Rapid Palm GUI

Contents

1 What is Rapid Palm GUI ?

It is a library to help you build C programs on Palm compatible devices.
Its design goal was to provide simple functions for some typical Palm user interface objects and an overall program structure so that you can create nice looking programs rapidly.
It is designed for use with the OnboardC freeware C compiler for Palm devices.

2 Why use Rapid Palm GUI ?

When you want to create C programs on the Palm itself (i.e. not on a PC or Mac), then the OnboardC compiler is a good choice, because it creates fast and small programs. However, with OnboardC you have to use the Palm API, with hundreds of functions, data structures, events, ... This is low level programming with a very steep learning curve. Rapid Palm GUI, however, simplifies the programming of user interfaces considerably.

3 What is in the package ?

The Rapid Palm GUI zip file contains:

Name in ZIP	Name on Palm	Description
C_gui_h.PDB	C_gui.h	The header file you have to include in your code.
C_gui_c.PDB	C_gui.c	The source file that implements Rapid Palm GUI. Include it in your OnboardC project. You do not have to understand or change anything in this file.
C_gui_res.PRC	C_gui.res	A resource file that is used by „C_gui.c“. Simply include it in your projects.
C_HelloWorld_c.PDB	C_HelloWorld.c	An almost empty example program that you can use as base for your own developments.
HelloWorld.PRC	HelloWorld	The HelloWord example as complete program.
C_guiTester_c.PDB	C_guiTester.c	An example program that uses all the features of the Rapid Palm GUI. I used it for testing purposes.
C_guiTester_res.PRC	C_guiTester.res	An example resource file for the above guiTester program, containing a few bitmaps and a menu. Include it in the project when you want to compile the guiTester programm.
GuiTester.PRC	guiTester	The guiTester example as complete program.

All source files are delivered as uncompressed DOC files.

4 What else do I need ?

What you have to have:

- A Palm compatible device with at least OS V3.5.
- The OnboardC compiler, written by Roger Lawrence. This is a real C compiler that creates small and fast programs. The compiler package contains a simple DOC file editor, too. I used V2.0.1. I got it from http://sourceforge.net/projects/onboardc. There are two header files included, please use „OnBoardHeaderV35.pdb“since it contains more definitions.
- A resource file builder is needed if you want to use bitmaps or menus (or other resources not covered by Rapid Palm GUI). I use RsrcEdit because it runs on the Palm.

What you should already know:

- How to program in C
- How typical Palm programs look and act like (from the outside)
- A basic understanding of event driven programming would be helpful
- How to handle the OnboardC compiler (and RsrcEdit, if you need it)

5 A simple program's structure

With Rapid Palm GUI, the overall program structure is simple. Let us consider a program that has just one page, i.e. creates a set of user interface objects (like buttons and text fields) that do not change while the program runs. The structure is as follows:

1. When a program starts, the GUI takes control and then calls your function

   ```
   giu_init()
   ```

   Here you create a user interface consisting of buttons, edit fields and so on. You do so by calling functions like

   ```
   gui_mkButton(...)    // makes a button
   gui_mkText(...)    // makes a text field
   ```

   These objects are then created and drawn by the GUI.

2. After your gui_init() terminated, the user will enter text in fields, click on buttons, and so on. Text entry is handled by the Palm OS.

3. Button taps result in calls to callback functions that you wrote and told the GUI about in the gui_mkButton(...) calls. These callback functions do the work in your program.
   A callback function has a very simple interface, for example:

   ```
   Void doSomething(Uint16 id)  // id is the number of the tapped button
   ```

4. When the user quits your program your function

   ```
   giu_exit()
   ```

   is called, where you can clean up, i.e. save some data.

Summary:
- The overall program control is in the GUI.
- You create user interface objects with callback functions you wrote.
- When the user acts on this objects your callback functions are called by the GUI, and do your program's work.

Example: In the HelloWorld example the program structure is as follows:
- gui_init() creates one button with title „tap me" and callback function doSomething()
- doSomething(...), when called, shows a „hello world" message
- when your program is left then gui_exit() is called, which does nothing

There are some more must-have functions in this example, one for idle processing (could make animation when the user is passive), one for processing menu selections, and others. In the HelloWorld example, they are empty.

So a „hello world" program with Palm-like look and feel consists of a few lines only.

6 A large program's structure

Larger programs change their user interface at runtime. With Rapid Palm GUI, this can be done as follows:

1. When a program starts, the GUI takes control and then calls your function

   ```
   giu_init()
   ```

2. Here you call one of your functions that creates your entry page

   ```
   entryPage(0)
   ```

   Then gui_init terminates.

3. Your entryPage() function creates and shows the objects on the entry page:

   ```
   gui_hide(0,999);   // hide all previous objects (see step 5)
   gui_mkButton(...,configPage)   // a button to jump to config page
   gui_mkText(...)   // makes a text field
   ...
   ```

   These objects are then created and drawn by the GUI.

4. After these functions terminated, the user can tap on the button created above, then the GUI calls your handler configPage() which could be defined as follows:

   ```
   void configPage(Uint16 id) {
   gui_hide(0,999);   // hide all previous objects
   gui_mkButton(...,entryPage)   // make a button to jump back
   gui_mkText(...)
   ... }
   ```

5. After configPage() created its layout, the user can tap on the the button created above, then the GUI calls your handler entryPage(), described in step 3 above.

6. This function then erases all objects shown (i.e. those created by configPage()) and re-creates the entry page objects. They are not really re-created but simply drawn again.

Summary:

- A large program can be implemented as a couple of page-creating functions.
- gui_init() calls the first of these.
- A page-creating function cleares the screen and then creates its objects, including buttons to go to other pages, via button handlers that are page-creating functions.

Of course, apart from buttons there are other ways to jump to other pages, like menus, hard button presses and so on. More on this in the following chapters.

Example: The guiTester example is structured like that. However, it is somewhat large because I wrote it to test every feature that the GUI offers.

7 How to build you own program

To build your own program, you should proceed like that:

1. Install the OnboardC compiler (including Assembler and big header file) and a DOC file editor (there is one included in the OnboardC package).
2. Install the resource editor RsrcEdit. However, you do not need it for very simple programs.
3. Install the DOC and RES files included in the Rapid Palm GUI package.
4. Copy or rename the C_HelloWorld.c file to use it as template for your own program.
5. Select the DOC editor you are using in the OnboardC menu Options -> Choose Editor.
6. Create an OnboardC project and include C_HelloWorld.c (or your copy), C_gui.c and C_giu.res.
7. Your project's Type (in OnboardC's main screen) should be „appl".
8. The Creator (in OnboardC's main screen) is a unique id that the Palm uses to distinguish all programs, and to associate the program's preference and data files with the program itself. As long as you are just experimenting on your own device you may use „Exa0". However, before you give your program to other people please visit http://dev.palmos.com/creatorid to get your own. See chapter 8.1.6.
9. Extend C_HelloWorld.c (or your copy) as you like, add more source files, ...

8 Function reference

For exact prototypes of the functions described above, please refer to the C_gui.h header file or to the C_HelloWorld.c example.
Every function is used in C_guiTester.c, so look there if in doubt how to use it.

8.1 Functions your program has to provide

8.1.1 gui_init() – the startup code

This is called by the GUI when your program starts. Use it to create user interface objects like buttons and text fields.

8.1.2 gui_exit() – the cleanup code

This is called when your program terminates. Here you can sava data. You do not need to hide all user interface objects here. It is a bad idea to put any user interaction in here (I did it in guiTester to test this function). Leave it empty if you do not need it.

8.1.3 gui_idle() – animation

This is called whenever your program waits for user input, about 100 times per second. You can put animations in here, for example. Leave it empty if you do not need it.
Idle processing is stopped as long as another window is on top, e.g. when the menu is open.

8.1.4 gui_menu(int id) – handle menus

This is called when the user has selected a menu entry. The id parameter is the menu item number specified in the menu resource. The Cut, Copy and Paste commands on text fields (id 9000-9002) are handled by the GUI itself. You can put more menu items in the MBAR 9000 resource in C_gui.res (with RsrcEdit). Leave this function empty if you do not need it. Simple programs do not necessarily have a menu.

8.1.5 gui_key(int c) – handle graffitti and button presses

This is called when the user enters a graffitti character, taps on a silkscreen button (like the Find button) or presses hard buttons. Normally you should simply do nothing here, and return 0. Then the GUI and Palm OS handle the text input for fields, and they handle the silkscreen and hard buttons. If you want to interfere, do it and return 1 only for the values of c that you handle, indicating that you have swallowed this particular key code.
If you, for example, return 1 for all values of c, then your program can only be left by a soft reset, since the Palm OS has no chance to know about button presses or silk screen taps.
Run the guiTester program, go to „key input“screen, for typical values of c.

8.1.6 gui_creator() – your program’s creator id

Return your program’s unique creator id. This is identical to the Creator value you have to enter in OnboardC’s main window. See step 8 in capter 7.

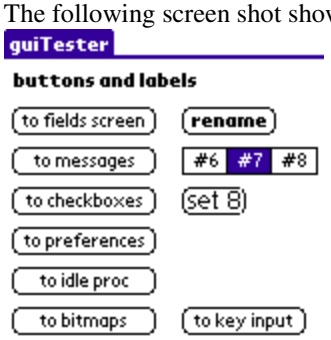
8.2 Functions that the GUI provides

How to create user interface objects

8.2.1 gui_mkButton() – create a button

The arguments are	id	a unique number to access the object later, range 0...ELEMCNT-1
	title	the name shown on the button
	x	horizontal coordinate, distance from left border, in pixels
	y	vertical coordinate, distance from top border, in pixels
	wid	the width, in pixels
	hei	the height, in pixels
	act	the callback function for user taps, type void (*)(Uint16)

Nothing is returned.
When called again for the same id, the original object is simply redrawn.
The additional parameter font applies, see chapter 8.4.
The following screen shot shows some bottons, all are rounded rectangles:



8.2.2 gui_mkPushButton() – create a push button

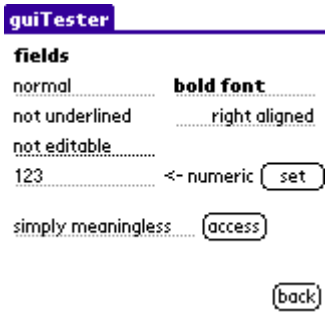
The arguments are		
	id	a unique number to access the object later, range 0...ELEMCNT-1
	title	the name shown on the button
	x	horizontal coordinate, distance from left border, in pixels
	y	vertical coordinate, distance from top border, in pixels
	wid	the width, in pixels
	hei	the height, in pixels
	group	the push button‘s group, or zero
	state	0 for not selected initially, 1 for selected
	act	the callback function for user taps, type void (*)(Uint16)

Nothing is returned.
When called again for the same id, the original object is simply redrawn.
When several push buttons have the same group number, then only one of them is selected (inverted). Whenever the user taps on another push button of the same group, the previously selected one is unselected.
So push buttons are used to offer choices to the user.
The additional parameter font applies, see chapter 8.4.
The above picture shows three push buttons named ‚#6“, ‚#7“and ‚#8“.

8.2.3 gui_mkField() – create a text field

The arguments are		
	id	a unique number to access the object later, range 0...ELEMCNT-1
	x	horizontal coordinate, distance from left border, in pixels
	y	vertical coordinate, distance from top border, in pixels
	wid	the width, in pixels
	hei	the height, in pixels
	len	max text length in characters
	value	initial text value

Nothing is returned.
When called again for the same id, the original object is simply redrawn.
Editing is handled by the GUI and the Palm OS, including Copy, Cut and Paste (as long as you leave them in the MENU 9000 resource in C_gui.res). The additional parameters font, editable, underlined, leftalign and numeric apply, see chapter 8.4.
Here is a screenshot with some fields:



8.2.4 gui_mkCheckbox() – create a checkbox

The arguments are	id	a unique number to access the object later, range 0...ELEMCNT-1
	title	the name shown on the checkbox
	x	horizontal coordinate, distance from left border, in pixels
	y	vertical coordinate, distance from top border, in pixels
	wid	the width, in pixels
	hei	the height, in pixels
	group	the checkbox group, or zero
	state	0 for not selected initially, 1 for pressed
	act	the callback function for user taps, type void (*)(Uint16)

Nothing is returned.
When called again for the same id, the original object is simply redrawn.
When several checkboxes have the same group number, then only one of them is selected. Whenever the user taps on another one of the same group, the previously selected one is unselected.
So checkboxes are used to offer choices to the user.
The additional parameter font applies, see chapter 8.4.
Here is a screenshot with four checkboxes, where the upper three are in the same group:

8.2.5 gui_mkLabel() – create a text label

The arguments are	id	a unique number to access the object later, range 0...ELEMCNT-1
	title	the name shown on the checkbox
	x	horizontal coordinate, distance from left border, in pixels
	y	vertical coordinate, distance from top border, in pixels

Nothing is returned.
When called again for the same id, the original object is simply redrawn.
A label is simply a text that cannot be selected or edited by the user.
The additional parameter lfont applies, see chapter 8.4.

8.2.6 gui_mkBitmap() – create a bitmap image

The arguments are	id	a unique number to access the object later, range 0...ELEMCNT-1
	res	the number of a bitmap resource
	x	horizontal coordinate, distance from left border, in pixels
	y	vertical coordinate, distance from top border, in pixels
	wid	the width, in pixels
	hei	the height, in pixels
	act	the callback function for user taps, type void (*)(Uint16)

Nothing is returned.
When called again for the same id, the original object is simply redrawn.
This is similar to a button, but a small image (a bitmap) is shown.
You can create bitmap resources with the RsrcEdit program, either in C_gui.res or in your own resource file that you add to your OnboardC project.

8.2.7 gui_mkList() – create a list in a box

The arguments are	id	a unique number to access the object later, range 0...ELEMCNT-1
	x	horizontal coordinate, distance from left border, in pixels
	y	vertical coordinate, distance from top border, in pixels
	wid	the width, in pixels
	hei	the height, in pixels

	items	array of strings to present in the list (char **)
	cnt	number of strings in that array
	act	the callback function for user taps, type void (*)(Uint16)

Nothing is returned.
When called again for the same id, the original object is simply redrawn.
See below a screenshot with a list on the left side, showing 4 elements.
When the act function is called, it can find out what list element has been selected by calling gui_getValue(), see 8.2.12 below.
You can change the list elements later, see 8.2.13 below.
The memory that „items“points to may not be altered after this call, since the Palm OS reads whenever it draws the list.

8.2.8 gui_mkPopupList() – create a popup list

The arguments are	id	a unique number to access the object later, range 0...ELEMCNT-1
	x	horizontal coordinate, distance from left border, in pixels
	y	vertical coordinate, distance from top border, in pixels
	wid	the width, in pixels
	hei	the height, in pixels
	items	array of strings to present in the list (char **)
	cnt	number of strings in that array
	act	the callback function for user taps, type void (*)(Uint16)

Nothing is returned.
When called again for the same id, the original object is simply redrawn.
See above a screenshot with a list on the right side, currently only showing the selection.
When the act function is called, it can find out what list element has been selected by calling gui_getValue(), see 8.2.12 below.
You can change the list elements later, see 8.2.13 below.
The memory that „items“points to may not be altered after this call, since the Palm OS reads whenever it draws the list.

How to manipulate user interface objects

8.2.9 gui_setText() – change text of a GUI object

The arguments are	id	a unique number to identiy the object, range 0...ELEMCNT-1
	value	the new text string

Nothing is returned.
When id refers to a field, the string shown is replaced by value.
For a button, push button or checkbox, the object’s title is changed.
For other objects, the function does nothing.

8.2.10 gui_getText() – get text value of a field

The arguments are	id	a unique number to identify the object, range 0...ELEMCNT-1
	buf	a buffer to hold the string fetched

Nothing is returned.
When id refers to a field, the string shown is copied into buf, which has to be large enough (not difficult since you specified a max len for the field).
For other objects, the function does nothing.

8.2.11 gui_setValue() – change number/state of a GUI object

The arguments are
- id — a unique number to identiy the object, range 0...ELEMCNT-1
- v — the new int value

Nothing is returned.
When id refers to a field, the string shown is replaced by a text representation of v.
For a push button or checkbox, the object is selected for v not zero, and deselected for v is zero.
For a bitmap object, the bitmap resource with number v is shown, i.e. the picture changes.
For a list or popup list, an item is selected by index (where zero is the first in the list).
For other objects, the function does nothing.

8.2.12 gui_getValue() – get number/state of a GUI object

The argument is
- id — a unique number to identiy the object, range 0...ELEMCNT-1

The int value is returned.
When id refers to a field, the string shown is converted to an int and returned.
For a push button or checkbox, the object state is returned, 1 for selected and 0 for deselected.
For a list or popup list, the selected item's index is returned (where zero is the first in the list).
For other objects, the function does nothing.

8.2.13 gui_setList() – change a list or popup list

The argument is
- id — a unique number to identiy the object, range 0...ELEMCNT-1
- items — array of strings to present in the list (char **)
- cnt — number of strings in that array

Nothing is returned.
The list or popup list now has a new contents.

8.2.14 gui_hide() – hide GUI objects

The arguments are
- lo — lower range limit
- hi — upper range limit

Nothing is returned.
All GUI objects with ids in the range lo..hi are hidden. It is not an error when hi is >= ELEMCNT.

8.2.15 gui_show() – show GUI objects

The arguments are
- lo — lower range limit
- hi — upper range limit

Nothing is returned.
All GUI objects with ids in the range lo..hi are shown. It is not an error when hi is >= ELEMCNT.

8.2.16 gui_title() – set screen title

The argument is
- buf — the new title

Nothing is returned.
Sets the program's title at the screen's top. Typically you use your program's name as the value for buf.

How to show messages

8.2.17 gui_msg() – show a message box

The arguments are
- style — the message box style, see below
- m — the format string
- ... — more args according to the format string

Returns 0 if the first button is pressed by the user, and 1 for the second (if any).
A meessage box is popped up.
The style parameter selects the kind of message box:

0	a simple message box with an ok button.
1	an error message box with an ok button.
2	a warning message box with buttons ok and cancel.
3	a confirm message box with buttons ok and cancel.
4	a choose message box with buttons yes and no.

The m parameter and following optional parameters specify a format string with embedded arguments.
This works a bit like the printf() function in LINUX systems.

A few examples for m and following args:

"number %d long %ld char %c string %s",1,300000,97,"hey"
-> number 1 long 300000 char a string hey

"10 digit number %10d",123
-> 10 digit number 123

8.2.18 gui_dbg() – show a debug message

The arguments are wait a boolean, specifiying whether the programs waits for a pen tap
m the format string
... more args according to the format string

Returns nothing.
A debug text is written in the screen's top left area.
The wait parameter tells if the program should halt after printing the string, and wait for a pen tap.
The m parameter and following optional parameters are as in 8.2.17.

Preference data I/O

8.2.19 gui_writePrefs() – save preferences data

The arguments are buf a pointer to a buffer, type char *
len data length of buf, in bytes

Returns nothing.
A memory region is saved, typically containing state data that your program has to use when it is called the next time. Define a global struct (say „g") and then save it with gui_writePrefs((char *)&g,sizeof(g)).
Such a call can be put into the gui_exit() function, for example.

8.2.20 gui_readPrefs() – load preferences data

The arguments are buf a pointer to a buffer, type char *
len data length of buf, in bytes

Returns nothing.
What a previous call to gui_writePrefs() has saved, is loaded back now. Typically containing state data that your program has to use again.
If there is no such data, or if it has had a different length, then buf is filled with zeros, byte after byte.

8.3 Configuration at compile time

8.3.1 ELEMCNT – number of GUI objects

This is the max number of user interface objects. Defined in C_gui.h. Configures an array in C_gui.c where the object data is strored. Since each entry takes a few bytes, this macro's value should not be much higher than needed. Change it as you need.

8.4 Configuration at run time

There is a number of configuration parameters that you can use to specify more details for the GUI creation functions, like gui_mkButton(). They are not function arguments, but members of the global struct variable gui, declared in C_gui.h.
You set them immediately before you call the object creation function that you want to modify, example:

```
gui.font = 1;   // set bold font
gui_mkButton(...)   // make a button with a bold title
gui_mkButton(...)   // make a button with a standard title
```

The first line above sets the parameter, the second uses and then resets it, the third line uses the default value again.
I did it that way so you have simple creation functions without too many parameters, and if you want some more, it is possible, too.

8.4.1 font – set the text font

This is the font used for the next GUI object creation call. 0 is standard and default, 1 is bold, 3 is large, ...

8.4.2 lfont – set the label font

This is the font used for the next GUI object creation call. 0 is standard, 1 is bold and default, 3 is large, ...

8.4.3 editable – fields can be editable or not

This is the editable property for text fields, 0 for not editable, 1 for editable, which is default. A counter that is controlled by your program may be a candidate for a field that is not editable.

8.4.4 underlined – fields can be underlined or not

This is the underlined property for text fields, 0 for not underlined, 1 for underlined, which is default.

8.4.5 leftaligned – fields can be leftaligned or not

This is the leftaligned property for text fields, 0 for right aligned, 1 for left aligned, which is default. My impression is that editing right aligned fields does not work properly.

8.4.6 numeric – fields can be numeric or not

This is the numeric property for text fields, 0 for not numeric (accepting any characters, which is default), 1 for numeric (accepting numbers only).

9 Questions you may have

9.1 „It does not work“ section

I get compiler errors even when trying to build the HelloWorld example.
The OnboardC package V2.1 contains two header files. „OnBoardHeaderV35.pdb“is newer and contains more definitions, please use this one. If you already installed the other one, please make OnboardC forget it (menu Options -> Delete Header).

9.2 „How can I“ section

Do you support colors or grayscale ?
Yes. The bitmaps can have colors and grayscale.

Do you support hi-res screen resolutions ?
Not yet. However, if you resize the FORM 9000 resource in C_gui.res, and maybe call WinScreenMode() appropriately, it may work.

Do you support multi-line fields, scrollbars, ... ?
Not yet.

Do you support drawing of rectangles, lines and so on ?
Not yet. However, you could do this by calling the Palm API routines yourself, such as WinDrawLine().

I want my application to have a cool icon, how can I do that ?
Take RsrcEdit and create an Icon resource (tAIB 1000), 32 x 32 pixels. For a small icon, create tAIB 1001 with width 15 and height 9.

How can I print the DOC source files ?
Install windows programs like BigDoc or DocReader. They can open the DOC files that are synced into your ...\Palm\...\backup directory, and convert them to text files.

I want to access additional PALM OS features, where is the description ?
Refer to the thousands and thousands of documentation pages in http://www.palmos.com/dev/support/docs/. The companion and reference are the right place to begin.

The OnboardC compiler does not support sprintf(), strcpy(), memcmp() and so on, what can I do ?
String management functions are available with slightly different names. See the String Manager Functions section in Part II of the reference document mentioned above. Memory handling functions are in the Memory Management section. And you can find a lot of prototypes in the OnboardC compiler's header file.

Can't I delete GUI objects to save memory ?
No. GUI objects do not take a lot of memory. I had permanent stability problems when deleting objects.

Can't I create menu entries in the program, without using resources ?
No. I found no reasonable way to do this without resources.

Can't I use my own form resources, button resources, ... ?
No. It is simpler to do it the Rapid Palm GUI way.

9.3 „Misc“ section

How large are files made with this GUI ?
Quite small actually. The guiTester program contains a lot of user interface objects and is 16 K in V1.1.

What quality does your GUI have ?
I implemented some consistency checks, and I tested it thorougly (using the guiTester example), including hours of automatic Gremlin tests in the Palm OS Emulator (POSE). I found a lot of bugs and fixed them.

Can I participate in this project ?
Sure. Send me your extensions.

How can I ever thank you ?
Write interesting freeware.

10 Version history

Version	Component			
	Description	**C_gui.c, .h**	**guiTester**	**HelloWorld**
1.0	Basic version			
1.1	Added here and there	Added lists and popup lists		Unchanged

11 Legal statement

YOU AGREE THAT BLABLA USE OF THE SOFTWARE ACKNOWLEDGES THAT YOU BLABLA BLABLA THIS BLABLA, UNDERSTAND IT, AND BLABLA BLABLA BE BOUND BY ITS TERMS AND CONDITIONS BLABLA BLABLA BLABLA AGREEMENT BLABLA THE SAME FORCE AND BLABLA AS A SIGNED BLABLA. IF YOU DO NOT AGREE BLABLA BLABLA BLABLA TERMS BLABLA THIS LICENSE, YOU ARE NOT BLABLA BLABLA BLABLA BLABLA BLABLA.

DISCLAIMER OF WARRANTIES. THE BLABLA IS PROVIDED TO BLABLA " BLABLA BLABLA " WITHOUT BLABLA BLABLA BLABLA ANY BLABLA. BLABLA DISCLAIMS ALL IMPLIED BLABLA, INCLUDING BUT NOT LIMITED TO, THE BLABLA BLABLA OF BLABLA, BLABLA FOR A BLABLA BLABLA, TITLE AND BLABLA, WITH RESPECT TO THE BLABLA. THE ENTIRE BLABLA ARISING OUT OF BLABLA BLABLA BLABLA BLABLA BLABLA BLABLA BLABLA REMAINS WITH BLABLA.
BLABLA DOES NOT WARRANT BLABLA BLABLA BLABLA BLABLA IN THE SOFTWARE WILL MEET LICENSEE' S BLABLA BLABLA BLABLA BLABLA, BLABLA BLABLA BLABLA BLABLA THE SOFTWARE BLABLA BLABLA BLABLA BLABLA UNINTERRUPTED OR ERROR-FREE, BLABLA BLABLA BLABLA BLABLA.
THE FOREGOING BLABLA BLABLA BLABLA BLABLA APPLICABLE LAW IN BLABLA BLABLA BLABLA BLABLA.

LIMITATION OF LIABILITY. TO THE FULL BLABLA PERMITTED BY APPLICABLE LAW, IN NO BLABLA BLABLA BLABLA BLABLA BE LIABLE FOR BLABLA BLABLA BLABLA BLABLA, INDIRECT, INCIDENTAL, OR SPECIAL BLABLA BLABLA BLABLA BLABLA WHATSOEVER, INCLUDING WITHOUT LIMITATION, BLABLA BLABLA BLABLA BLABLA, BLABLA BLABLA BLABLA BLABLA, BLABLA BLABLA BLABLA BLABLA, AND THE LIKE, ARISING OUT OF THIS BLABLA BLABLA BLABLA BLABLA OR INABILITY TO BLABLA BLABLA BLABLA BLABLA DAMAGES.

Got it ?