

EasyPLUGINS

COLLABORATORS			
	TITLE : EasyPLUGINS		
ACTION	NAME	DATE	SIGNATURE
WRITTEN BY		August 22, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1	EasyPLUGINS	1
1.1	EasyPLUGINS	1
1.2	Introduction	1
1.3	Registered tag bases and programmer ID's	2
1.4	Style Guide	3
1.5	Style Guide (1)	4
1.6	Style Guide (2)	4
1.7	Style Guide (3)	5
1.8	Style Guide (4)	6
1.9	Style Guide (5)	9
1.10	Style Guide (6)	10
1.11	Works in Progress	11
1.12	Requests	12
1.13	History: Contents	12
1.14	History: v1.0	12
1.15	History: v1.1	13
1.16	History: v1.2	13
1.17	History: v1.3	13
1.18	History: v1.4	14
1.19	History: v1.5	15
1.20	History: v1.6	15
1.21	History: v1.7	16
1.22	Submissions	16
1.23	Authors	17

Chapter 1

EasyPLUGINS

1.1 EasyPLUGINS

EasyPLUGINS v1.7 (25.3.98)

Introduction

Authors

History

Submissions

Style Guide

Works in Progress

Registered tag bases

Requests

	[s..]	bitmapimage_plugin
	[seg]	corners_plugin
P	[seg]	dclistview_plugin
	[seg]	dtpic_plugin
L	[seg]	iconbox_plugin
	[seg]	icongad_plugin
U	[se.]	location_plugin
	[seg]	multitext_plugin
G	[seg]	popasl_plugin
	[seg]	popmisc_plugin
I	[se.]	rawkey_plugin
	[seg]	register_plugin
N	[seg]	simplegauge_plugin
	[seg]	space_plugin
S	[seg]	textfield_plugin
	[seg]	toolbar_plugin
	[seg]	xygadget_plugin

(s) - PLUGIN comes with source code.

(e) - PLUGIN comes with example/s of usage.

(g) - PLUGIN is EasyPLUGINS Style Guide Compliant.

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; most of them include documentation. Some also come with example sources and/or the source code for the modules themselves.

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...)

(This should be taken as a general policy for all PLUGINS for which the documentation does not specify any differently. Check the documentation for any individual PLUGIN you are interested in to see if the legal status of that PLUGIN is different from the default status outlined above.)

1.3 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
textfield_plugin:	\$08

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

Registered PLUGIN ID's:

iconbox_plugin:	\$01
multitext_plugin:	\$02
toolbar_plugin:	\$03
xygadget_plugin:	\$04
space_plugin:	\$05
popmisc_plugin:	\$06
corners_plugin:	\$07

Will Harwood \$02
<147800.97@swansea.ac.uk>

Fabio Rotondo \$03
<fsoft@intercom.it>

(Currently unused) \$04
-> \$7E

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

1.4 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

1.5 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.

1.6 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.

Another suggestion is to define a `CONST`, equal to the value of `PLUGIN`, with an uppercase version of your PLUGIN's name (with `EP_` prepended to avoid any compilation clashes and the like.

e.g. `sample_plugin` would have

```
CONST EP_SAMPLE=PLUGIN
```

and could then be added to an EasyGUI with the line

```
[EP_SAMPLE, {sample_action}, sample]
```

1.7 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)
```

1.8 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
```

1.9 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.

Many PLUGINS will offer the `AppWindow` method, and this is a good basic form for it to take:

```
PROC appmessage(ams:PTR TO appmessage, win:PTR TO window) OF sample_plugin
```

```
    DEF xok=FALSE, yok=FALSE
```

```
    xok:=(ams.mousex>=(self.x)) AND (ams.mousex<=(self.x+self.xs-1))
    yok:=(ams.mousey>=(self.y)) AND (ams.mousey<=(self.y+self.ys-1))
```

```
ENDPROC ((xok AND yok) AND (self.disabled=FALSE))
```

(Of course, if your PLUGIN does not implement being disabled/enabled then just leave that part out of the final return-value).

1.10 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.

Also, consider some brief documentation in the sources and examples themselves, as well as commenting the sources well. Always include your name and e-mail address somewhere in the sources, as it makes it much easier for someone to contact you if that particular PLUGIN source has become separated from the package as a whole.

I suggest that you use the following template as a way of describing your PLUGINS, and use it in each one you write:

```
$VER: <plugin_name> <plugin_version> (<plugin_date>)
```

Author: <your name> (<your EPSG programmer id>)
 <your e-mail address>

```
PLUGIN id:    <EPSG plugin id>
```

Desc.: <brief description of the PLUGIN>

The following two sections are optional:

Tags: <list of supported PLA_... tags, including [ISG] status>

Values: <list of supported PLV... values>

History: <sequential list of the changes to the PLUGIN>

Examples for your plugins should contain, as a minimum, the \$VER:, Author and Desc. fields.

1.11 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:

```
+-----+
| 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|
+-----/
```

1.12 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.]

1.13 History: Contents

```
\
  v1.0   (30.8.97)
/
  v1.1   (2.10.97)
\
  v1.2   (28.10.97)
/
  v1.3   (28.11.97)
\
  v1.4   (30.1.98)
/
  v1.5   (18.2.98)
\
  v1.6   (25.2.98)
/
  v1.7   (25.3.98)
\
```

1.14 History: v1.0

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>

```

1.15 History: v1.1

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>

```

1.16 History: v1.2

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>

```

1.17 History: v1.3

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>
```

1.18 History: v1.4

v1.4 (30.1.98)

- o Sorry about the delay - life gets in the way sometimes, especially around Christmas :(
- o Added another registered programmer.
- o Split the install scripts up by language.
- o Added =NIL as a default argument for the constructors.
- o Added a beta version of epSG, the EasyPLUGINS StyleGuide SkeletonGenerator (maybe ep(SG)\$^2\$ would be a better name :) Thanks to Ralph Wermke for writing this! Send him feedback if you want to see changes or improvements made to this program...
- o Added one new PLUGIN:

```
space_plugin:      Ali Graham      <agraham@hal9000.net.au>
```

- o Changes made to three PLUGINS:

```
dclistview_plugin: Victor Ducedre    <victord@netrover.com>
iconbox_plugin:    Ali Graham      <agraham@hal9000.net.au>
popasl_plugin:     Ralph Wermke    <wermke@gryps1.rz.uni-greifswald.de ↵
>
```

1.19 History: v1.5

v1.5 (18.2.98)

- o Added a few things to the Style Guide – suggestions on including an EP_... CONST and on the appmessage() method, and a brief mention of the importance of comments and documentation.
- o Corrected some documentation bugs (even here in the history! :)
- o Added five new PLUGINS:

```
corners_plugin:    Ali Graham      <agraham@hal9000.net.au>
dtpic_plugin:      Will Harwood    <147800.97@swansea.ac.uk>
location_plugin:   Ali Graham      <agraham@hal9000.net.au>
poptask_plugin:    Ali Graham      <agraham@hal9000.net.au>
textfield_plugin:  Ralph Wermke    <wermke@gryps1.rz.uni-greifswald.de ↵
>
```

- o Changes made to three PLUGINS:

```
dclistview_plugin: Victor Ducedre    <victord@netrover.com>
icongad_plugin:    Fabio Rotondo    <fsoft@intercom.it>
popasl_plugin:     Ralph Wermke    <wermke@gryps1.rz.uni-greifswald.de ↵
>
```

1.20 History: v1.6

v1.6 (25.2.98)

- o Added a template for intra-source description of PLUGINS in the last section of the Style Guide.
- o Added a '.build' script for EBuild in the examples directory; since the compiled demos add up to over 500K, I felt this was a healthier alternative than including them.
- o Oops! Left out dclistview_demo.e. It's there now :).
- o Changes made to three PLUGINS:

```
corners_plugin:      Ali Graham      <agraham@hal9000.net.au>
multitext_plugin:    Ali Graham      <agraham@hal9000.net.au>
    (name changed from text_plugin - increased functionality)
popmisc_plugin:      Ali Graham      <agraham@hal9000.net.au>
    (name changed from poptask_plugin - increased functionality)
```

1.21 History: v1.7

v1.7 (25.3.98)

- o Split up the history in the documentation, as I'm sure you'll have noticed by now :)
- o Changed the .build script into one that will compile all examples, all sources or both -- have a look at it to see what I mean. (It now resides in the main directory.)
- o Changed the information about subscribing to the AmigaE mailing list in the authors section.
- o Changes made to four PLUGINS:

```
dclistview_plugin:   Victor Ducedre   <victord@netrover.com>
multitext_plugin:    Ali Graham      <agraham@hal9000.net.au>
space_plugin:        Ali Graham      <agraham@hal9000.net.au>
toolbar_plugin:      Ali Graham      <agraham@hal9000.net.au>
```

1.22 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.

1.23 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>
corners_plugin:	Ali Graham	<agraham@hal9000.net.au>
dclistview_plugin:	Victor Ducetre	<victord@netrover.com>
dtpic_plugin:	Will Harwood	<147800.97@swansea.ac.uk>
iconbox_plugin:	Ali Graham	<agraham@hal9000.net.au>
icongad_plugin:	Fabio Rotondo	<fsoft@intercom.it>
location_plugin:	Ali Graham	<agraham@hal9000.net.au>
multitext_plugin:	Ali Graham	<agraham@hal9000.net.au>
popasl_plugin:	Ralph Wermke	<wermke@gryps1.rz.uni-greifswald.de>
popmisc_plugin:	Ali Graham	<agraham@hal9000.net.au>
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>
space_plugin:	Ali Graham	<agraham@hal9000.net.au>
textfield_plugin:	Ralph Wermke	<wermke@gryps1.rz.uni-greifswald.de>
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

listserv@intercom.it

with SUBSCRIBE amigae-list <first-name> <last-name> in the subject line.

i.e. I would send a message with SUBSCRIBE amigae-list Ali Graham if I wanted to subscribe to the list.

To unsubscribe, send a message to the above address with UNSUBSCRIBE amigae-list <first-name> <last-name> in the subject line.