**GP142: The 142 Graphics Package  
User's Guide**

This document provides simple graphical output (drawing lines, rectangles, etc.), simple animation, and simple user input (mouse clicks and keyboard input).

Contents:

* [**The Simplest GP142 Program**](https://courses.cs.washington.edu/courses/cse142/GP142/GP142.html#simple_program)
* [**Drawing Model**](https://courses.cs.washington.edu/courses/cse142/GP142/GP142.html#draw)
* [**Flow of Control and User Interaction**](https://courses.cs.washington.edu/courses/cse142/GP142/GP142.html#control)
* [**Periodic Update Events and Animation**](https://courses.cs.washington.edu/courses/cse142/GP142/GP142.html#periodic)
* [**The Run Menu**](https://courses.cs.washington.edu/courses/cse142/GP142/GP142.html#menu)
* [**Drawing Limits, Undo, Clear**](https://courses.cs.washington.edu/courses/cse142/GP142/GP142.html#undo)
* [**List of Function**](https://courses.cs.washington.edu/courses/cse142/GP142/GP142.html#header)

**The Simplest GP142 Program**

To help you get started quickly, we will begin by looking at one of the simplest possible GP142 programs.  It doesn't do anything interesting at all; it merely opens a blank window and then waits for the user to tell it to close.  All GP142 programs share this same basic structure, and in fact this basic program can be used as a basis on which to build more elaborate programs, such as your homework.

A GP142 program must do at least the following things, in this order:

1. #include GP142.h
2. Open a GP142 graphics window by calling GP142\_open()
3. Wait for "events" by calling GP142\_await\_event().  Events are simply input from the user (mouse clicks or keystrokes) or from the system ("periodic" events, which the system generates automatically).  One special event that the user can send to your program is the GP142\_QUIT event, which tell the program that the user wishes to exit the program.  The user can send this event to the program by pulling down the **Run** menu from the GP142 graphics window and clicking **Quit**, or by hitting control-Q on the keyboard.
4. Close the GP142 graphics window when the user asks to quit by calling GP142\_close().

Here is a program that does the above, and nothing else:

/\*

\* CSE142

\*

\* This program illustrates one of the simplest possible GP142

\* programs that can be built.

\*/

#include "gp142.h"

#define FALSE 0

#define TRUE 1

int main(void)

{

int quit; /\* Track whether the user has asked to quit \*/

int event; /\* Holds GP142 events \*/

int mouse\_x, mouse\_y; /\* Not used in this program \*/

char key\_pressed; /\* Not used in this program \*/

/\* Open a blank GP142 graphics window.\*/

GP142\_open();

/\*

\* Main event loop:

\* ---- ----- -----

\* All GP142 programs need to have an "event loop", which is simply a

\* while loop that repeatedly gets events and then calls appropriate

\* functions in response. This program ignores all events except the

\* GP142\_QUIT event.

\*/

quit = FALSE;

while (!quit) {

/\* Get the next event \*/

event = GP142\_await\_event(&mouse\_x, &mouse\_y, &key\_pressed);

/\* Decide what kind of event we got \*/

switch (event) {

case GP142\_QUIT:

/\* The user asked to quit by selecting "Quit" from the "Run \*/

/\* menu or by hitting ctrl-Q. \*/

quit = TRUE;

break;

default:

/\* Ignore all other events, such as keystrokes, mouse clicks, \*/

/\* and periodic events. \*/

break;

}

} /\* end event loop \*/

/\* Close the graphics window and exit \*/

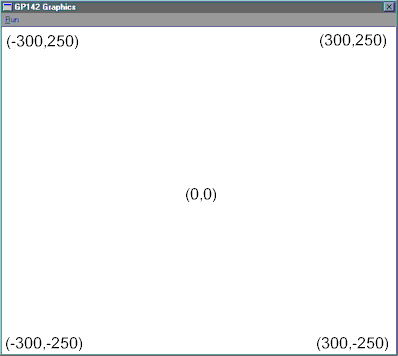
GP142\_close();

return 0;

}

**Drawing Model**

The unit of a computer graphics systems is the **pixel** (short for "picture element"), which is the smallest dot that can be drawn. Just to give you some sense of scale, there are about 72 pixels per inch on typical computer screens. GP142 does all of the drawing in a small window of approximately 600 by 500 pixels called the "graphics window", shown below. Positions in this window are described by the usual (x,y) coordinate system, with its origin approximately centered in the window.



A typical drawing command is the one to draw a line:

GP142\_lineXY(RED, -10, 50, 60, 50, 5);

which will draw a horizontal red line 5 pixels wide and approximately 70 pixels long, from the point (-10,50) to the point (60,50). To be somewhat more precise, you can imagine the line being drawn by a red marker pen with a 5 pixel wide tip. The pen starts with its tip centered at (-10,50) and is dragged to (60,50). Note that the drawn line will overhang the start and end point by 2 or 3 pixels in each direction due to the width of the pen tip. (As an example of the kinds of differences that you can expect to see between the Macintosh and PC versions, Windows draws with a round pen, hence the ends of thick lines are half-circles, while the Macintosh draws with a square pen. This typically will not matter to you, but may make visible differences in certain situations.)

As another example of the basic drawing routines, GP142\_rectangleXY will draw a rectangle, again using the model of dragging a marker pen of designated thickness around the perimeter of the designated rectangle. Other functions draw triangles and ovals (ellipses). For these closed figures (i.e., rectangles, triangles, and ovals) a line thickness of **0** specifies that the figure should be **filled**, rather than outlined, with the designated color. For example,

GP142\_rectangleXY(BLUE, -10, -5, 10, 5, 0)

will draw a 20 by 10 pixel solid blue rectangle in the center of the window.

Most of the GP142 drawing routines come two flavors. One uses X and Y coordinates to describe points in the graphics window, and the other uses the struct GP142\_point (which is #defined in gp142.h). If you haven't yet covered structs in lecture, don't worry. Just use the XY versions-- the functionality is the same. The type GP142\_point is defined as follows:

typedef struct {

int x;

int y;

} GP142\_point;

Consider the example above which drew a 20x10 pixel blue rectangle in the center of the screen. Using the XY function, this was done like this:

GP142\_rectangleXY(BLUE, -10, -5, 10, 5, 0)

The exact same shape can be drawn using the point functions like this:

GP142\_point lower\_left = {-10, 5}; /\* Declare and init the corners \*/

GP142\_point upper\_right = {10, 5};

...

GP142\_rectangleP(BLUE, lower\_left, upper\_right, 0);

Here is another example showing how you might use either the XY or the point functions to do the same thing. A call to:

GP142\_circleXY(BLUE, -10, -5, 10)

draws a blue, filled circle of radius 10 centered at -10,-5, which is slightly below and to the left of the center of the graphics window. You can declare and initialize a variable some\_point to do exactly the same:

GP142\_point some\_point; /\* We'll seperate declaration and \*/

/\* initialization this time. \*/

...

some\_point.x = -10; /\* Set up the center of the circle. \*/

some\_point.y = -5;

GP142\_circleP(BLUE, some\_point, 10)

.

You can put display text in the graphics window with GP142\_printfXY.

GP142\_printfXY(c, x, y, s, "format-string", expression1, expression2, ...)

is much like printf("format-string", expression1, expression2,...), except that the resulting string of characters is "printed" in the graphics window, starting at position (x,y) in color c and "point size" s. The point size of text is roughly the width in pixels of a typical character. Point sizes of nine to twelve are good sizes for labeling things in your figures. (Ordinary printf works, too, but its output goes to the console window, not the graphics window (this can be very useful for debugging). Like ordinary printf, GP142\_printfXY moves to a new line when you put a newline character ("\n") in the format string.

For example,

GP142\_printfXY(RED, 0, 50, 10, "%d %f %c", 10, 11.1, 'A');

prints 10 11.1 A in red at position (0,50) using a medium sized font (10 point).

**Flow of Control and User Interaction**

Just as you cannot use C's standard I/O facilities without having #include <stdio.h>, you cannot use GP142 without #include "GP142.h". The prototype for each of the functions is defined there. (Note: use "GP142.h", not <GP142.h>.)

Before you can use any of the GP142 functions, you must call GP142\_open() to allow the package to do various initializations, such as opening the graphics window. Similarly, before your program quits, it should call GP142\_close() to allow the package to do certain cleanup operations. In between, you can do any desired series of drawing commands. If your program is "noninteractive" and not animated, that's all there is to it.

To write interactive programs, i.e., ones where the program's behavior depends on user actions like clicking the mouse, or to do animation, there is one other important feature of the package -- the notion of "events". The function call GP142\_await\_event does just what its name suggests: it waits for some interesting "event" to occur before returning to your program. The four kinds of events it waits for are mouse clicks, keystrokes on the keyboard, "periodic update" events (explained [below](https://courses.cs.washington.edu/courses/cse142/GP142/GP142.html#periodic)), and the quit signal. GP142\_await\_event returns a code indicating which kind of event occurred. Also, if the event was a mouse click, it returns the *x* and *y* coordinates of the mouse pointer. Similarly, for a keyboard event, it returns the character corresponding to the key pressed.

So, in outline, a typical interactive program using GP142 will:

* Call GP142\_open().
* Call various of the GP142 drawing commands to set up the static part of the picture.
* Have a loop calling GP142\_await\_event repeatedly, until a quit event is signaled. Inside the loop the other three kinds of events are handled. (A switch statement might be a handy way to do this.)
* Call GP142\_close().

#include <stdio.h>

#include "GP142.h"

#define FALSE 0

#define TRUE 1

int main(void)

{

int quit; /\* has user signaled quit yet? \*/

int mouse\_x, mouse\_y; /\* mouse coordinates from GP142\_await\_event \*/

char key\_pressed; /\* character from GP142\_await\_event \*/

GP142\_open(); /\* Open and initialize the Graphics Window \*/

/\*

\* Whatever initialization you want goes here. For example, you

\* might want to draw your name, student number, and section on

\* the screen, together with other static parts of your picture.

\*/

/\*

\* The main event loop:

\* --- ---- ----- -----

\* Wait for the next user action (mouse click, keystroke, or a

\* periodic update event) and call the appropriate function to

\* handle it. Repeat this until user signals "quit".

\*/

quit = FALSE;

while (!quit) {

switch (GP142\_await\_event(&mouse\_x, &mouse\_y, &key\_pressed)) {

case GP142\_QUIT:

quit = TRUE; /\* set flag to terminate loop \*/

break;

case GP142\_MOUSE:

demo\_handle\_mouse(mouse\_x,mouse\_y);

break;

case GP142\_KBD:

demo\_handle\_kbd(key\_pressed);

break;

case GP142\_PERIODIC:

demo\_handle\_periodic();

break;

default:

break;

}

}

GP142\_close(); /\* Clean up and close graphics window \*/

return(0);

}

**Periodic Update Events and Animation**

Unlike keyboard, mouse, and quit events, the "periodic update" events are not signaled by any explicit user action. Rather, they are something like clock ticks. The system generates them automatically for you; you provide a way in your event loop for these events to get handled when they happen, and then you just trust that they happen.  If you wish to use GP142\_PERIODIC events in your program, you must get them started by calling GP142\_animate(ANI\_RUN).

Periodic events occur approximately 10-20 times per second, although not with clockwork regularity. They are intended to allow you to do simple animations: every time a periodic event happens, your program could update the display to show the next "frame" of an animation sequence. For example, our [demo](https://courses.cs.washington.edu/courses/cse142/GP142/GP142.html#demo) program (among other things) displays something like a clock, erasing and redrawing the clock hand a few degrees forward with every periodic update event.

You might wonder why we do not do animation by just explicitly writing a loop that draws frame after frame. For one thing, that would likely go faster than the eye could follow. More seriously, it would prevent other user interaction, like keyboard and mouse actions, from being noticed by the program, and prevent various useful "background" tasks from being performed by the computer.

**The Run Menu**

GP142 provides a menu called "Run", with the following items:

1. **Animate:** As an alternative to calling GP142\_animate( ANI\_RUN) from inside your program, the use can cause periodic update events to begin by choosigng "Animate" from the Run menu .
2. **Halt Animation:** Stop signalling periodic update events.
3. **Advance one frame:** Selecting this menu item signals just **one** periodic update. This is very useful for debugging your animations, since it lets you step through them frame by frame.
4. **Logging:** Selecting "Logging" causes all GP142 function calls to be logged to the console window. This can be helpful during debugging. Selecting it again turns off logging. Note that the graphics window runs quite slowly with logging turned on, so for animations you will want it turned off once you have finished debugging. Also, with logging on, the machine gets quite sluggish at recognizing mouse clicks, so it is hard (but not impossible) to pull down the menu to turn logging off again. The solution is to use the logging keyboard shortcut (-L) instead of the mouse to turn it off.
5. **Record GP142.SCR:** Record subsequent events in a file named GP142.SCR. You should avoid having a file with this name (unless you want it overwritten).
6. **Playback GP142.SCR:** Playback the sequence of events recorded in GP142.SCR.
7. **Playback One Event:** Playback just the next event in GP142.SCR.
8. **Quit:** signals the quit event (as does clicking the "close box" in the upper left corner of the graphics window). Note that these actions do not **force** your program to quit; they merely signal the event that requests it to quit. It is up to you to make your program respond appropriately.

GP142's version number is also displayed on the menu, in case you need to know.

**List of Functions**

/\* color pallet \*/

#define MAX\_COLORS 24

#define BLACK 0

#define WHITE 1

#define RED 2

#define GREEN 3

#define BLUE 4

#define YELLOW 5

#define MAGENTA 6

#define CYAN 7

#define PURPLE 8

#define NAVY\_BLUE 9

#define DUSTY\_PLUM 10

#define ICE\_BLUE 11

#define TURQUOISE 12

#define ORANGE 13

#define BROWN 14

#define PINK 15

#define CHALK 16

#define GOLD 17

#define PEACH 18

#define FOREST\_GREEN 19

#define SEA\_GREEN 20

#define OLIVE 21

#define MED\_GRAY 22

#define LT\_GRAY 23

void GP142\_open(void); /\* Initialize the GP142 library. \*/

/\* Invoke this at the beginning of main() \*/

void GP142\_close(void); /\* Close the library. \*/

/\* Invoke this at the end of main() \*/

void GP142\_pixelXY( /\* Draw a pixel on the graphics window. \*/

int, /\* color \*/

int, int); /\* x, y coordinates \*/

void GP142\_lineXY( /\* Draw a line segment on the graphics window. \*/

int, /\* color \*/

int, int, /\* x, y coordinates of starting point \*/

int, int, /\* x, y coordinates of ending point \*/

int); /\* line width \*/

void GP142\_rectangleXY( /\* Draw a rectangle on the graphics window. \*/

int, /\* color \*/

int, int, /\* x, y coordinates of any corner \*/

int, int, /\* x, y coordinates of opposite corner \*/

int); /\* line width \*/

void GP142\_triangleXY( /\* Draw a triangle on the graphics window. \*/

int, /\* color \*/

int, int, /\* x, y coordinates of first corner point \*/

int, int, /\* x, y coordinates of second corner point \*/

int, int, /\* x, y coordinates of third corner point \*/

int); /\* line width \*/

void GP142\_ovalXY( /\* Draw an oval on the graphics window. \*/

int, /\* color \*/

int, int, /\* x, y coordinates of any (bounding box) corner \*/

int, int, /\* x, y coordinates of opposite corner \*/

int); /\* line width \*/

void GP142\_circleXY( /\* Draw a filled circle on the graphics window. \*/

int, /\* color \*/

int, int, /\* x, y coordinates of center \*/

int); /\* radius \*/

/\*

\* GP142\_printfXY provides formated printout to the graphics window,

\* essentially like printf. Newlines do advance text to the next line.

\*

\* For example, the call

\* GP142\_printfXY(RED, 0, 20, 10, "%c%c%c %d", 'C', 'S', 'E', 142);

\* prints the string CSE 142 in red starting at position (0, 20) using

\* 10 point type.

\*/

void GP142\_printfXY(

int color, /\* text color \*/

int x, int y, /\* x, y coords of left end of text \*/

int point\_size, /\* text size \*/

char fmt[], /\* the printf format string \*/

type1 expr1,

type2 expr2, ... ); /\* list of values to print \*/

void GP142\_clear(void); /\* Clear graphics window. \*/

extern int FAR

GP142\_flush(void); /\* flushes the window by executing all drawing

requests that have been issued. The screen is flushed

when GP142\_await\_event is called, so you don't need

to use this function unless you want to synchronize

your display (perhaps when debugging) \*/

int GP142\_await\_event( /\* Fetch the next event. \*/

int \*, /\* pointer to mouse's x coordinate \*/

int \*, /\* pointer to mouse's y coordinate \*/

char \*); /\* pointer to character just entered \*/

extern int FAR

GP142\_gets( /\* Display a dialog box, and ask user \*/

/\* for a string \*/

const char prompt[], /\* Prompt string \*/

char result[]); /\* Result string \*/

extern void FAR /\* has an effect in Windows only \*/

GP142\_show\_text( /\* show or hide the text window \*/

int showit); /\* nonzero = show, zero = hide \*/

/\* Possible returns values: \*/

#define GP142\_MOUSE 1

#define GP142\_KBD 2

#define GP142\_PERIODIC 3

#define GP142\_QUIT 4

void GP142\_logging( /\* Switch logging on or off. \*/

int); /\* Possible values: \*/

#define LOG\_OFF 0 /\* == no logging \*/

#define LOG\_ON 1 /\* == logging \*/

void GP142\_animate( /\* Controls flow of program. \*/

int); /\* Possible values: \*/

#define ANI\_HALT 0 /\* == Stop animation \*/

#define ANI\_SINGLE\_STEP 1 /\* == Advance one frame \*/

#define ANI\_RUN 2 /\* == Start animation \*/

#define ANI\_QUIT 5 /\* == Quit program \*/

/\*

\* Point versions of many of the above functions

\*/

typedef struct {

int x;

int y;

} GP142\_point;

typedef struct {

int x, y;

char c;

} GP142\_event\_t;

extern int FAR

GP142\_await\_eventP( /\* Fetch the next event \*/

GP142\_event\_t \*event); /\* pointer to event record \*/

extern int FAR

GP142\_pixelP( /\* draw a pixel in the graphics window \*/

int color, /\* color \*/

GP142\_point p); /\* Point to draw \*/

extern int FAR

GP142\_lineP( /\* draw a line segment on the graphics window \*/

int color, /\* color \*/

GP142\_point p1, /\* starting point \*/

GP142\_point p2, /\* ending point \*/

int thickness); /\* line width \*/

extern int FAR

GP142\_rectangleP( /\* draw a rectangle on the graphics window \*/

int color, /\* color \*/

GP142\_point p1, /\* one corner point \*/

GP142\_point p2, /\* opposite corner point \*/

int thickness); /\* line width; 0 => fill \*/

extern int FAR

GP142\_triangleP( /\* draw a triangle on the graphics window \*/

int color, /\* color \*/

GP142\_point p1, /\* first corner point \*/

GP142\_point p2, /\* second corner point \*/

GP142\_point p3, /\* third corner point \*/

int thickness); /\* line width; 0 => fill \*/

extern int FAR

GP142\_ovalP( /\* draw an oval on the graphics window \*/

int color, /\* color \*/

GP142\_point p1, /\* one corner point \*/

GP142\_point p2, /\* opposite corner point \*/

int thickness); /\* line width; 0 => fill \*/

extern int FAR

GP142\_circleP( /\* draw a circle on the graphics window \*/

int color, /\* color \*/

GP142\_point p, /\* center point \*/

int radius); /\* radius \*/

extern int FAR

GP142\_printfP( /\* printf onto the graphics window \*/

int color, /\* text color \*/

GP142\_point p, /\* x, y coords of left end of text \*/

int size, /\* text size \*/

const char fmt[], /\* the printf format string \*/

... ); /\* list of values to print \*/