with the supplied project file: g2.dsw (To obtain an icon and use menu functions you must also build the g2res project in g2.dsw)
3. users of gcc or other commandline based compilers with make support continue as in Unix example
4. It is also possible to compile g2 on winNT/95 using the free cygwin32 library and a X-windows library for windows. Theoretically it should be possible to support both X-windows and native NT/95 windows at the same time.

## PERL (old instructions)

1. Change to directory g2\_perl
2. Perform following steps
  - (a) perl Makefile.PL
  - (b) make
  - (c) make test
  - (d) make install
3. See the **Perl interface**(p. 5) section for more information
4. swig is also supported, more details are coming ...

## VMS

1. Try to extract either the tar.gz or the zip distribution (whatever is easier for you)
2. type mms to compile library (descrip.mms file is supplied)
3. run mms in demo directory to compile demo applications

### 1.3.3 A simple example

The following example is a minimal application. It draws a rectangle in a postscript file.

```
#include <g2.h>
#include <g2_PS.h>

main()
{
    int id;
    id = g2_open_PS("rect.ps", g2_A4, g2_PS_land);
    g2_rectangle(id, 20, 20, 150, 150);
    g2_close(id);
}
```

- Always include `<g2.h(p.??)>`. Additionally include header files for all types of devices you want to use.
- Open devices using `g2_open_XY` functions. The open function returns a device id of type `int`, which you need to refer to the device.
- Call `g2_close()(p.??)` to close device.
- Consider turning off auto flush (`g2_set_auto_flush()(p.??)`) for improved performance.

You want to draw a PNG file instead of a PostScript file ?

replace the PS header file with

```
#include <g2_gd.h>
```

and replace the `g2_open_PS` function call with

```
id = g2_open_gd("rect.png", 300, 200, g2_gd_png);
```

You want to draw to a PNG file and a PostScript file with one plot command ?

Here we use the concept of virtual devices. Open a PNG and PostScript device, then open a virtual device and attach both the PNG and PostScript device to the virtual device. Plot commands to the virtual device will be issued to both PNG and PostScript device. You can attach and detach further devices at any time.

```
#include <g2.h>
#include <g2_PS.h>
#include <g2_gd.h>

main()
{
    int id_PS,id_PNG,id;

    id_PS = g2_open_PS("rect.ps", g2_A4, g2_PS_land);
    id_PNG = g2_open_gd("rect.png", 300, 200, g2_gd_png);
    id     = g2_open_vd();

    g2_attach(id, id_PS);
    g2_attach(id, id_PNG);

    g2_rectangle(id, 20, 20, 150, 150);
    g2_circle(id, 50, 60, 100);

    g2_close(id);
}
```

Note: closing a virtual device automatically closes all attached devices.

### 1.3.3.1 More examples

More examples showing the usage of different user coordinate systems, multiple virtual devices, etc. can be found in the distribution (demo directory).



### 1.3.4 Fortran interface

The Fortran interface for g2 is currently tested for Linux and Digital Unix/OSF. Function names for Fortran are the same as in C, however following differences exist:

all variables including device IDs are of type REAL void functions are implemented as subroutines and must be called with CALL constants defined by #define in C (e.g. g2\_A4) do not work. Get corresponding values from the appropriate header files.

A short Fortran example:

```
program demo
real d,color
d=g2_open_PS('demo.f.ps', 4.0, 1.0)
call g2_plot(d, 50.0, 50.0)
call g2_string(d, 25.0, 75.0, 'TEST ')
color=g2_ink(d, 1.0, 0.0, 0.0)
write (6,*) color
call g2_pen(d, color)
call g2_circle(d, 20.0, 20.0, 10.0)
call g2_flush(d)
call g2_close(d)
stop
end
```

### 1.3.5 Perl interface (old info)

The perl interface for g2 is currently tested for Linux and Digital Unix/OSF. Function names in perl are the same as in C, however the device itself is implemented object orientated, i.e. the device argument is omitted in all functions. E.g., following simple perl script:

```
use G2;

$d = newX11 G2::Device(100,100);
$d->circle(10, 10, 20);
$d->string(20, 40, "Hello World");

print "\nDone.\n[Enter]\n";
getc(STDIN);

$d->close()
```

The creator functions are newX11, newGIF, newPS, etc. and accept the same arguments as the open functions in the C version. See the perl documentation (perldoc G2) for more details and the test.pl script for a more extensive example.

## 1.4 Contact

You can contact the authors and contributors by e-mail (/ is @ and - is .):

- Ljubomir Milanovic: ljubo/users-sourceforge-net
- Horst Wagner: wagner/users-sourceforge-net
- Tijs Michels (spline implementation): tijs/vimec-nl

or visit g2 home page on: <http://g2.sourceforge.net/>



# Chapter 2

## g2 Module Index

### 2.1 g2 Modules

Here is a list of all modules:

g2 User Interface . . . . .	32
color manipulations . . . . .	12
output control . . . . .	14
devices control . . . . .	16
graphical output . . . . .	18
virtual device related functions . . . . .	25
g2 Physical devices . . . . .	33
FIG . . . . .	11
GD . . . . .	26
PostScript . . . . .	27
MS Windows . . . . .	30
X11 . . . . .	31



# Chapter 3

## g2 Page Index

### 3.1 g2 Related Pages

Here is a list of all related documentation pages:

PS paper sizes . . . . .	35
--------------------------	----



# Chapter 4

## g2 Module Documentation

### 4.1 FIG

#### Functions

- G2L int **g2\_open\_FIG** (const char \*file\_name)

#### 4.1.1 Detailed Description

FIG devices generate output in the FIG 3.2 format. For more details about FIG format and xfig application please visit <http://www.xfig.org> .

#### Note:

FIG is a vector-oriented (as oposed to pixel-oriented) format. Therefore **g2\_image**(p. 21) function and splines are not optimally supported.

#### 4.1.2 Function Documentation

##### 4.1.2.1 G2L int **g2\_open\_FIG** (const char \* *file\_name*)

Create a FIG device. g2 uses A4 paper size (landscape orientation) as default.

#### Parameters:

*file\_name* fig file name

#### Returns:

physical device id

## 4.2 color manipulations

### Functions

- void **g2\_pen** (int dev, int color)
- void **g2\_set\_background** (int dev, int color)
- int **g2\_ink** (int pd\_dev, double red, double green, double blue)
- void **g2\_reset\_palette** (int dev)
- void **g2\_clear\_palette** (int dev)
- void **g2\_allocate\_basic\_colors** (int dev)

#### 4.2.1 Detailed Description

The color concept used in the g2 library is inspired by Sir Clive Sinclair solution implemented in the ZX Spectrum computer. With the **g2\_pen()**(p.??) function it is possible to choose a pen created by the **g2\_ink()**(p.??) function. Note that g2\_ink function is only defined for physical devices. The predefined colors (see g2\_test demo program) have pens from 0 till 26 (inclusive).

Some basic colors are:

- 0 white
- 1 black
- 3 blue
- 7 green
- 19 red
- 25 yellow

#### 4.2.2 Function Documentation

##### 4.2.2.1 void g2\_allocate\_basic\_colors (int *dev*)

Allocate basic colors

**Parameters:**

*dev* device

##### 4.2.2.2 void g2\_clear\_palette (int *dev*)

Remove all inks.

**Parameters:**

*dev* device



#### 4.2.2.3 int g2\_ink (int *pd\_dev*, double *red*, double *green*, double *blue*)

Create an ink. To put ink into the pen use **g2\_pen()**(p. ??).

**Parameters:**

*pd\_dev* physical device

*red* red component (0-1) according to the RGB color model

*green* green component (0-1) according to the RGB color model

*blue* blue component (0-1) according to the RGB color model

**Returns:**

new pen, see **g2\_pen()**(p. ??)

#### 4.2.2.4 void g2\_pen (int *dev*, int *color*)

Set pen color for all following operations, see also **g2\_ink()**(p. ??).

**Parameters:**

*dev* device

*color* pen (either one of default pens 0-26, or a pen returned by **g2\_ink()**(p. ??) )

#### 4.2.2.5 void g2\_reset\_palette (int *dev*)

Clear color palette (remove all inks) and reallocate basic colors.

**Parameters:**

*dev* device

#### 4.2.2.6 void g2\_set\_background (int *dev*, int *color*)

Set the background color

**Parameters:**

*dev* device

*color* pen (either one of default pens 0-26, or a pen returned by **g2\_ink()**(p. ??) )

## 4.3 output control

### Functions

- void **g2\_flush** (int dev)
- void **g2\_save** (int dev)
- void **g2\_clear** (int dev)
- void **g2\_set\_font\_size** (int dev, double size)
- void **g2\_set\_line\_width** (int dev, double w)
- void **g2\_set\_dash** (int dev, int N, double \*dashes)
- void **g2\_set\_QP** (int dev, double d, enum QPshape shape)
- void **g2\_query\_pointer** (int dev, double \*x, double \*y, unsigned int \*button)
- void **g2\_get\_pd\_handles** (int pd, void \*handles[G2\_PD\_HANDLES\_SIZE])

### 4.3.1 Function Documentation

#### 4.3.1.1 void g2\_clear (int *dev*)

Clear device

**Parameters:**

*dev* device number

#### 4.3.1.2 void g2\_flush (int *dev*)

Flush output buffers.

**Parameters:**

*dev* device id

#### 4.3.1.3 void g2\_get\_pd\_handles (int *pd*, void \* *handles*[G2\_PD\_HANDLES\_SIZE])

Get pointers to physical device specific handles. This function should be used only if you are familiar with the g2 source code. For details see physical device source code (e.g. in src/X11/). Example usage can be found in demo/handles.c.

**Parameters:**

*pd* physical device

*handles* returns pointers to physical device low level handles

#### 4.3.1.4 void g2\_query\_pointer (int *dev*, double \* *x*, double \* *y*, unsigned int \* *button*)

Query pointer (e.g. mouse for X11) position and button state. See the demo program pointer.c for an example.

**Parameters:**

*dev* device  
*x* returns pointer x coordinate  
*y* returns pointer y coordinate  
*button* returns button state

**4.3.1.5 void g2\_save (int *dev*)**

Save output

**Parameters:**

*dev* device id

**4.3.1.6 void g2\_set\_dash (int *dev*, int *N*, double \* *dashes*)**

Set line dash. Set *N* to 0 and *dashes* to NULL to restore solid line.

**Parameters:**

*dev* device  
*N* number of dash components (0 for solid line)  
*dashes* vector of dash lengths (black, white, black, ...)

**4.3.1.7 void g2\_set\_font\_size (int *dev*, double *size*)**

Set font size

**Parameters:**

*dev* device  
*size* new font size

**4.3.1.8 void g2\_set\_line\_width (int *dev*, double *w*)**

Set line width.

**Parameters:**

*dev* device  
*w* new line width

**4.3.1.9 void g2\_set\_QP (int *dev*, double *d*, enum QPshape *shape*)**

Set QuasiPixel size and shape.

**Parameters:**

*dev* device  
*d* size  
*shape* shape (rectangle or circle, see ::QPshape )

## 4.4 devices control

### Functions

- void **g2\_close** (int dev)
- void **g2\_set\_auto\_flush** (int dev, int on\_off)
- void **g2\_set\_coordinate\_system** (int dev, double x\_origin, double y\_origin, double x\_mul, double y\_mul)
- int **g2\_ld** (void)
- void **g2\_set\_ld** (int dev)

### 4.4.1 Function Documentation

#### 4.4.1.1 void g2\_close (int dev)

Close and delete a device.

**Parameters:**

*dev* device

#### 4.4.1.2 int g2\_ld (void)

Get the last accessed device. G2LD macro is defined as the g2\_ld function.

```
g2_open_X11(100, 100);
g2_plot(G2LD, 50, 50);
```

#### 4.4.1.3 void g2\_set\_auto\_flush (int dev, int on\_off)

Set auto flush mode for device *dev*. Auto flush mode means that after each graphical operation g2 library automatically calls flush function to ensure that output is really displayed. However, frequent flushing decreases performance. Alternative is to flush output when needed by calling g2\_flush function.

**Parameters:**

*dev* device

*on\_off* 1-on 0-off

#### 4.4.1.4 void g2\_set\_coordinate\_system (int dev, double x\_origin, double y\_origin, double x\_mul, double y\_mul)

Set the user coordinate system.

**Parameters:**

*dev* device

*x\_origin* x coordinate of the new origin (expressed in the default coordinate system)

*y\_origin* y coordinate of the new origin (expressed in the default coordinate system)

*x\_mul* x scaling factor

*y\_mul* y scaling factor

#### 4.4.1.5 void g2\_set\_ld (int *dev*)

Set the last accessed device. See also **g2\_ld()**(p.??) function.

**Parameters:**

*dev* device

## 4.5 graphical output

### Functions

- void **g2\_move** (int dev, double x, double y)
- void **g2\_move\_r** (int dev, double dx, double dy)
- void **g2\_plot** (int dev, double x, double y)
- void **g2\_plot\_r** (int dev, double rx, double ry)
- void **g2\_line** (int dev, double x1, double y1, double x2, double y2)
- void **g2\_line\_r** (int dev, double dx, double dy)
- void **g2\_line\_to** (int dev, double x, double y)
- void **g2\_poly\_line** (int dev, int N\_pt, double \*points)
- void **g2\_triangle** (int dev, double x1, double y1, double x2, double y2, double x3, double y3)
- void **g2\_filled\_triangle** (int dev, double x1, double y1, double x2, double y2, double x3, double y3)
- void **g2\_rectangle** (int dev, double x1, double y1, double x2, double y2)
- void **g2\_filled\_rectangle** (int dev, double x1, double y1, double x2, double y2)
- void **g2\_polygon** (int dev, int N\_pt, double \*points)
- void **g2\_filled\_polygon** (int dev, int N\_pt, double \*points)
- void **g2\_ellipse** (int dev, double x, double y, double r1, double r2)
- void **g2\_filled\_ellipse** (int dev, double x, double y, double r1, double r2)
- void **g2\_circle** (int dev, double x, double y, double r)
- void **g2\_filled\_circle** (int dev, double x, double y, double r)
- void **g2\_arc** (int dev, double x, double y, double r1, double r2, double a1, double a2)
- void **g2\_filled\_arc** (int dev, double x, double y, double r1, double r2, double a1, double a2)
- void **g2\_string** (int dev, double x, double y, const char \*text)
- void **g2\_image** (int dev, double x, double y, int x\_size, int y\_size, int \*pens)
- void **g2\_plot\_QP** (int dev, double x, double y)

#### 4.5.1 Function Documentation

**4.5.1.1 void g2\_arc** (int *dev*, double *x*, double *y*, double *r1*, double *r2*, double *a1*, double *a2*)

Draw an arc.

##### Parameters:

*dev* device

*x* x coordinate of the center

*y* y coordinate of the center

*r1* x radius

*r2* y radius

*a1* starting angle (in deg. 0-360)

*a2* ending angle (in deg. 0-360)

**4.5.1.2 void g2\_circle (int *dev*, double *x*, double *y*, double *r*)**

Draw a circle.

**Parameters:**

*dev* device  
*x* x coordinate of the center  
*y* y coordinate of the center  
*r* radius

**4.5.1.3 void g2\_ellipse (int *dev*, double *x*, double *y*, double *r1*, double *r2*)**

Draw an ellipse.

**Parameters:**

*dev* device  
*x* x coordinate of the center  
*y* y coordinate of the center  
*r1* x radius  
*r2* y radius

**4.5.1.4 void g2\_filled\_arc (int *dev*, double *x*, double *y*, double *r1*, double *r2*, double *a1*, double *a2*)**

Draw a filled arc.

**Parameters:**

*dev* device  
*x* x coordinate of the center  
*y* y coordinate of the center  
*r1* x radius  
*r2* y radius  
*a1* starting angle (in deg. 0-360)  
*a2* ending angle (in deg. 0-360)

**4.5.1.5 void g2\_filled\_circle (int *dev*, double *x*, double *y*, double *r*)**

Draw a filled circle.

**Parameters:**

*dev* device  
*x* x coordinate of the center  
*y* y coordinate of the center  
*r* radius

**4.5.1.6 void g2\_filled\_ellipse (int *dev*, double *x*, double *y*, double *r1*, double *r2*)**

Draw a filled ellipse.

**Parameters:**

*dev* device  
*x* x coordinate of the center  
*y* y coordinate of the center  
*r1* x radius  
*r2* y radius

**4.5.1.7 void g2\_filled\_polygon (int *dev*, int *N\_pt*, double \* *points*)**

Draw a filled polygon.

**Parameters:**

*dev* device  
*N\_pt* number of points (Note: It is not size of *points* vector!)  
*points* vector of coordinates: x1, y1, x2, y2, ...

**4.5.1.8 void g2\_filled\_rectangle (int *dev*, double *x1*, double *y1*, double *x2*, double *y2*)**

Draw a filled rectangle specified by the two opposite corner points.

**Parameters:**

*dev* device  
*x1* x coordinate of the 1st corner  
*y1* y coordinate of the 1st corner  
*x2* x coordinate of the 3rd corner  
*y2* y coordinate of the 3rd corner

**4.5.1.9 void g2\_filled\_triangle (int *dev*, double *x1*, double *y1*, double *x2*, double *y2*, double *x3*, double *y3*)**

Draw a filled triangle specified by the 3 corner points.

**Parameters:**

*dev* device  
*x1* x coordinate of the 1st corner  
*y1* y coordinate of the 1st corner  
*x2* x coordinate of the 2nd corner  
*y2* y coordinate of the 2nd corner  
*x3* x coordinate of the 3rd corner  
*y3* y coordinate of the 3rd corner



**4.5.1.10** void `g2_image` (int *dev*, double *x*, double *y*, int *x\_size*, int *y\_size*, int \**pens*)

Draw a pen image

**Parameters:**

*dev* device

*x* x coordinate

*y* y coordinate

*x\_size* x size

*y\_size* y size

*pens* vector of *x\_size*\**y\_size* pens: p11, p21, ... pxy, ...

**4.5.1.11** void `g2_line` (int *dev*, double *x1*, double *y1*, double *x2*, double *y2*)

Draw a line from *x1*, *y1* to *x2*, *y2*.

**Parameters:**

*dev* device

*x1* see above

*y1* see above

*x2* see above

*y2* see above

**4.5.1.12** void `g2_line_r` (int *dev*, double *dx*, double *dy*)

Draw line relative to the graphic cursor.

**Parameters:**

*dev* device

*dx* relative x coordinate

*dy* relative y coordinate

**4.5.1.13** void `g2_line_to` (int *dev*, double *x*, double *y*)

Draw line from graphic cursor to the point *x*, *y*

**Parameters:**

*dev* device

*x* x coordinate

*y* y coordinate

**4.5.1.14 void g2\_move (int *dev*, double *x*, double *y*)**

Move graphic cursor.

**Parameters:**

*dev* device  
*x* x coordinate  
*y* y coordinate

**4.5.1.15 void g2\_move\_r (int *dev*, double *dx*, double *dy*)**

Move graphic cursor relative to the currner graphical cursor position.

**Parameters:**

*dev* device  
*dx* x coordinate increment  
*dy* y coordinate increment

**4.5.1.16 void g2\_plot (int *dev*, double *x*, double *y*)**

Plot a point

**Parameters:**

*dev* device  
*x* x coordinate  
*y* y coordinate

**4.5.1.17 void g2\_plot\_QP (int *dev*, double *x*, double *y*)**

Quasi Pixel fake. Quasi pixel is introduced to make easier plotting of cellular automata and related pictures. QP is simple a big pixel as specified by **g2\_set\_QP()**(p.??). Coordinates are skaled accordingly, so no recalculation is needed on client side.

**Parameters:**

*dev* device  
*x* x coordinate  
*y* y coordinate

**4.5.1.18 void g2\_plot\_r (int *dev*, double *rx*, double *ry*)**

Plot a point relative to graphical cursor.

**Parameters:**

*dev* device  
*rx* relative x coordinate  
*ry* relative y coordinate

**4.5.1.19 void g2\_poly\_line (int *dev*, int *N\_pt*, double \* *points*)**

Draw a poly line.

**Parameters:**

*dev* device

*N\_pt* number of points (Note: It is not size of *points* vector!)

*points* vector of coordinates: x1, y1, x2, y2, ...

**4.5.1.20 void g2\_polygon (int *dev*, int *N\_pt*, double \* *points*)**

Draw a polygon.

**Parameters:**

*dev* device

*N\_pt* number of points (Note: It is not size of *points* vector!)

*points* vector of coordinates: x1, y1, x2, y2, ...

**4.5.1.21 void g2\_rectangle (int *dev*, double *x1*, double *y1*, double *x2*, double *y2*)**

Draw a rectangle specified by the two opposite corner points.

**Parameters:**

*dev* device

*x1* x coordinate of the 1st corner

*y1* y coordinate of the 1st corner

*x2* x coordinate of the 3rd corner

*y2* y coordinate of the 3rd corner

**4.5.1.22 void g2\_string (int *dev*, double *x*, double *y*, const char \* *text*)**

Draw string, see also `g2_set_font_size()(p.??)`.

**Parameters:**

*dev* device

*x* x coordinate

*y* y coordinate

*text* null terminated string

**4.5.1.23**   `void g2_triangle (int dev, double x1, double y1, double x2, double y2,  
double x3, double y3)`

Draw a triangle described by 3 corner points.

**Parameters:**

*dev* device

*x1* x coordinate of the 1st corner

*y1* y coordinate of the 1st corner

*x2* x coordinate of the 2nd corner

*y2* y coordinate of the 2nd corner

*x3* x coordinate of the 3rd corner

*y3* y coordinate of the 3rd corner

## 4.6 virtual device related functions

### Functions

- `int g2_open_vd (void)`
- `void g2_attach (int vd_dev, int dev)`
- `void g2_detach (int vd_dev, int dev)`

#### 4.6.1 Detailed Description

Virtual device is a method to redirect g2 output to multiple devices. Here is an example:

```
int d1 = g2_open_X11(100, 100);    create first X11 window
int d2 = g2_open_X11(100, 100);    create 2nd X11 window

int vd = g2_open_vd();             open a new virtual device

g2_attach(vd, d1);                 attach d1 (1st window) to virtual device
g2_attach(vd, d2);                 attach d2 (2nd window) to virtual device

g2_plot(d1, 11, 11);               output to the 1st X11 window
g2_plot(d2, 12, 12);               output to the 2nd X11 window
g2_plot(vd, 13, 13);               output to both X11 windows
```

#### 4.6.2 Function Documentation

##### 4.6.2.1 void g2\_attach (int *vd\_dev*, int *dev*)

Attach a device to virtual device *vd\_dev*.

###### Parameters:

*vd\_dev* virtual device (create virtual device by calling `g2_open_vd()`(p. ??) ) 160  
*dev* device

##### 4.6.2.2 void g2\_detach (int *vd\_dev*, int *dev*)

Dettach a device from the virtual device *vd\_dev*.

###### Parameters:

*vd\_dev* virtual device 160  
*dev* device

##### 4.6.2.3 int g2\_open\_vd (void)

Create a new virtual device.

###### Returns:

virtual device ID

## 4.7 GD

### Enumerations

- enum `g2_gd_type` { `g2_gd_jpeg` = 0, `g2_gd_png` = 1 }

### Functions

- int `g2_open_gd` (const char \**filename*, int *width*, int *height*, enum `g2_gd_type` *gd\_type*)

#### 4.7.1 Enumeration Type Documentation

##### 4.7.1.1 enum `g2_gd_type`

g2 gd bitmap types

Enumeration values:

*g2\_gd\_jpeg* jpeg

*g2\_gd\_png* png

#### 4.7.2 Function Documentation

##### 4.7.2.1 int `g2_open_gd` (const char \* *filename*, int *width*, int *height*, enum `g2_gd_type` *gd\_type*)

Create a GD device.

Parameters:

*filename* output file name

*width* width

*height* height

*gd\_type* file type, see `g2_gd_type`(p. 26)

Returns:

physical device id

## 4.8 PostScript

### Enumerations

- enum **g2\_PS\_paper** {  
**g2\_A0**, **g2\_A1**, **g2\_A2**, **g2\_A3**,  
**g2\_A4**, **g2\_A5**, **g2\_A6**, **g2\_A7**,  
**g2\_A8**, **g2\_A9**, **g2\_B0**, **g2\_B1**,  
**g2\_B2**, **g2\_B3**, **g2\_B4**, **g2\_B5**,  
**g2\_B6**, **g2\_B7**, **g2\_B8**, **g2\_B9**,  
**g2\_B10**, **g2\_Comm\_10\_Envelope**, **g2\_C5\_Envelope**, **g2\_DL\_Envelope**,  
**g2\_Folio**, **g2\_Executive**, **g2\_Letter**, **g2\_Legal**,  
**g2\_Ledger**, **g2\_Tabloid** }
- enum **g2\_PS\_orientation** { **g2\_PS\_land**, **g2\_PS\_port** }

### Functions

- G2L int **g2\_open\_PS** (const char \*file\_name, enum **g2\_PS\_paper** paper, enum **g2\_PS\_orientation** orientation)
- G2L int **g2\_open\_EPSF** (const char \*file\_name)
- G2L int **g2\_open\_EPSF\_CLIP** (const char \*file\_name, long width, long height)

### 4.8.1 Enumeration Type Documentation

#### 4.8.1.1 enum **g2\_PS\_orientation**

**g2** paper orientation.

**Enumeration values:**

***g2\_PS\_land*** landscape  
***g2\_PS\_port*** portrait

#### 4.8.1.2 enum **g2\_PS\_paper**

**g2** paper type.

**Enumeration values:**

***g2\_A0*** A0 2384 x 3370  
***g2\_A1*** A1 1684 x 2384  
***g2\_A2*** A2 1191 x 1684  
***g2\_A3*** A3 842 x 1191  
***g2\_A4*** A4 595 x 842  
***g2\_A5*** A5 420 x 595  
***g2\_A6*** A6 297 x 420  
***g2\_A7*** A7 210 x 297

*g2\_A8* A8 148 x 210  
*g2\_A9* A9 105 x 148  
*g2\_B0* B0 2920 x 4127  
*g2\_B1* B1 2064 x 2920  
*g2\_B2* B2 1460 x 2064  
*g2\_B3* B3 1032 x 1460  
*g2\_B4* B4 729 x 1032  
*g2\_B5* B5 516 x 729  
*g2\_B6* B6 363 x 516  
*g2\_B7* B7 258 x 363  
*g2\_B8* B8 181 x 258  
*g2\_B9* B9 127 x 181  
*g2\_B10* B10 91 x 127  
*g2\_Comm\_10\_Envelope* Comm #10 Envelope 297 x 684  
*g2\_C5\_Envelope* C5 Envelope 461 x 648  
*g2\_DL\_Envelope* DL Envelope 312 x 624  
*g2\_Folio* Folio 595 x 935  
*g2\_Executive* Executive 522 x 756  
*g2\_Letter* Letter 612 x 792  
*g2\_Legal* Legal 612 x 1008  
*g2\_Ledger* Ledger 1224 x 792  
*g2\_Tabloid* Tabloid 792 x 1224

## 4.8.2 Function Documentation

### 4.8.2.1 G2L int *g2\_open\_EPSF* (const char \* *file\_name*)

Create an encapsulated PS device.

**Parameters:**

*file\_name* postscript file name

**Returns:**

physical device id

### 4.8.2.2 G2L int *g2\_open\_EPSF\_CLIP* (const char \* *file\_name*, long *width*, long *height*)

Create an encapsulated PS device with clipping.

**Parameters:**

*file\_name* postscript file name

*width* clipping region width

*height* clipping region height

**Returns:**

physical device id



**4.8.2.3** G2L int `g2_open_PS` (const char \* *file\_name*, enum `g2_PS_paper` *paper*, enum `g2_PS_orientation` *orientation*)

Create a PS device.

**Parameters:**

*file\_name* postscript file name

*paper* paper type, see `g2_PS_paper`(p. 27) and appendix Appendix

*orientation* paper orientation, see `g2_PS_orientation`(p. 27)

**Returns:**

physical device id

## 4.9 MS Windows

### Enumerations

- enum `g2_win32_type` { `g2_win32`, `g2_wmf32` }

### Functions

- int `g2_open_win32` (int *width*, int *height*, const char \**title*, int *type*)

### 4.9.1 Enumeration Type Documentation

#### 4.9.1.1 enum `g2_win32_type`

Window type

Enumeration values:

*g2\_win32* regular window

*g2\_wmf32* windows meta file

### 4.9.2 Function Documentation

#### 4.9.2.1 int `g2_open_win32` (int *width*, int *height*, const char \* *title*, int *type*)

Create a Windows device.

Parameters:

*width* window width

*height* window height

*title* window title

*type* window type, see `g2_win32_type`(p. 30)

Returns:

physical device id

## 4.10 X11

### Functions

- `int g2_open_X11` (int *width*, int *height*)
- `int g2_open_X11X` (int *width*, int *height*, int *x*, int *y*, char \**window\_name*, char \**icon\_name*, char \**icon\_data*, int *icon\_width*, int *icon\_height*)

#### 4.10.1 Function Documentation

##### 4.10.1.1 `int g2_open_X11` (int *width*, int *height*)

Open a simple X11 window (physical device device).

**Parameters:**

*width* window width

*height* window height

**Returns:**

physical device id

##### 4.10.1.2 `int g2_open_X11X` (int *width*, int *height*, int *x*, int *y*, char \**window\_name*, char \**icon\_name*, char \**icon\_data*, int *icon\_width*, int *icon\_height*)

Open a X11 window (physical device device). If *icon\_width* or *icon\_height* is smaller than 0, the *icon\_data* is interpreted as a file name.

**Parameters:**

*width* window width

*height* window height

*x* x position on screen

*y* y position on screen

*window\_name* hint for window manager

*icon\_name* hint for window manager

*icon\_data* icon bitmap (*icon\_width* \* *icon\_height* bits) or file name containing bitmap (if *icon\_width* <= 0 or *icon\_height* <= 0)

*icon\_width* icon width

*icon\_height* icon height

**Returns:**

physical device id

## 4.11 g2 User Interface

### Modules

- group color manipulations
- group output control
- group devices control
- group graphical output
- group virtual device related functions
- group g2 Physical devices

## 4.12 g2 Physical devices

### Modules

- group **FIG**
- group **GD**
- group **PostScript**
- group **MS Windows**
- group **X11**

#### 4.12.1 Detailed Description

g2 physical devices are drivers for different output formats.



## Chapter 5

# g2 Page Documentation

### 5.1 PS paper sizes

g2 Name	Name	Size(Pt)
g2_A0	A0	2384 x 3370
g2_A1	A1	1684 x 2384
g2_A2	A2	1191 x 1684
g2_A3	A3	842 x 1191
g2_A4	A4	595 x 842
g2_A5	A5	420 x 595
g2_A6	A6	297 x 420
g2_A7	A7	210 x 297
g2_A8	A8	148 x 210
g2_A9	A9	105 x 148
g2_B0	B0	2920 x 4127
g2_B1	B1	2064 x 2920
g2_B2	B2	1460 x 2064
g2_B3	B3	1032 x 1460
g2_B4	B4	729 x 1032
g2_B5	B5	516 x 729
g2_B6	B6	363 x 516
g2_B7	B7	258 x 363
g2_B8	B8	181 x 258
g2_B9	B9	127 x 181
g2_B10	B10	91 x 127
g2_Comm_10_Envelope	Comm #10 Envelope	297 x 684
g2_C5_Envelope	C5 Envelope	461 x 648
g2_DL_Envelope	DL Envelope	312 x 624
g2_Folio	Folio	595 x 935
g2_Executive	Executive	522 x 756
g2_Letter	Letter	612 x 792
g2_Legal	Legal	612 x 1008
g2_Ledger	Ledger	1224 x 792
g2_Tabloid	Tabloid	792 x 1224

# Index

- color
  - g2\_allocate\_basic\_colors, 12
  - g2\_clear\_palette, 12
  - g2\_ink, 12
  - g2\_pen, 13
  - g2\_reset\_palette, 13
  - g2\_set\_background, 13
- color manipulations, 12
- control
  - g2\_clear, 14
  - g2\_flush, 14
  - g2\_get\_pd\_handles, 14
  - g2\_query\_pointer, 14
  - g2\_save, 15
  - g2\_set\_dash, 15
  - g2\_set\_font\_size, 15
  - g2\_set\_line\_width, 15
  - g2\_set\_QP, 15
- device
  - g2\_close, 16
  - g2\_ld, 16
  - g2\_set\_auto\_flush, 16
  - g2\_set\_coordinate\_system, 16
  - g2\_set\_ld, 16
- devices control, 16
- FIG, 11
  - g2\_open\_FIG, 11
- g2 Physical devices, 33
- g2 User Interface, 32
- g2\_A0
  - PS, 27
- g2\_A1
  - PS, 27
- g2\_A2
  - PS, 27
- g2\_A3
  - PS, 27
- g2\_A4
  - PS, 27
- g2\_A5
  - PS, 27
- g2\_A6
  - PS, 27
- g2\_A7
  - PS, 27
- g2\_A8
  - PS, 27
- g2\_A9
  - PS, 28
- g2\_allocate\_basic\_colors
  - color, 12
- g2\_arc
  - graphic, 18
- g2\_attach
  - vd, 25
- g2\_B0
  - PS, 28
- g2\_B1
  - PS, 28
- g2\_B10
  - PS, 28
- g2\_B2
  - PS, 28
- g2\_B3
  - PS, 28
- g2\_B4
  - PS, 28
- g2\_B5
  - PS, 28
- g2\_B6
  - PS, 28
- g2\_B7
  - PS, 28
- g2\_B8
  - PS, 28
- g2\_B9
  - PS, 28
- g2\_C5\_Envelope
  - PS, 28
- g2\_circle
  - graphic, 18
- g2\_clear
  - control, 14
- g2\_clear\_palette
  - color, 12
- g2\_close
  - device, 16



- g2\_Comm\_10\_Envelope
  - PS, 28
- g2\_detach
  - vd, 25
- g2\_DL\_Envelope
  - PS, 28
- g2\_ellipse
  - graphic, 19
- g2\_Executive
  - PS, 28
- g2\_filled\_arc
  - graphic, 19
- g2\_filled\_circle
  - graphic, 19
- g2\_filled\_ellipse
  - graphic, 19
- g2\_filled\_polygon
  - graphic, 20
- g2\_filled\_rectangle
  - graphic, 20
- g2\_filled\_triangle
  - graphic, 20
- g2\_flush
  - control, 14
- g2\_Folio
  - PS, 28
- g2\_gd\_jpeg
  - GD, 26
- g2\_gd\_png
  - GD, 26
- g2\_gd\_type
  - GD, 26
- g2\_get\_pd\_handles
  - control, 14
- g2\_image
  - graphic, 20
- g2\_ink
  - color, 12
- g2\_ld
  - device, 16
- g2\_Ledger
  - PS, 28
- g2\_Legal
  - PS, 28
- g2\_Letter
  - PS, 28
- g2\_line
  - graphic, 21
- g2\_line\_r
  - graphic, 21
- g2\_line\_to
  - graphic, 21
- g2\_move
  - graphic, 21
- g2\_move\_r
  - graphic, 22
- g2\_open\_EPSF
  - PS, 28
- g2\_open\_EPSF\_CLIP
  - PS, 28
- g2\_open\_FIG
  - FIG, 11
- g2\_open\_gd
  - GD, 26
- g2\_open\_PS
  - PS, 28
- g2\_open\_vd
  - vd, 25
- g2\_open\_win32
  - win32, 30
- g2\_open\_X11
  - X11, 31
- g2\_open\_X11X
  - X11, 31
- g2\_pen
  - color, 13
- g2\_plot
  - graphic, 22
- g2\_plot\_QP
  - graphic, 22
- g2\_plot\_r
  - graphic, 22
- g2\_poly\_line
  - graphic, 22
- g2\_polygon
  - graphic, 23
- g2\_PS\_land
  - PS, 27
- g2\_PS\_orientation
  - PS, 27
- g2\_PS\_paper
  - PS, 27
- g2\_PS\_port
  - PS, 27
- g2\_query\_pointer
  - control, 14
- g2\_rectangle
  - graphic, 23
- g2\_reset\_palette
  - color, 13
- g2\_save
  - control, 15
- g2\_set\_auto\_flush
  - device, 16
- g2\_set\_background
  - color, 13
- g2\_set\_coordinate\_system
  - device, 16

- g2\_set\_dash
  - control, 15
- g2\_set\_font\_size
  - control, 15
- g2\_set\_ld
  - device, 16
- g2\_set\_line\_width
  - control, 15
- g2\_set\_QP
  - control, 15
- g2\_string
  - graphic, 23
- g2\_Tabloid
  - PS, 28
- g2\_triangle
  - graphic, 23
- g2\_win32
  - win32, 30
- g2\_win32\_type
  - win32, 30
- g2\_wmf32
  - win32, 30
- GD, 26
  - g2\_gd\_jpeg, 26
  - g2\_gd\_png, 26
  - g2\_gd\_type, 26
  - g2\_open\_gd, 26
- graphic
  - g2\_arc, 18
  - g2\_circle, 18
  - g2\_ellipse, 19
  - g2\_filled\_arc, 19
  - g2\_filled\_circle, 19
  - g2\_filled\_ellipse, 19
  - g2\_filled\_polygon, 20
  - g2\_filled\_rectangle, 20
  - g2\_filled\_triangle, 20
  - g2\_image, 20
  - g2\_line, 21
  - g2\_line\_r, 21
  - g2\_line\_to, 21
  - g2\_move, 21
  - g2\_move\_r, 22
  - g2\_plot, 22
  - g2\_plot\_QP, 22
  - g2\_plot\_r, 22
  - g2\_poly\_line, 22
  - g2\_polygon, 23
  - g2\_rectangle, 23
  - g2\_string, 23
  - g2\_triangle, 23
- graphical output, 18
- MS Windows, 30
- output control, 14
- PostScript, 27
- PS
  - g2\_A0, 27
  - g2\_A1, 27
  - g2\_A2, 27
  - g2\_A3, 27
  - g2\_A4, 27
  - g2\_A5, 27
  - g2\_A6, 27
  - g2\_A7, 27
  - g2\_A8, 27
  - g2\_A9, 28
  - g2\_B0, 28
  - g2\_B1, 28
  - g2\_B10, 28
  - g2\_B2, 28
  - g2\_B3, 28
  - g2\_B4, 28
  - g2\_B5, 28
  - g2\_B6, 28
  - g2\_B7, 28
  - g2\_B8, 28
  - g2\_B9, 28
  - g2\_C5\_Envelope, 28
  - g2\_Comm\_10\_Envelope, 28
  - g2\_DL\_Envelope, 28
  - g2\_Executive, 28
  - g2\_Folio, 28
  - g2\_Ledger, 28
  - g2\_Legal, 28
  - g2\_Letter, 28
  - g2\_open\_EPSF, 28
  - g2\_open\_EPSF\_CLIP, 28
  - g2\_open\_PS, 28
  - g2\_PS\_land, 27
  - g2\_PS\_orientation, 27
  - g2\_PS\_paper, 27
  - g2\_PS\_port, 27
  - g2\_Tabloid, 28
- vd
  - g2\_attach, 25
  - g2\_detach, 25
  - g2\_open\_vd, 25
- virtual device related functions, 25
- win32
  - g2\_open\_win32, 30
  - g2\_win32, 30
  - g2\_win32\_type, 30
  - g2\_wmf32, 30
- X11, 31

g2\_open\_X11, 31  
g2\_open\_X11X, 31