GTDriver

A graphic tablet and serial mouse driver for Amiga
Edition 1.0, for GTDriver V 1.0, rev 37.5
26 October 1994

by Roberto Attias & Marco Zandonadi

1 Overview

`GTDriver` is a program to control serial graphic tablets and serial mice with your Amiga. Version 1.0 of `GTDriver` supports the following tablets:

- Summagraphics MM
- Summagraphics Bitpadone
- CalComp 2000
- Cherry
- TekTronix 4957
- Wacom (see Chapter 9 [Known bugs or problems], page 30)

and the following mice:

- Microsoft mouse
- Mouse System mouse

If your tablet or mouse emulates one of the previous models you can use it with `GTDriver`. If not, contact the authors to have your device supported in the next release of the package (see Chapter 8 [How to have your tablet supported], page 28).

`GTDriver` has many parameters that can be configured using the program `GTDOptions`. Among the parameters, you can find the emulation type, the baud rate, the dpi tablet resolution and the dimensions of the clip region. These parameters must be properly set for both the driver and the tablet before you can start working (see Chapter 2 [Configuring hardware], page 3 and see Section 3.1 [Configuring software. GTDOptions], page 7).

`GTDriver` has two working modes: *driver mode* and *server mode*. When started, `GTDriver` is in driver mode. In this mode, you can move the screen pointer just like you do with the mouse. If you are using a stylus, pressing it down is equivalent to clicking the mouse select button. Pressing the button on the stylus body (if present) is comparable to clicking the mouse menu button. If you have a digitizing puck it may have more than two buttons. However, only three buttons are supported in driver mode. The third button is mapped to the third Amiga mouse button (that is supported by few programs).

Server mode is intended for programmers only. You can write your own programs that use `GTDriver` in server mode: in this way `GTDriver` will send complete tablet information to your program (instead of inserting it into the "input.device" chain) in a standard format, so you can take full advantage of tablet features in your software (see Chapter 6 [Programming GTDriver], page 20).

`GTDriver` supports user-defined tablet buttons (called *pseudo-buttons* or *p-buttons*). A p-button is a rectangular area on the tablet, to which an arbitrary key combination has been associated. Whenever you press the stylus on a pseudo button, its key combination is sent to the active program just like you were pressing it on the keyboard. As many programs use keyboard shortcuts to select tools and menu items, you can define sets of p-buttons even for programs not explicitly written to work with `GTDriver`. For example, you can use Deluxe Paint or DynaCadd with a graphic tablet without having tool bars on the screen, without selecting menu items and, most of all, without the software being aware of `GTDriver`!

> **Warning:** Before you can use your tablet with `GTDriver` you must configure both the tablet and the driver. Tablet configuration consists of setting parameters affecting the way your tablet behaves. Driver configuration consists of setting parameters that define how `GTDriver` interprets data coming from the tablet. Some driver parameters must be set in a compatible way with their counterparts on the tablet. I.e. you will have to set the baud rate at wich your tablet sends data to the driver. The driver will have to be set at the same rate, or nothing will work.

`GTDriver` requires O.S. V2.0 or greater. A fast CPU (68020/30/40) is not required but recommended.

2 Configuring hardware

Your graphic tablet has many internal parameters that must be configured before you can use it with `GTDriver`. Depending or your tablet model you can use one of the following methods:

- sending configuration strings to the tablet from the computer (see Section 3.1.2 [Init string], page 7);
- moving dip switches on your tablet;
- pressing the stylus on a menu/mask over the tablet;

Please refer to your tablet manual to understand which method to use. If your tablet is able to receive configuration strings, you should use the INIT field in the `GTDOptions` program to specify the string. This is the preferred way to set up your device. To use this method you will need to compose a string using the character sequences associated to every setting. These sequences may be different from model to model so you will need to look them up in your tablet manual.

If you set parameters by using dip switches you will need to do this only once.

If you use menu/mask there should be a way to make permanent the changes you made (i.e. a button labeled "save" on your tablet).

If you use an Init String, it will be sent to the tablet every time you start `GTDriver`.

In the following sections we show the typical tablet parameters explaining how to set them.

2.1 Emulation

Your tablet sends to the computer a stream of packets, each one containing information about the position and the button status of the pointing device (stylus or digitizing puck). The way these information are stored in the packets is different from tablet model to tablet model. Some tablets may have only one packet format, while others may allow the user to choose among many formats. We call *emulation* the format of the packet.

You must set one emulation among those supported by `GTDriver`. You will have to set the same emulation for both tablet and the driver (see Section 3.1.1 [Emulation (GTDOptions gadget)], page 7).

The available emulations are:

- Summagraphics MM
- Summagraphics Bitpadone
- CalComp 2000
- Cherry
- TekTronix 4957
- Wacom
- Microsoft mouse
- Mouse System mouse

2.2 Data format

Some emulations has a dual form: binary or ASCII. The binary format is more compact and allows faster manipulation by the software. For this reason all the emulations supported by `GTDriver` are binary, and you must set your tablet in binary format.

2.3 Baud rate

Being a serial device, a tablet may use different speeds to communicate with the computer. Usually the speed can range from 150 baud to 9600 baud. You can choose any of these, but a speed between 2400 and 9600 baud will give better results.

You will have to set the same rate for both tablet and the driver (see Section 3.1.6 [Baud], page 10).

2.4 Data bits

Number of bits sent to the serial port. Usually this parameter may be set to 7 or 8, but `GTDriver` expects eight bits, so you must set it to 8.

2.5 Operating mode

This parameter defines a set of conditions which trigger the transmission of coordinate data to the computer. Usually the following modes are available:

POINT A packet is transmitted only when one button of the pointing device is pressed while in proximity of the tablet surface.

STREAM Packets are transmitted continuosly as long as the pointing device is in proximity of the tablet surface. This mode is also referred to as RUN.

TRACK Packets are transmitted continuosly as long as one button of the pointing device is hold pressed while in proximity of the tablet surface.

As `GTDriver` uses the stylus as a mouse replacement, the tablet must always send data to the driver; so you must set it to STREAM (or RUN) mode.

2.6 Sampling rate

Speed at which the tablet generates packets. If this value is too high, the serial device will be overloaded and you may experiment some strange behaviour of the screen pointer. On the other hand, if this value is too low, the pointer movements will be jerky. It is recommended to choose a value between 50 and 100 events per second.

2.7 Resolution

The number of separately discernable points in a given distance. It is expressed in Dots Per Inch (DPI). Resolution can usually range from 100 to about 1000 DPI. You can choose any value for this parameter. Of course, the higher

the resolution the better the precision.

You will have to set the same rate for both tablet and the driver (see Section 3.1.9 [DPI], page 11).

2.8 Origin setting

This parameter defines where the origin of the coordinate system is located and how the axes are oriented. Usually, you should set the origin to the upper left corner of the tablet. You may avoid to modify this parameter of your tablet because `GTDOptions` allows to adjust the origin by software using the MirrorX, MirrorY and SwapXY parameters.

2.9 Coordinate mode

This parameter specifies if the coordinate pairs sent by the tablet to the serial port are absolute or are differences from the previous pairs (delta coordinates). `GTDriver` does not handle delta coordinates, so you must set this to "absolute".

3 Configuring software. GTDOptions

Once you have set the parameters for your tablet, you must set those for the driver using the GTDOptions prefs program. This program saves the configuration in a file called 'GTD.prefs'; this file can be found bin the "ENVARC:" and "ENV:" directori and is read by GTDriver every time it is started.

GTDOptions is operated via a font-adaptive graphic user interface. This means that the program window adjusts itself and its gadgets depending on the current screen font (if you use square pixel resolution for your Workbench and you never gave up the old topaz 8 font, try helvetica 13). If the screen font is too large, the program window is opened in topaz 8 font.

> **Tip**: GTDOptions lets you cycle through string and integer gadgets (via the Tab key) even when no one of them is selected. Try pressing the Tab key right after you started the program.

3.1 GTDOptions gadgets

In the following sections we describe the window gadget functions.

3.1.1 Emulation

Format used by the tablet to send data. To know wich emulations are supported by GTDriver see Chapter 1 [Overview], page 1.

> **Warning**: this parameter **must** match the tablet emulation setting.

3.1.2 Init string

This gadget should be used to set your tablet if it supports configuration via serial port. This is the preferred way to set up your device (see Chapter 2 [Configuring hardware], page 3).

Please understand that the Init String has nothing to do with GTDriver configuration: its purpose is to set tablet internal parameters. It could be said that the Init String field configures

the hardware (the tablet) whereas all other gadgets in the window configure the software (the driver). Of course hardware and software are to be set in a compatible way so that they can communicate correctly.

For example, if you have a tablet supporting Summagraphics MM and Calcomp 2000 emulations, you have to set it to use one emulation or the other. Depending on your tablet you can do it by adjusting dip switches, by moving the stylus over a menu (or a mask) on the tablet or by software (sending a configuration string to the serial port). The Init String field provides a simple way to use the third method (if your tablet supports it). If you use the Init String, you have to check the manual that came with your tablet and look up the commands you can send. If your tablet does not support the configuration string you have to use the other methods (again, check your tablet manual). If a command sequence includes non-printable characters they can be indicated by using the two digit hexadecimal ASCII code, preceeded by a "\x" string (i.e. the character whose ASCII code is 9 is indicated as "\x09"). You can also use "\n" for newline (ASCII code 10) and the "\\" sequence to specify the "\" character.

After some commands a short delay may be needed before your tablet can process the remaining part of the init string. You can produce this delay using the "\p" sequence.

3.1.2.1 Init string for Summagraphics tablets

For Summagraphics and compatible tablet models (Kurta, SummaSketch) we suggest to use the following Init string:

```
\x00\p@Rh
```

The following table shows the most important commands and their meaning:

```
\x00   tablet reset
@      set run mode
Q      set maximal data rate
R      set high data rate
S      set medium data rate
T      set minimal data rate
b      origint to upper left corner
c      origin to lower left corner
d      Resolution 100 DPI
e      Resolution 200 DPI
f      Resolution 10  DPmm
```

```
g      Resolution 400 DPI
h      Resolution 500 DPI
i      Resolution 20  DPmm
j      Resolution 1000 DPI
```

As you can see the suggested init string executes a tablet reset, sets a high data rate and a resolution of 500 DPI. Note that the reset command requires a delay, so a "\p" command has been introduced.

3.1.2.2 Init string for Wacom tablets

For Wacom and compatible tablet models we suggest to use the following Init string:

- Wacom A5
  RE\x0D\n\pSR\nAS1\nLA2\nIT4\nIC1\nSU0\nAS1\nPH0\n
- Wacom A5 pressure
  RE\x0D\n\pSR\nAS1\nLA2\nIT4\nIC1\nSU0\nAS1\nPH1\n
- Wacom A4+
  RE\x0D\n\pSR\nAS1\nLA2\nIT4\nIC0\nSU0\nAS1\nPH0\nLA3\n
- Wacom A4+ pressure
  RE\x0D\n\pSR\nAS1\nLA2\nIT4\nIC0\nSU0\nAS1\nPH1\nLA3\n
- Wacom A3
  RE\x0D\n\pSR\nAS1\nLA2\nIT4\nIC0\nSU0\nAS1\nPH0\nLA3\n
- Wacom A3 pressure
  RE\x0D\n\pSR\nAS1\nLA2\nIT4\nIC0\nSU0\nAS1\nPH1\nLA3\n
- Wacom A3 +
  RE\x0D\n\pSR\nAS1\nLA2\nIT4\nIC0\nSU0\nAS1\nPH0\nLA3\n
- Wacom A3 + pressure
  RE\x0D\n\pSR\nAS1\nLA2\nIT4\nIC0\nSU0\nAS1\nPH1\nLA3\n

3.1.3 Priority

Priority of the GTDriver process. A value of 5 should be appropriate in almost every situation. The maximum value allowed for this parameter is 19.

3.1.4 Pressure

This parameter has a meaning only for pressure sensitive tablets. It tells the driver the threshold above which a stylus pressure will be translated in a left mouse button event by GTDriver. If the stylus pressure value is inferior to the Pressure field value, GTDriver ignores it. The only pressure sensitive tablet supported by GTDriver 1.0 is Wacom.

3.1.5 Device

Under normal conditions your tablet will be connected to the standard Amiga serial port. This port is controlled by the "serial.device" , and its unit number is 0 (these two settings are the defauts). However, third party serial boards exist that are driven by different software devices and that may have different unit numbers, so you are allowed to specify the name of the device and the unit number in the corresponding fields of GTDOptions.

3.1.6 Baud

Baud rate at wich GTDriver communicates with the tablet. You can choose values between 150 and 9600.

Warning: this parameter **must** match the tablet baud rate setting.

3.1.7 Metric

Metric preferences (centimeters or inches) for the values expressed in fields Clip, XDim and YDim. If Metric is changed then Clip, XDim and YDim values are converted on the fly.

3.1.8 XDim and YDim

Dimensions of the tablet *active area*: the area in which a stylus move or pressure is translated to a serial message (the sensitive part of your tablet). Values are expressed in centimeters or inches depending on the status of the Metric gadget. The minimum value allowed for both fields is 2 inches.

Note: You may notice that after lowering XDim or YDim values, Clip value gets lowered too. This happens because GTDOptions makes sure that the clip region is always at least 2x2 inches.

3.1.9 DPI

Tablet DPI resolution. DPI is for Dots per Inch. The higher this value the better the precision.

Warning: this parameter **must** match the tablet emulation setting.

3.1.10 Clip

The *clip region* is a part of the active area that is used for screen pointer movements. The Clip field specifies the clip region as a rectangle centered inside the active area; the Clip value is the distance between a border of the active area and the corresponding border of the clip region.

I.e. if a value of 1 is specified, the metric setting is inches, the XDim and YDim values are both 12 inches, the clip region is a 10-inch-sided square.

The Clip value is expressed in centimeters or inches depending on the status of the Metric gadget.

Note well: since the minimum value for XDim and YDim is 2 inches, the maximum value for Clip is

$$\frac{(\min(\text{XDim, YDim}) - 2 \text{ inches})}{2}$$

This ensures that the clip region is always at least 2 inches.

3.1.11 Swap XY

The SwapXY flag exchanges the tablet X and Y axes. Mixing SwapXY, MirrorX and MirrorY allows you to rotate and use your tablet at any orientation.

3.1.12 Mirror X and Mirror Y

It may happen that mouse movements are reversed (e.g. you move the stylus right and the screen pointer moves left). This should be due to a wrong position of the tablet origin (see Chapter 2 [Configuring hardware], page 3). You may solve this problem by using the MirrorX and MirrorY gadgets. MirrorX mirrors the X-axis and MirrorY mirrors the Y-axis on the tablet.

3.2 GTDOptions menus

In the following sections we describe the menu items functions.

3.2.1 Open

Opens an ASL file requester for choosing a prefs file to load in GTDOptions.

Keyboard shortcut: Right-Amiga-o.

3.2.2 Save As

Opens an ASL file requester for saving the current preferences with a user-selected name.

Keyboard shortcut: Right-Amiga-a.

3.2.3 About

Gives some informations about the authors and about current program versions.

3.2.4 Quit

Exits GTDOptions.

Keyboard shortcuts: Right-Amiga-q or the ESC key.

3.2.5 Last Saved

Loads last saved prefs file, without opening a file requester.

Keyboard shortcut: Right-Amiga-l.

4 Starting and exiting GTDriver

GTDriver can be started from shell or Workbench.

To start GTDriver from Workbench (preferred way) you must double click on its icon. The driver will run using the parameters found in the file 'ENV:GTD.prefs'. You have to set these parameters by using the GTDOptions program. If this file is not found, GTDriver informs you about it and runs with the default parameters.

You can also start GTDriver from Workbench by double clicking on a p-button project icon. Such icon has to be associated to an ASCII file containing the pseudo-buttons definition (see Chapter 5 [Pseudo-buttons], page 16 to know how to use this feature).

If you start GTDriver from shell, you may override some of the settings in 'ENV:GTD.prefs' by using arguments on the command line. The shell template is the following:

```
EMUL=Emulation/K,Device/K,Unit/K/N,Baud/K/N,DPI/K/N,Metric/K,
Xdim/K,Ydim/K,MX=MirrorX/S,MY=MirrorY/S,SwapXY/S,CLIP/K,Init/K,
Pri/K/N,Pressure/K/N,ButDef/K,Verbose/S,Help/S
```

Most parameters are equivalent to the ones specified by using GTDOptions; the remaining parameters are:

ButDef	allows to specify a file name containing the description of a set of p-buttons; (see Chapter 5 [Pseudo-buttons], page 16)
Verbose	if this keyword is given the state of the parameters is printed in the shell;
Help	Shows a little explanation of the keywords and exits

GTDriver does not detach from the shell, so if you want to use the shell after starting it, you must use the run command.

There are many ways to kill GTDriver. If you started the program from shell you can press CTRL-C or send a break-signal to the program with the command

`break` *CLI process number*

(you can obtain the CLI process number by using the `status` command).

A second way you to end `GTDriver` is by starting it again. A requester will be opened asking you if you want to kill it or to not. The last way to kill the driver is intended for programmers and is explained in the programmer section.

5 Pseudo-buttons

Pseudo-buttons are rectangular regions defined on the tablet. They must be outside the clip region.

Each pseudo-button may have one or two keyboard sequences associated to it. When the stylus (or button 0 of the digitizing puck) is pressed over a pseudo-button, the first keyboard sequence is sent to the active program. When the button on the stylus (or button 1 of the digitizing puck) is pressed over the pseudo-button, the second keyboard sequence is sent. I.e.: if you created a pseudo button with keyboard sequence `LEFT AMIGA-O` (usually mapped to the "Project/Open" menu item) when you press the stylus on it, the active application opens a file requester just like if you pressed `LEFT AMIGA-O` on the keyboard.

You can edit your own sets of p-buttons to be used with paper masks over the tablet.

Every set of p-buttons is defined in an ASCII file. We suggest to use a filename with extension ".but". The file contains a command specifying the metric, followed by the clip region coordinates and one or more button definitions. If a ";" is found, the remaining part of the line is interpreted as a comment. The text can be typed in lower or upper case.

The metric-mode command must be the first command in the file. This command spcifies the metric unit used to interret the button and clip definitions. It has the following syntax:

```
Metric cm
```

or

```
Metric inches
```

The metric definition must be followed by the *clip* command. This command allows to define the clip region. When you move the pointing device inside this area the screen pointer moves accordingly. P-buttons will have to reside on the tablet outside of the clip area. When using `GTDriver` with a button file, the clip rectangle definition replaces the one defined with the `clip` gadget of `GTDOptions`. The syntax for this command is:

`Clip` *ULC_x ULC_y LRC_x LRC_y*

where (ULC_x, ULC_y) are the coordinates of the upper left corner, while (LRC_x, LRC_y) are the coordinates of the lower right corner. Coordinates can be positive, negative or null. A positive x coordinate means a distance from the side upon wich the y axis lies. A negative x coordinate means a distance from the opposite side of the tablet. The same holds for y coordinates. Null coordinates are interpreted in different ways for ULC and LRC. A null coordinate is interpreted as a positive coordinate for ULC, while it is interpreted as negative for LRC. Here there are some examples of metric and clip commands:

```
Metric cm
Clip 1 2 -3 -4
```

defines a clip region with left side 1 cm far from tablet left side, top side 2 cm far from tablet top side, right side 3 cm far from tablet right side and bottom side 4 cm far from tablet bottom side.

```
Metric cm
Clip 0 0 0 0
```

defines a clip region covering the whole active area.

Negative coordinates have been implemented for defining clip regions and p-buttons that adapt to the width and height of the tablet. In this way you can create masks and distribute them to people with different sized tablets. I.e. suppose you want to define a clip region leaving a half-inch border on the left top and bottom sides, and a two inch border on the right side to put a p-button bar in it. If your tablet is 12 * 12 inches you can insert in the button file the following commands:

```
Metric inches
Clip 0.5 0.5 10 11.5
```

Anyway if you give this file to someone having a different sized tablet, he will not be able to use it. The following method ensures the correct compatibility:

```
Metric inches
Clip 0.5 0.5 -2 -0.5
```

obviously the clip region will have different sizes in the two cases (depending on the size of the tablet used), but the border widths will be the same.

After the clip command you have to type one or more p-button definitions, each one with the following syntax:

`Button` *rectangle definition Selector* `[`*Qualifiers*`]` *key*
`[`*Selector* `[`*Qualifiers*`]` *key*`]`

where:

– *rectangle definition* are two coordinate pairs defining the button area. as for the CLIP command you can use positive or negative coordinates.

– *Selector* can be the string B0 or B1. B0 refers to the action of pressing the stylus over the p-button while B1 refers to pressing the button on the stylus body (if present).

– *Qualifiers* and *key* define a key combination to be sent to the active program when the corresponding action (B0 or B1 pressure) happens. *Qualifiers* is a sequence of one or more strings corresponding to special keys such as Ctrl, Alt, Shift, etc. Possible string values for *Qualifier* are

```
control      (CTRL key)
lalt         (LEFT-ALT key)
ralt         (RIGHT-ALT key)
lcommand     (LEFT-AMIGA key)
Rcommand     (RIGHT-AMIGA key)
lshift       (LEFT-SHIFT key)
rshift       (RIGHT-SHIFT key)
numericpad   (Numeric Pad key)
```

You can not use the `l/rshift` qualifier but with `raw` special command (see below).

– *key* is a key identifier. On a keyboard normal and special keys are present. Normal keys generate characters: among them there are alfanumeric keys, punctuation keys, and others. Special keys have particular functions: among them there are the editing keys (i.e. backspace, delete and cursor keys), function keys and others. To express a normal key in the *key* field you can type it or you can use its two digit hexadecimal ASCII code, preceeded by a `"\x"` string (i.e. the character whose ASCII code is 09 is indicated as `"\x09"`). You can also use `"\n"` for

newline (ASCII code 10) and the "\\" sequence to specify the "\" character. To express a special key just use one of the following special identifiers:

```
"BACKSPACE", "CURSOR_DOWN", "CURSOR_LEFT", "CURSOR_RIGHT",
"CURSOR_UP","DEL", "ESC", "F1", "F2", "F3", "F4", "F5", "F6",
"F7", "F8", "F9", "F10", "HELP","TAB".
```

If the program to build the button mask for gets keyboard inputs as raw keys (i.e. Dpaint 4.5) you may need to express some keys as raw data. For this purpose the special command `raw` has been provided. `Raw` is to be follwed by the raw code for the key (see the example below). You can use any qualifier together with a raw key definition.

To understand the BUTTON syntax easily take a look at the following examples:

```
Button -1 0 0 1 b0 LAlt LShift p
```

defines a 1 x 1 inches or cm (depending on the METRIC command) p-button located in the upper right corner of the tablet active area. When you press the stylus over it, the active program receives a `LEFT-ALT-LEFT-SHIFT-P` key combination.

```
Button -1 1 0 2 b0 Ctrl a b1 TAB
```

defines a 1 x 1 inches or cm (depending on the METRIC command) p-button located under the previous one. When you press the stylus over it, the active program receives a `CTRL-A` key combination, while if you press the button on the stylus a `TAB` key is sent.

```
Button -1 2 0 3  b0 LShift RAW \x39
```

defines a 1 x 1 inches or cm (depending on the METRIC command) p-button located under the previous one. When you press the stylus over it, the active program receives a key combination composed of the left shift key and the key whose raw code is 39. Such key corresponds to the ">" key on American keyboards, and ":" on Italian keyboards.

For further examples, take a look at the '`Dpaint.but`' file of the distribution.

6 Programming GTDriver

This section is intended for programmers only, so, if you are not a programmer, you may skip to the next section. GTDriver has a public message port, named GTDPubPort. By sending special messages to such port you can quit the driver, change the internal settings, and set it to "server mode" (see below).

To enable your C programs to send commands to this port you will need to include the file 'GTDriverMPC.h' containing constant definitions and structures used to communicate with the driver. 'GTDriverMPC.h' can be found in the 'Sources' directory of the distribution.

Every command sent to GTDriver is a structure TabletCommand:

```
struct TabletCommand {
        struct Message msg;
        UWORD  Command;
        APTR   Data;
    };
```

The Command field must be initialized with one of the MPC_XXX constants, that identify commands (and that are found in 'GTDriverMPC.h'). The Data field can be initialized with a command parameter (if the Command requires it). The msg field is a normal struct Message (see the example below, to understand how to initialize it). When the struct TabletCommand instance is initialized correctly, you can send it to the public port of GTDriver by using the PutMsg() system function. After you sent the message you have to wait for the reply to a message port you previously set up. When you receive the reply you must check the Command field of the message for the acknowledgement: if it's equal to COMMAND_OK your request has been satisfied, otherwise the field contains COMMAND_FAILED. In this case you can get some info on the reason of the failure by reading the Data field (it will be set to one of the CMDERR_ constants).

Here follows a fragment of code showing how to send commands:

```
:   :   :   :
/* The TabletCommand instance (Tcmd) is a global variable in this example
*/
struct TabletCommand Tcmd;
:   :   :   :
/* Somewhere you must create a message port to receive replies to. You
** also need to init your TabletCommand instance with the port address
```

```
*/
    Tcmd.msg.mn_Node.ln_Type = NT_MESSAGE;
    Tcmd.msg.mn_ReplyPort = myreplyport;
    Tcmd.msg.mn_Length = sizeof(struct TabletCommand);
:   :   :   :

/* Here follows a sample function you can use to send commands
*/
short SendCommand(ULONG comm, ULONG data)
{
    Tcmd.Command = comm;
    Tcmd.Data = (void *)data;

    /* Here we look for GTDriver message port; if it does not exist we return
    ** NO_PUBPORT. Note that you should always FindPort() before sending a
    ** message because the driver may have been quitted.
    */
    Forbid();
    port = FindPort(GTDPUBPORTNAME);
    if (! port) {
        Permit();
        return NO_PUBPORT;
}
    /* We send the message and wait for the reply
    */
    PutMsg(port,(struct Message *)&Tcmd);
    Permit();
    WaitPort(repport);
    GetMsg(repport);

    /* We return Tcmd.Command: if it contains COMMAND_OK then everything went
    ** fine, otherwise in Tcmd.Data you find a CMDERR_xxx code explaining what
    ** happened.
    */
    return (short )Tcmd.Command;
}
```

GTDriver works synchronously, so you must wait for the reply to a command before sending a new one. Here follows command explanations. They are preceeded by a short reference with three fields:

Command:	*the name of the command*
Data:	*explanation of the parameter for the command (if needed)*
Errrors:	*name of the constant that can be found in the Data field of the message after the WaitPort() if the command has failed (Command ==* COMMAND_FAIL*)*

You can find some examples on how to program GTDriver in the 'sources' directory of the distribution.

The allowed commands are

6.1 MPC_SERVERMODE command

```
Command: MPC_SERVERMODE
Data:    address of the client port
Error:   CMDERR_SERVER   (GTDriver is already a server)
```

This command can switch GTDriver to "server mode". When started, GTDriver is in "driver mode": this means that the driver sends its data in the input.device chain, controlling the screen pointer movements. With an MPC_SERVERMODE command, your program makes the driver send it information in the form of messages sent to a private message port of its. In this case your program is called *client*, while the driver is called *server*. While it is in "server mode" GTDriver does not control the screen pointer any more.

When sending the MPC_SERVERMODE command the Data field must be set to the address of the port at which you wait for serial info. Note that this port must be different from the reply port where you WaitPort() for acknowledgement after you send a command to the driver. The MPC_SERVERMODE command may fail if GTDriver is already in "server mode": in fact when in "server mode" the driver will refuse every command not coming from the client program exept MPC_GETPREFS. The format of the messages sent by GTDriver to your program is the following:

```
struct TabletMessage {
    struct Message msg;
    UWORD type;
    UWORD x,y;
    ULONG buttons;
    char  key;
    WORD  pressure;
};
```

where

- `type` can be TMTYPE_COORDS, indicating that the message is due to a stylus movement or button pressure, or TMTYPE_PBUTTON, if some pseudo-button has been used;
- `x,y` are the coordinates of the stylus expressed in tablet dots;
- `buttons` is a binary mask indicating the state of the buttons. Each bit expresses the status of a button (0 means released and 1 means pressed).
- `key` matches the *key* field of the definition of a pseudo-button if such p-button has been pressed (in this case *type* == TMTYPE_PBUTTON);
- `pressure` reports the pressure value of the stylus on the tablet in the Wacom emulation;

X and y are expressed in dots. Dots are used internally by the driver instead of centimeters or inches. If you need to convert dots in inches or centimeters use the following formulas:

```
x_inches = ((float)Tmsg->x)/DPI;    x_cm = x_inches * CONV_INCHES2CM;
y_inches = ((float)Tmsg->y)/DPI;    y_cm = y_inches * CONV_INCHES2CM;
```

where CONVINCHES2CM is a constant defined in the '`GTDriverMPC.h`' file. You can know about the internal `GTDriver` DPI setting with the command MPC_GETPREFS.

> **Warning:** `GTDriver` works sincronously. This means that it does not send a new TabletMessage to your program if the previous one has not been ReplyMsg()'ed. As the driver Wait() for your reply, data coming from the tablet are lost if you don't do it quickly. Your tablet event routine has to handle messages quite fastly, or you will loose data.

Note that by using server mode your program may know about more than three buttons (if they are present on your device) and it may also know about pressure (if you have a Wacom tablet). These information are not available in driver mode.

> **Warning:** your program `must` switch `GTDriver` back to driver mode before closing its message port. For performance reasons `GTDriver` does not search for your program's port before sending data. So, if your program closes it before resetting `GTDriver` to driver mode, a system crash is almost guaranteed.

6.2 The MPC_DRIVERMODE command

```
Command: MPC_DRIVERMODE
Data:    -
Error:   CMDERR_SERVER     (your program is not the client)
```

This command switches GTDriver back to driver mode. After getting it, GTDriver will start controlling the screen pointer movements and will stop sending data to the client program. The command will not work if the sending program is not the client.

6.3 The MPC_QUIT command

```
Command: MPC_QUIT
Data:    -
Error:   MPC_SERVER     (your program is not the client)
```

This command makes GTDriver exit. If GTDriver is in server mode, only the client can send this command. If GTDriver is in driver mode any program is allowed to send this command.

6.4 The MPC_GETPREFS command

```
Command: MPC_GETPREFS
Data:    pointer to an instance of a GTDPrefs structure
Error:   -
```

By sending this command any program (client or not) can know the internal settings of GTDriver. The Data field of the command must point to an instance of GTDPrefs structure. The definition of the structure can be found in the 'GTDriverMPC.h' file. After your program gets the reply of the command your GTDPrefs structure will have been filled with GTDriver internal settings.

6.5 The MPC_NEWPREFS command

```
Command: MPC_NEWPREFS
```

```
Data:   -
Error:  MPC_SERVER    (GTDriver is in server mode)
```

This command tells GTDriver that its preferences have been changed. It was implemented only to allow the needed interaction with GTDOptions and you should not need to use it. The command will fail if the driver is in "server mode" to avoid problems with the client program.

6.6 The MPC_TESTPREFS command

```
Command: MPC_TESTPREFS
Data:   -
Error:  MPC_SERVER    (GTDriver is in server mode)
```

This command is sent by GTDOptions when you press the Test gadget. It was implemented only to allow the needed interaction with GTDOptions and you should not need to use it. The command will fail if the driver is in "server mode" to avoid problems with the client program.

6.7 The MPC_NOTESTPREFS command

```
Command: MPC_NOTESTPREFS
Data:   -
Error:  MPC_SERVER    (GTDriver is in server mode)
```

This command is sent by GTDOptions when you press the OK gadget after finishing the test of some preferences setting. It was implemented only to allow the needed interaction with GTDOptions and you should not need to use it. The command will fail if the driver is in "server mode" to avoid problems with the client program.

6.8 The MPC_NEWBUTTONS command

```
Command: MPC_NEWBUTTONS
Data:   address of the name of the p-button file
Error:  MPC_SERVER    (GTDriver is in server mode)
```

This command tells GTDriver to load a new p-buttons definition. Data field must be a string pointer pointing to the address of the name of the file containing such definition. The command will fail if the driver is in server mode to avoid problems with the client program.

7 Troubleshooting

- When I move the stylus, the screen pointer behaves in a crazy way.

  Check that Emulation and Baud Rate settings are the same for the tablet and for the driver. If any one of them is set to different values `GTDriver` won't work correctly. If you configured the tablet using the init string check the commands in the string (consult your tablet manual)

- The screen pointer cannot reach the screen limits

  Check that DPI settings are the same for the tablet and for the driver. If they are set to different values `GTDriver` won't work correctly. Also check that XDim and YDim correspond to the dimensions of the active area.

- Only a small area of the tablet is mapped to the screen mouse movements.

  Check that DPI settings are the same for the tablet and for the driver. If they are set to different values `GTDriver` won't work correctly. Also check that XDim and YDim correspond to the dimensions of the active area. Another possibility is that you set a clip region (check clip regions dimensions.

- The screen pointer follows your stylus movement too slowly.

  This problem may be due to a too high sampling rate of the tablet: this causes an overloading of the driver and of the serial device. It can be solved by reducing the sampling rate (it's a tablet parameter and not a driver parameter) or by reducing the baud rate (both in the driver and in the tablet). You may also try to increase the priority value.

- The screen pointer movements are jerky.

  This problem may be due to a too low sampling rate of the tablet. It can be solved by raising the sampling rate (it's a tablet parameter and not a driver parameter) or by raising the baud rate (both in the driver and in the tablet). The problem may be also due to other porgrams interfering with `GTDriver`, so you may try to increase the priority value.

- My tablet seems to loose stability when I do water sports with it.

  Remember to remove seaweeds from the bottom of the tablet periodically.

8 How to have your tablet supported

If your tablet is not supported by `GTDriver`, contact the authors. In this section it is explained which information you should send them to have your tablet supported. To know how to reach the authors see Chapter 13 [Disclaimer and authors info], page 34.

In your tablet manual you should find a section with a name similar to "DATA OUTPUT FORMATS". Usually tablets have an ASCII mode and a BINARY mode. Send us the binary format preferably (because they're faster).

The binary format is usually indicated in a table like the following one (valid for the Summagraphics emulation):

```
     MSbit                                  LSbit
BYTE|  7    6    5    4    3    2    1    0
----|-----------------------------------------
 1  |  1    PR   T0   SX   SY   C2   C1   C0
 2  |  0    X6   X5   X4   X3   X2   X1   X0
 3  |  0    X13  X12  X11  X10  X9   X8   X7
 4  |  0    Y6   Y5   Y4   Y3   Y2   Y1   Y0
 5  |  0   Y13   Y12  Y11  Y10  Y9   Y8   Y7
```

We need the table for the binary emulation of your tablet, along with the legenda of the acronyms (i.e. PR = proximity bit, C2 = button 2 and so on). Don't forget to report the name of the emulation also!

If you have only an ASCII format, it should be found in the manual in a format similar to the following one:

```
< 508LPI   XXXX,YYYY,C CR LF
> 508LPI   XXXXX,YYYYY,C CR LF
```

with a table stating the C possible values and the meaning (i.e. C can vary from '0' to '7', and it means the value of the three buttons in the binary format. i.e. if C2=1, C1=0 and C0=1 then C='5')

How to send data formats. E-mail is preferred because it is faster and if you have an e-mail address we can solve any problem in the implementation easily and quickly. If you don't have an e-mail address you can send the data by mail (in this case it would be better if you send a copy of the most important pages of the tablet manual).

9 Known bugs or problems

There are no known bugs. On the Acecad tablet we used in the developement most emulations work quite well. If you change device or emulation in your tablet you should kill the driver before doing so, as some unexpected byte is sent to `GTDriver` in response to the change.

Wacom emulation could not be tested, because no beta tester with such tablet model has been found. The emulation routine is based on a PD source for Unix. We decided to implement it in this release (even if we couldn't test it) because we hope to find someone with this tablet. If you have one and the emulation does not work properly, please contact us to help us get rid of the problem.

We found that some init strings do not work properly. It could happen that the first command is executed correctly by the tablet, but the following are ignored. As far as we know, the commands are sent correctly by `GTDriver`: the problem should be due to some undocumented hardware constrain in the tablet.

Under critical conditions some data can be lost by the driver, but don't panic: `GTDriver` will resynchronize on the following packets, and you should not experience any real problem. After losing data it may happen that a mouse button seems to be pressed while it is not. Press and release it to solve this problem.

10 Why GTDriver is not a commodity

You may ask why GTDriver is not a commodity. The reason is that commodities have been misunderstood by many programmers. Commodities were defined to simplify the job of writing software that NEEDS to get events from the input device. The programs supposed to need this feature are basically screen blankers and system monitors. I know it's nice to have programs that can be recalled using a shortcut and that have hidable GUIs, but the main problem is that whenever you press a key, every commodity gets the input event and lets it flow down in the commodities chain. If you have many commodities running at the same time you get a heavy slowdown in all of the system activities. GTDriver does not need to know about input events (it only generates input events), so it is not a commodity.

11 Future improvements

We plan to add:

- the ability to start programs and to execute Arexx scripts by pressing p-buttons;
- Arexx port (if someone proves it is useful :-));
- new multiwindow layout for `GTDOptions`;
- GadTools and MUI version of `GTDOptions`;
- ability to set XDim, YDim and Clip parameters in `GTDOptions` using the tablet pointing device;
- more and more predefined p-buttons masks;
- new tablet emulations (if you send us any ...);

... and any interesting feature you suggest us !

12 Acknowledgments

We must thank (in alphabetical order):

Federica Colla, whose kind patience and support have been unvaluable to our project;
Giovanni Gentile for eight color icons;
Remco Straatman who gave us many good ideas and helped with the documentation;
Sebastiano Vigna for having shown us the problem and having given the first suggestions;
Vittorio Calzolari who nicely lent us the tablet used during the development of the software;
the Amiga user group of Milan... just because they exists!

and last but not least, the beta testers

Andreas Geierlehner
Carlo Zambellini
Giovanni Gentile
Jan Robijns
Peter Larsen
Remco Straatman

13 Disclaimer and authors info

The unregistered version is freely distributable as long as all of its files are included in their original form without additions, deletions, or modifications of any kind, and only a nominal fee is charged for its distribution. This software is provided **AS IS** without warranty of any kind, either expressed or implied. By using `GTDriver`, you agree to accept the entire risk as to the quality and performance of the program.

Comments, complaints, desiderata are welcome.

`GTDriver` was developed by Roberto Attias

`GTDOptions` was developed by Marco Zandonadi

Please, send any bug reports regarding `GTDriver` and request for new tablet support to Roberto Attias:


```
e-mail: attias@ghost.sm.dsi.unimi.it
UUCP:   roby@utopia.adsp.sub.org
mail:   Roberto Attias
        Via Lissoni, 5
        20162-I Milano
        ITALY
```


and any bug reports regarding `GTDOptions` to Marco Zandonadi:


```
e-mail: zandonad@ghost.sm.dsi.unimi.it
UUCP:   marcoz@utopia.adsp.sub.org
mail:   Marco Zandonadi
        Via Deledda, 23
        20052-I Monza (MI)
        ITALY
```

14 Registration and upgrades

`GTDriver` is a shareware program. The unregistered version has no functional limitations, but exits after 10 minutes of use.

To become a registered user you have to send us a registration fee and your full name and address.

Registration fees and upgrades:

- Internet registration: if you have an internet address, send us $10 US. We will send you an uuencoded keyfile that upgrades the unregistered version to registered. The keyfile contains your encrypted name and must not be spreaded.
- Mail registration: if you don't have an internet address, send us $13 US. We will send you a disk by mail. This disk will contain the latest version of `GTDriver` together with a keyfile that upgrades the unregistered version to registered. The keyfile contains your encrypted name and must not be spreaded.
- Internet upgrades: If you are a registered user and you have ftp access, you can download new versions of `GTDriver` without additional fees.
- Upgrade by mail: If you are a registered user and you don't have ftp access, send us $3 US. We will send you a disk by mail. This disk will contain the latest version of `GTDriver` together with a keyfile that upgrades the unregistered version to registered.

If you like `GTDriver` and use it, please register yourself, helping us to enhance this product. If you don't think this program is useful enough to pay for it, please, at least e-mail (or mail) us your suggestions, so that we can make improvements.

Concept Index

Q

R

S

U

W

X

Y

Table of Contents