EasyPLUGINs

COLLABORATORS			
	TITLE : EasyPLUGINs		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		December 31, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

EasyPLUGINs

1.1 EasyPLUGINs

```
                EasyPLUGINs v1.3 (28.11.97)



      Introduction

      Distribution

-----------------------------------------

PLUGINs:

S  E  G

.              bitmapimage_plugin
.  .           dclistview_plugin
.  .  .        iconbox_plugin
.  .           icongad_plugin
.  .  .        popasl_plugin
.  .           rawkey_plugin
.  .  .        register_plugin
.  .  .        simplegauge_plugin
.  .  .        text_plugin
.  .  .        toolbar_plugin
.  .  .        xygadget_plugin

(S) - PLUGIN comes with source code.
(E) - PLUGIN comes with example/s of usage.
(G) - PLUGIN is EasyPLUGINs Style Guide Compliant.

-----------------------------------------



      Style Guide
```

1.2 Introduction

EasyPLUGINs is a collection of PLUGINs for EasyGUI, the GUI module for AmigaE users by Wouter van Oortmerssen and Jason R. Hulance.

These PLUGINs are by a number of different authors; they all include documentation. Some also come with example sources and/or the source code for the modules themselves. The source code provided is for educational purposes only; redistribution of these sources, changed or not, is against the conditions of usage for this package. If you really want publically propagated changes to any of the PLUGINs included in this package, contact the author and see what you can work out.

1.3 Distribution

- EasyPLUGINs should be considered as freeware, but users may NOT redistribute a changed version of any of the PLUGINs included in the package. Instead, they should contact the original authors and have those additions, bugfixes etc. happen at the source (no pun intended :)

- If people release software based on one or more PLUGINs, they should credit the author of those PLUGINs in their documentation and/or software; if it's shareware/commercial, a registered copy

```
  to the author(s) wouldn't go astray either (not a requirement,
  but it would be polite...)
```

1.4 Registered tag bases and programmer ID's

```
Ralph Wermke                                $00
<wermke@gryps1.rz.uni-greifswald.de>

    Registered PLUGIN ID's:

        register_plugin:                    $01
        simplegauge_plugin:                 $02
        popasl_plugin:                      $07


-------------------------------------------------


Ali Graham                                  $01
<agraham@hal9000.net.au>

     Registered PLUGIN ID's:

        iconbox_plugin:                     $01
        text_plugin:                        $02
        toolbar_plugin:                     $03
        xygadget_plugin:                    $04


-------------------------------------------------


(Currently unused)                          $02
                                         -> $7E

-------------------------------------------------


(Code for private PLUGINs)                  $7F
```

1.5 Style Guide

```
                EasyPLUGINs Style Guide

    by Ralph Wermke and Ali Graham.


This is a document describing a useful, elegant and easily
extensible standard style for PLUGINs. While its acceptance
is by no means compulsory, the use of this framework as a
```

template for the development of your PLUGINs can speed up
your work and improve the functionality of the PLUGINs you
produce.

Even if you do not want to program your own PLUGINs, but
only want to use the ones provided, you should probably
read this Style Guide anyway, especially the section
on
tags
; it will give you a better understanding of the
PLUGINs that use this standard and the manner in which
they work.

This, then, is the general procedure that you should
follow in the development of any PLUGIN:

decide the features that your PLUGIN should have

construct the object

list the tags to be used by the PLUGIN

build the public interface to the PLUGIN
(constructor, destructor, set() & get() methods)

actually implement the desired functionality

test, debug, extend, enhance, and document

1.6 Style Guide (1)

1. Design

Always the most important part of programming; you
may have some pre-existing code that you want to adapt,
or merely a few ideas about what you want to write.
Take out a pen and a piece of paper, sit down and plan
it all - it saves a lot of time in the long run. :)

This is one of the more valuable things I've learnt

in my studies of Computer Science at university; that
you should attempt to minimize the time actually spent
coding. Thinking, testing and debugging should be more
important parts of the process.

1.7 Style Guide (2)

2. Construction

Adopt an easy to remember, reasonably brief name that
is representative of the functionality of the PLUGIN.
Then add '_plugin' to that, like so:

```
OBJECT <xyz>_plugin PRIVATE
   a
   b
   c
ENDOBJECT
```

The various values defined inside the object; these
should all be defined as PRIVATE unless you have a
very good reason, as this encourages encapsulation/
data-hiding/good OO practice in general. Instead, use
the
set() & get()
methods to access these values in
a more strictly defined manner.

I also suggest separating the values inside the
PLUGIN object into two sections; initially, place
all of the variables that will be accessible by
set() or get() (or both) at the top of the object,
and then under those place the truly private
variables.

No global variables should be defined in PLUGIN
modules, except library bases - which only enter
as a side effect of the inclusion of library
modules anyway. Global variables are bad, and
will may well prevent people using more than one
instance of your PLUGIN.

1.8 Style Guide (3)

3. Tags

This style guide defines a special form for the
tags to be used by the PLUGINs. (Advantages of the
use of tags, while somewhat outside the scope of this
document, include clarity of code and the enforcement

of certain consistencies of interface.)

The prefix PLA_ (PLugin Attribute) is used for tags which modify attributes of the object and PLV_ (PLugin Value) for special values for attributes. The prefix is now followed by the object name and then by a special name for the tag e.g. attribute name.

```
PLA_Xyz_<AttrName>
```

The first letter of every word is uppercase, in the same way as the standard library calls.

e.g.

```
PLA_Text_Color
PLA_Bar_Hidden
PLA_BigGadget_Contents
```

The tag base to be used for these tags should be calculated in the following way:

```
TAG_USER OR Shl(PROGRAMMER_ID, 24) OR
Shl(PLUGIN_ID, 16) OR (<Number of tag>)
```

where

TAG_USER is as defined in <utility/tagitem.h>, $80000000.

PROGRAMMER_ID is your assigned ID - you will be given one on application to the coordinator of EasyPLUGINs. If you don't have an ID assigned yet, use the code for a private module, $7F.

PLUGIN_ID is a number decided by you, between $00 and $FF. Make sure this number does not conflict with any of your other publically released modules.

Number of Tag (just iterate from 0 or 1 using ENUM, not forgetting to EXPORT these ENUM'd tags as they serve as the external interface to your PLUGIN.)

For example, say that I have decided to code a PLUGIN that draws a scaleable file-gadget image. I would start by calculating the tag-base - assume that I am the 53rd registered EasyPLUGINs module programmer (I wish :) and that my PROGRAMMER_ID was therefore $34 (counting in hex and from 0 like all good programmers...) and that this is the 5th PLUGIN that I am writing.

The tag base for this PLUGIN would then be: $B4050000

The tag base is best understood when segmented, like this:

```
$ [B4] [05] [0000]
```

```
        ^^   ^^   ^^^^
        |    |    |
        |    |    |___ (wil be incremented for each tag)
        |    |
        |    |___ ($050000, for the fifth PLUGIN)
        |
        |___ ($80000000 OR $34000000)
```

1.9 Style Guide (4)

```
4.  Public interface


Now to move on to the standard methods. Each PLUGIN
should have three or four standard methods: a constructor
(and often a destructor), and methods to set() and get()
certain values in order to control the PLUGIN's behaviour.

The constructor should have the same name as the PLUGIN
object, for similarity to C++ semantics; i.e. a PLUGIN
called vector_plugin would have a constructor method
named vector(). This method should only take one argument,
a tag list.

Thus, the constructor for this hypothetical vector_plugin
would be defined like this:

    PROC vector(tags:PTR TO tagitem) OF vector_plugin

        ....

    ENDPROC

Inside the constructor, the initial action should be the reading
of the tags provided in the tag list by the user of the PLUGIN:

    PROC vector(tags:PTR TO tagitem) OF vector_plugin

        IF utilitybase

            self.colour  :=  GetTagData(PLA_Vector_Colour, 1, tags)
            self.disabled  := GetTagData(PLA_Vector_Disabled, FALSE, tags)

        ELSE

            Raise("util")

        ENDIF

        -> any other necessary resource allocation

    ENDPROC

You may want to open and close the utility.library inside
```

```
the constructor; this is really a matter of personal choice.
However, it is easiest to only open this once per project;
to have, for example, five PLUGINs all opening the library
is a waste of executable size.
(Remember, utilitybase is defined in the 'utility.m' module,
so you'll need to include that in the MODULE definitions at
the start of your PLUGIN module.)


The PLUGIN should also have set() & get() methods in order to
change and read the values inside the PLUGIN. In order to avoid
a myriad number of setxxx() and getyyy() PROCs, like in the
original EasyGUI :), these should be generalized to one method
each:

e.g.

    PROC set(attr, value) OF vector_plugin

        SELECT attr

            CASE PLA_Vector_Colour

                self.colour:=value

                -> call the draw() method to redraw

            CASE PLA_Vector_Disabled

                self.disabled:=value

                -> call the draw() method to redraw

        ENDSELECT

    ENDPROC


and the get() method works in a similar way:

e.g.

    PROC get(attr) OF xyz

        SELECT attr

            CASE PLA_Xyz_Colour;     RETURN self.colour, TRUE
            CASE PLA_Xyz_Disabled;   RETURN self.disabled, TRUE

        ENDSELECT

    ENDPROC -1, FALSE

The get() method has 2 return values. The first one is
the requested value and the second one indicates whether
the attribute is gettable or not. This may not really be
necessary, but it is probably a handy convention to
```

```
introduce now in case sub-classing of PLUGINs becomes
endemic...


The destructor, as is always the case in E's OOP
structures, must be called end() and take no arguments.
In this you can free any structures or resources allocated
in the constructor.

    PROC end() OF vector_plugin

        ....

    ENDPROC
```

1.10 Style Guide (5)

```
                5.  Implementation


This basically involves the render() methods, the message
testing methods and any private methods you may want the
PLUGIN to use. As these methods are not a part of the public
interface of your PLUGIN, as the methods discussed in the

            last section
             were, this section can therefore be taken as
an advisory one; you can diverge from the standards shown
here (except, of course, the ones set by the EasyGUI docs
themselves) while still earning the EasyPLUGIN's Style
Guide Seal of Quality®.

If your PLUGIN does a lot of drawing, or has a number of
various paths it can take in the course of its rendering,
you may want to split up the render() or gtrender() method;
your method would then look like this:

    PROC [gt]render(...) OF sample_plugin

       DEF oldwin

       self.ta :=ta         /* save ta for use in other methods (if needed)*/

       /* this part is necessary  because this function behaves not like the
          other ones and the self.gh.wnd is not correctly set */
       oldwin:=self.gh.wnd  /* at this time wnd is 0 but it may change in a  ↩
          newer version*/
       self.gh.wnd:=win
       self.draw(TRUE)      /* full redraw */
       self.gh.wnd:=oldwin

    ENDPROC

and the draw() method would look like this:
```

```
    PROC draw(redraw=FALSE) OF sample_plugin

        IF self.gh=0 THEN RETURN  /* don't draw when gui isn't initialized */
        IF self.gh.wnd=0 THEN RETURN  /* ignore draw when window is closed */

        -> ...
        -> here you can do your drawing stuff
        -> and it's possible to select between update and full redraw -
        -> if your PLUGIN can be disabled, you may want to use the
        -> functions from the module 'tools/ghost'
        -> ...

    ENDPROC
```

```
The advantage of this is that the draw() method can then be
easily called from, for example, the set() method of your
PLUGIN.

If you don't need this then just use the render() or
gtrender() method in the normal fashion.

As to the message handling methods, message_test() and
message_action(), there are no special rules in this
Style Guide concerning them; just obey the suggestions
made in EasyGUI.doc, and make sure that the message_test()
method executes very quickly.
```

1.11 Style Guide (6)

```
6.  Testing, debugging and documentation


A minor but necessary part of the process; adapt an
example program to use your new PLUGIN and make sure
it all works as advertised.

If not, get the bugs out and try again; repeat
ad nauseam until the PLUGIN is working well.

Then it's time to sit down and actually document the
PLUGIN; because of the strictly followed structure of
this Style Guide, this should not be too difficult,
as you should have a good idea of what you have done
and the manner in which your PLUGIN operates.
```

1.12 Works in Progress

```
This section has been included in the documentation as
a public noticeboard; here authors of PLUGINs can announce
their current projects, and ask for feedback or suggestions
from other authors and from the general public.
```

The PLUGINs which are currently being worked on are:

```
+- corners plugin ----------------------------------------+
|                                                         |
| This is a PLUGIN which enables you to choose between    |
| the four corners of a square (envisaged usage; screen   |
| blanker preferences program). While not immediately     |
| useful to the general public, it will illustrate the    |
| usage of detecting input in different areas of the      |
| PLUGIN.                                                 |
+---------------------+-----------------------------------+
|Ali Graham           |
|agraham@hal9000.net.au|
+---------------------/

+- newlistview plugin -----------------------------------+
|                                                       |
| It's just an idea and I only started to program the   |
| basics of this class. I planned to create a  multi    |
| column lv with horizontal scrolling and PC like titles. |
| It should provide sorting and resizing of columns.    |
| But as I said I just started this plugin and I need a |
| lot of suggestions.                                   |
+-------------------------------+-----------------------+
|Ralph Wermke                   |
|wermke@gryps1.uni-greifswald.de|
+-------------------------------/

+- popup plugin -----------------------------------------+
|                                                       |
| It supports pop-up menus simular in purpose to cycle  |
| gadgets except that the full list pops up.            |
| A beta version will be available soon.                |
+-------------------------------+-----------------------+
|Ralph Wermke                   |
|wermke@gryps1.uni-greifswald.de|
+-------------------------------/

+- updown plugin ----------------------------------------+
|                                                       |
| An integer gadget with arrows for increasing/decreasing |
| the value and a settable range.                       |
+-------------------------------+-----------------------+
|Ralph Wermke                   |
|wermke@gryps1.uni-greifswald.de|
+-------------------------------/
```

1.13 Requests

[While this part is empty now, if anyone has a PLUGIN they would like to see written, if they send that request to me I'll place it here. If anyone can do it, I'm sure they can

```
  volunteer, and I'll then move it over into the

            Works in Progress
             section. ]
```

1.14 History

```
                v1.0                                    (30.8.97)

o   Initial Aminet release, with six PLUGINs:

  basicgauge_plugin:    Ali Graham        <agraham@hal9000.net.au>
  bitmapimage_plugin:   Daniel Westerberg <deniil@algonet.se>
  iconbox_plugin:       Ali Graham        <agraham@hal9000.net.au>
  imagebutton_plugin:   Ali Graham        <agraham@hal9000.net.au>
  title_plugin:         Ali Graham        <agraham@hal9000.net.au>
  xygadget_plugin:      Ali Graham        <agraham@hal9000.net.au>


v1.1                                  (2.10.97)

o   Added four new PLUGINs:

  dclistview_plugin:    Victor Ducedre     <victord@netrover.com>
  register_plugin:      Ralph Wermke       <wermke@gryps1.rz.uni-greifswald.de ↩
     >
  simplegauge_plugin:   Ralph Wermke       <wermke@gryps1.rz.uni-greifswald.de ↩
     >
  text_plugin:          Ali Graham         <agraham@hal9000.net.au>

o   Changes made to four PLUGINs:

  basicgauge_plugin:    Ali Graham         <agraham@hal9000.net.au>
  iconbox_plugin:       Ali Graham         <agraham@hal9000.net.au>
  imagebutton_plugin:   Ali Graham         <agraham@hal9000.net.au>
  xygadget_plugin:      Ali Graham         <agraham@hal9000.net.au>


v1.2                                  (28.10.97)

o   Forgot to make some minor changes to some of the examples which
    they would not compile without. Now fixed.

o   Added one new PLUGIN:

  rawkey_plugin:        Fabio Rotondo      <fsoft@intercom.it>

o   Changes made to five PLUGINs:

  bitmapimage_plugin:   Daniel Westerberg  <deniil@algonet.se>
  iconbox_plugin:       Ali Graham         <agraham@hal9000.net.au>
  imagebutton_plugin:   Ali Graham         <agraham@hal9000.net.au>
  text_plugin:          Ali Graham         <agraham@hal9000.net.au>
  title_plugin:         Ali Graham         <agraham@hal9000.net.au>
```

```
v1.3                                    (28.11.97)

o   Added
            Style Guide
            ,
            Works in Progress
            ,
            Tag Bases
             and

            Requests
             sections to the documentation.

o   Added German installation strings, which will automatically be used if
    the system is localized to use the language (for some reason, forcing it
    through the tooltype LANGUAGE=deutsch doesn't work at the moment,
    despite what the docs for the Installer say.).

  Thanks to Ralph Wermke for his invaluable work on the above two items!

o   Added three new PLUGINs:

  popasl_plugin:        Ralph Wermke       <wermke@gryps1.rz.uni-greifswald.de ↩
     >
  icongad_plugin:       Fabio Rotondo      <fsoft@intercom.it>
  toolbar_plugin:       Ali Graham         <agraham@hal9000.net.au>


o   Removed three PLUGINs:

  basicgauge_plugin:    Ali Graham         <agraham@hal9000.net.au>
    (too similar to simplegauge_plugin)
  imagebutton_plugin:   Ali Graham         <agraham@hal9000.net.au>
    (too similar to an example in the EasyGUI package)
  title_plugin:         Ali Graham         <agraham@hal9000.net.au>
    (functionality subsumed into text_plugin)


o   Changes made to five PLUGINs:

  iconbox_plugin:       Ali Graham         <agraham@hal9000.net.au>
  register_plugin:      Ralph Wermke       <wermke@gryps1.rz.uni-greifswald.de ↩
     >
  simplegauge_plugin:   Ralph Wermke       <wermke@gryps1.rz.uni-greifswald.de ↩
     >
  text_plugin:          Ali Graham         <agraham@hal9000.net.au>
  xygadget_plugin:      Ali Graham         <agraham@hal9000.net.au>
```

1.15 Submissions

```
This is the general procedure for submissions:
```

```
- You send me the plugin that you would like in the
  package; source code, compiled module and a small
  amount of documentation on how to use the methods
  of the PLUGIN (which I'll weave into this
  documentation in the appropriate place).

- You retain copyright over the PLUGIN; you are listed
  in the documentation as the author of the software,
  and people are encouraged to contact you with changes
  to be included in the next release.

- Source code is provided to the modules for two reasons;
  future-proof-ness (i.e. if the general EasyGUI PLUGIN
  definitions change somewhat, people can change the source
  and recompile), and the fact that people may want to
  change the way the PLUGINs operate before using them.
  I prefer to include source code for the modules, but I
  do not insist on it -- however, you should be willing
  to pass the source on to someone else if you have no
  further desire to support or improve the PLUGIN.

If anybody wants to release a module in this package under
conditions differing from those above, contact me and I'll
see if we can come to some arrangement.
```

1.16 Authors

```
The general coordinator for the EasyPLUGINs package is

    Ali Graham <agraham@hal9000.net.au>

Contact me concerning general organizational issues about
this package, or about any of the PLUGINs that I've written.
In general, though, if you have questions/changes/whatever
about a PLUGIN, these should be addressed to the relevant
authors:


bitmapimage_plugin    Daniel Westerberg <deniil@algonet.se>
dclistview_plugin:    Victor Ducedre    <victord@netrover.com>
iconbox_plugin:       Ali Graham        <agraham@hal9000.net.au>
icongad_plugin:       Fabio Rotondo     <fsoft@intercom.it>
popasl_plugin:        Ralph Wermke      <wermke@gryps1.rz.uni-greifswald.de>
rawkey_plugin:        Fabio Rotondo     <fsoft@intercom.it>
register_plugin:      Ralph Wermke      <wermke@gryps1.rz.uni-greifswald.de>
simplegauge_plugin:   Ralph Wermke      <wermke@gryps1.rz.uni-greifswald.de>
text_plugin:          Ali Graham        <agraham@hal9000.net.au>
toolbar_plugin:       Ali Graham        <agraham@hal9000.net.au>
xygadget_plugin:      Ali Graham        <agraham@hal9000.net.au>


Some (most? all?) of these people are subscribed to the AmigaE mailing
list; you can subscribe to this list by sending a message to

    amigae-list@intercom.it
```

```
with the word SUBSCRIBE in the subject line.

To unsubscribe, send a message to the above address with the word
UNSUBSCRIBE in the subject line.
```