Multiformatinputfilesorrydoesn’tknowthisargumentisnotusers

Carsten Jahn

COLLABORATORS			
	TITLE : Multiformatinputfilesorrydoesn’tknowthisargumentisnotusersfault.		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Carsten Jahn	August 26, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Multiformatinputfilesorrydoesn’tknowthisargument

1.1 Madhouse Developer Guide

```
                Madhouse Developer Guide -- Preface

It is not very difficult to program your own Madhouse blanker. It would
be great if some people really do so!

All a blanker does is opening the madblankersupport.library and calling
the support functions. The library is explained in an AutoDoc file in
this archive.

Apart from that, MadhouseConfigEd needs a "gadget" description file which
is used to arrange the gadgets of the blanker's window. (Yes, this work
is done completely by Madhouse!) Here, almost every gadget gets a
descriptive name (e.g. "speed" for a speed slider), the blanker queries
it's prefs by just asking for "speed" in a madblankersupport.library
function.
Another purpose of the gadget file is to submit more information about
the blanker. What category, strong CPU usage? MadhouseConfigEd reads this
information when you press "Update" on the ConfigEd's System page and
stores them together with other settings in ENV[ARC]:Madhouse.prefs.

So the purpose of this doc file is to show you how to write a gadget
file. You have lots of examples, every Madhouse blanker has this file in
its subdir, imported blankers get their gadget files created by Madhou-
seConfigEd. If you have both blanker executable (with no MBS_DebugMode()
call in it, remember that... see AutoDoc) and gadget file, create a new
subdirectory in your blankers directory and press "Update" on the Confi-
gEd's System page. Your blanker should be in the list of available
blankers now. If you would like to see your blanker included with Madh-
ouse, please send it to me! As long as it is not a remake of a simple
blanker, we will be happy to include it in the next version. And please
feel free to ask questions.

Yours sincerely,
Carsten Jahn

(January 1998)
```

Now follows the gadget file explanation text, done for Madhouse version 2.0 and partly version 1.1, briefly edited to wipe out obsolete instructions for the GadTools interface of MadhouseConfigEd (which doesn’t exist any more). And I deleted the old "prefs" file structure explanations, as preferences are now read by madblankersupport.library. It was checked for spelling and corrected by Aaron Roberts in January 1997. Thanks again, Aaron!

The gadget file

1.2 The gadget file

You should already know that "gadgets" are the little boxes, where you move the mouse cursor and click with the left mousebutton...

Since Madhouse needs MUI to run (this version and the upcoming ones...), I won’t describe the commands which open a normal Intuition window again. This won’t make any sense, because Madhouse doesn’t look at this part of the gadget file anyway. (But you can still enter them without getting errors - as you can enter anything in between the chunks.)

As an example, this is the Snow gadget file.

```
Madhouse v1.0, BlankerPrefs-Window

CHUNK:LOCALE
english,deutsch

CHUNK:MUI-WINDOWLAYOUT
ColumnGroup(2),
  Label( "Co_llision,Ko_llision" ),
  Cycle( "l", "Windowborders|Screen image,Fensterkanten|Bildschirminhalt" ),
End,

CHUNK:BLANKERINFO
Carsten Jahn
1.25 (06.11.1997)
1
8000
2
Madhouse2
WB
```

You can see:

- The first line includes text which MUST be in the first line. Madhouse recognizes its gadget files with this line.
- The gadget file is organized in Chunks. This means that every Chunk has a Chunk-Header which identifies it (for example: CHUNK:BLANKERINFO). If the Chunk-Header is unknown, Madhouse skips it.

o You may only insert blank lines between two complete chunks. The Chunks are case sensitive.

Before I continue explaining every chunk in detail, here's a short overview:

o The CHUNK:MUI-WINDOWLAYOUT is the most important one, it defines your BlankerPrefs window.
o The CHUNK:BLANKERINFO is not needed for the window itself. The MadhouseConfigEd reads it while reading the whole blankers directory, and stores the information in Madhouse.prefs, where 'Madhouse' can reach it. It contains information about the CPU loading of the blanker, the stack used, etc.
o The CHUNK:LOCALE is for the locale support (used if your OS is 2.1 or better). If it does exist in your gadget file, you can seperate the languages in CHUNK:MUI-WINDOWLAYOUT by commas.

And now for the explanation of the chunks in detail. '*' means that the chunk is optional, and doesn't have to appear in your file.

CHUNK:BLANKERINFO

* CHUNK:LOCALE
In the next chapters, I will explain the usage of the chunk for ↩ MUI-
windows. I hope everyone understands my explanations, though I will write them very quickly (I want to FINISH this english doc). If you don't, please look into the MUI developer kit (available as FD). But I think you will get that. So start the work:

MUI for MUI-Greenhorns

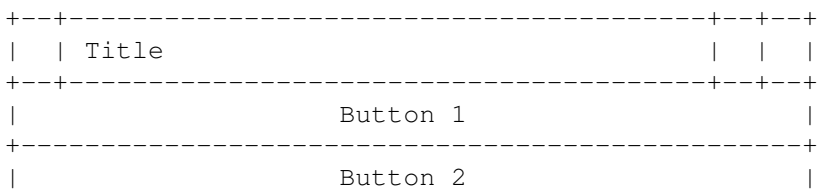
The MUI Groups

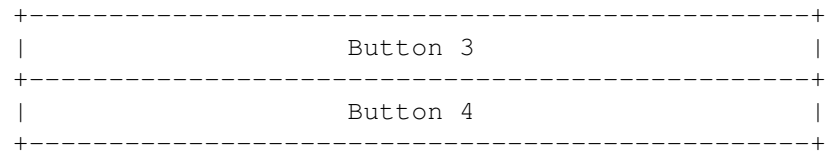
The MUI Gadgets

1.3 for MUI-Greenhorns

Creating an MUI-windowlayout is very different. So stop ↩ thinking about
pixels and fonts and start thinking about the basic layout of windows.

Here you see a window:





What do you see there? Notice how the buttons are grouped vertically.

If the window from above was made by MUI, the programmer would only need to tell MUI to build a vertical group and give it these four gadgets. MUI would calculate the positions and sizes of the gadgets - very easy.

Perhaps you already know what comes next: horizontal groups containing some gadgets. So we have vertical and horizontal groups and gadgets (of course not just buttons but different types of gadgets, that makes no difference at the moment). But a real window layout is much more complicated. Now to the point: Groups can contain more than gadgets, they can also contain new groups! It’s the same thing with files and directories: a directory can contain files and directories containing even more files and directories.

< Think a bit. >

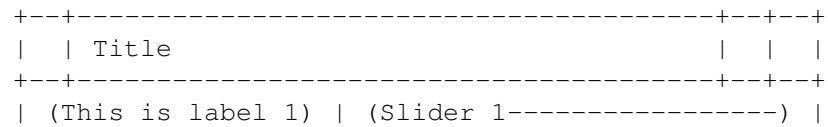
So how does your window look if we replace a horizontal group with two gadgets for button 3? If you think you know the answer, let’s see if you are right:

.

But for real windows we need more gadget types. And these other types have a box containing the gadget itself, like our button, but they also have a text in front of them, like the ‘Exchange Blankers’ checkmark and all the others have one. We call these texts "labels".

Usually, you define two objects for every gadget: the label and the gadget itself. Madhouse has some short forms of gadget definitions containing a label on the left side of the gadget, but you use these forms very seldomly.

The reason for that is that all MUI-objects (gadgets, labels [and groups]) have some stretching-limitations. You can make a slider as long as you want, but it has a fixed height. This is also true for buttons, stringgadgets and cycle gadgets. Some objects cannot be stretched at all: labels (containing their text with a fixed width and height) and checkmarks. Listviews can be stretched in all directions. So if you use the short form of a slider for three sliders in a vertical group, to build three sliders over another with descriptions on their left sides; you will get these sliders. The problem is the left edges of the sliders won’t match at all. The window will look like:



```
+-------------------------------------------------+
| (label 2) | (Slider 2-----------------------) |
+-------------------------------------------------+
| (and label 3) | (Slider 3-------------------) |
+-------------------------------------------------+
```

(Note that I marked the dimensions of the objects like that: (-----).)

This does not look very nice. This happens because the labels have a fixed width and the sliders are used to fill the box. In this case, you need column groups. Unlike the other groups, column groups have an argument: the amount of columns. So if you create a column group with three columns, containing 6 Buttons, you will get something like this:

```
+--+-----------------------------------------------+--+--+
|  | Title                                         |  |  |
+--+-----------------------------------------------+--+--+
|    Button 1    |    Button 2    |    Larger Button 3   |
+-------------------------------------------------------+
|    Button 4    |    Button 5    |       Button 6       |
+-------------------------------------------------------+
```

Even if the text inside the buttons has a different lenght, MUI will group them in a way that every part of one column has the same width.

But we need a solution for our problem above, with the sliders. To have the same width for all sliders, we make a column group with two columns, containing a label, a slider, a label, a slider, a label and a slider (in that order). Then we get

```
+--+-----------------------------------------+--+--+
|  | Title                                   |  |  |
+--+-----------------------------------------+--+--+
| (This is label 1) | (Slider 1----------------) |
+-------------------------------------------------+
| (        label 2) | (Slider 2----------------) |
+-------------------------------------------------+
| (    and label 3) | (Slider 3----------------) |
+-------------------------------------------------+
```

That’s much better! But how does this look in the gadget-file? First, I will show you what our first example looks like:

```
CHUNK:MUI-WINDOWLAYOUT
VGroup,
   Button1,
   Button2,
   Button3,
   Button4,
End,
```

In our new chunk "MUI-WINDOWLAYOUT" we put a vertical group. This group

reaches up to "End". It’s the same with IF and ENDIF in a programming language: every ENDIF matches to one IF. Inside the group, we write the objects which are associated with it, in this case four buttons. Every line ends with a "," - but you are not allowed to merge two lines to one! You don’t have to "indent" the lines like I did, but you should do so because it’s easier to read. You can use spaces or TAB’s for that.

And how about our column group from above? For didactical purposes, I will use a cyclegadget instead of the third slider.

```
CHUNK:MUI-WINDOWLAYOUT
ColumnGroup( 2 ),
   Label( "_Label 1" ),
   Slider( "l", 1, 10 ),
   Label( "L_abel 2" ),
   Slider( "a", -5, 20 ),
   Label( "La_bel 3" ),
   Cycle( "b", "First entry|Second entry|Third entry" ),
End,
```

Now the chunk contains a column group, as we need one for the correct formatting of the labels. The "(2)" makes two column groups. A column group has to be filled up to the right-most column. In this case, the group can have 2, 4, 6, 8, ... "children". Child is the MUI-word for an associated object. If a column group has five columns, it can respectively have 5, 10, 15 and so on children. If you don’t want to fill every place in a column group, there is a "dummy" object for that, which I will explain later.

The last example is one which could be written into a gadget-file without changes, you can see the the definitions for labels, sliders and cycle-gadgets. A label is done by the keyword "Label", including the text of the label in this form: Label("Text"),. You can use an underscore _ to underline the following char. This is used to show the user which key of the keyboard can be pressed to control the gadget.

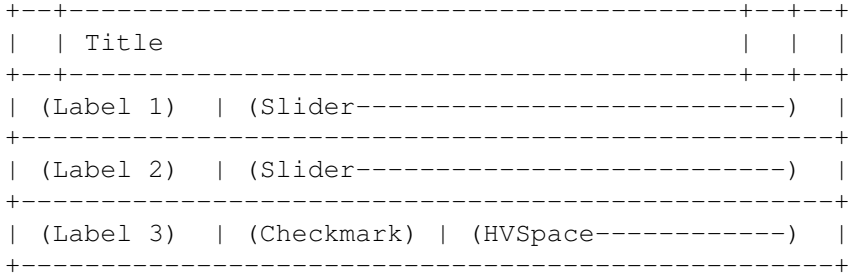
A slider comes next: this type of object needs three arguments: the control key (which should be marked in the label before), the minimum and the maximum value. The first example produces a slider which ranges from 1 to 10. You can also use negative values, see the second slider. The cycle object needs two parameters: the control key, which is always the first parameter of every gadget, and the different cycle entries, separated with "|". On my german keyboard, I can find this char next to the backspace key.

Please note that you cannot use every control key you want: you can only use non-capital letters and you cannot use the chars o, t and c, because they are already used by the three buttons on the bottom of every BlankerPrefs-window.
Now an example where we need to put one group into another. Please wait patiently while I’m explaining the situation. (just building up the problem for my solution :-/)

Perhaps (oh, I’m sure...) you have noticed that MUI arranges the gadgets automatically (that is why we don’t need pixel-coordinates) and dynamically. This means that almost every MUI window has a size gadget, which

allows us to size it, and MUI recalculates the gadget positions so everything fits into the window again. But have you noticed that you can size a lot of MUI windows only horizontally, the height is fixed? (Try some BlankerPrefs windows). Can you imagine why this happens? Every MUI-object (gadgets and groups, but we’ll start with gadgets) has minimums and maximums for width and height. Sliders, string gadgets cycle gadgets and some more have a fixed height (minimum height = maximum height), but no fixed width (maximum width is VERY big). To scale the height of a cycle gadget would make the cycle gadget look silly. Lists have neither a fixed height nor a fixed width, checkmarks have both. So we have answered half of our question: you can size a lot of MUI windows only horizontally, because a lot of windows contain objects with a fixed height.
Now for the problem. Try to replace the cycle gadget by a checkmark (see our last example, slider - slider - cycle). Now, the result looks quite strange: as the two sliders and the checkmark are placed in the same column, MUI wants to give them the same width. Since MUI fails in scaling the checkmark horizontally, it gives the sliders the width of the checkmark, which is of course too small for a slider. The window has no size gadget at all, because the group inside the window cannot be sized (the left column is blocked by the labels, the right by the checkmark, both objects with fixed width). We have to help MUI, and make the checkmark bigger. (This is ... impossible.) We could replace the checkmark by a horizontal group, containing a checkmark and a slider. The window could then be sized horizontally again, because the column group can be sized again. Since it has a column (the right one) which can be sized, all the members in this column can be sized. Even the horizontal group (containing a checkmark and a slider) can be sized, because this group has one object (the slider) that can be sized. Uff... "can be sized" means here "can be sized horizontally", MUI does the same thing for vertical sizing.

- Hey, my window looks stupid with all these sliders which I filled in after I’ve heard your explanation! - No problem, MUI offers another solution for our problem. It has an object that can be sized to unlimited dimensions and which is... invisible. We call it HVSpace. Our little ANSI-graphic - AmigaGuide has no drawing tools :-(is here:



And you code this:

```
CHUNK:MUI-WINDOWLAYOUT
ColumnGroup(2),
   Label( "_Label 1" ),
   Slider( "l", 1, 10 ),
   Label( "L_abel 2" ),
```

```
   Slider( "a", -5, 20 ),
   Label( "La_bel 3" ),
   HGroup,
      CheckMark( "b" ),
      HVSpace,
   End,
End,
```

I hope you get it. The HVSpace is also useful in a second case: if you have a big columngroup and do not want to put objects in all cells, you can assign the HVSpace to the cell which should be empty.

To get further impressions of MUI-layout, you can look into the gadget files. This was not the whole MUI documentation for Madhouse, just the basic knowledge. All types of gadgets and another feature of the groups can be found in the next chapters.

Identifiers -- connecting gadgets to madblankersupport.library

madblankersupport.library requires gadget identifiers to give you a prefs item, e.g. "cycle speed" for a slider going from 1...5 setting the speed of color cycling. To be able to identifiy every gadget, give it a name. Using the example above:

```
CHUNK:MUI-WINDOWLAYOUT
ColumnGroup(2),
   Label( "Effect _speed" ),
   speed = Slider( "s", 1, 10 ),
   Label( "_Artifical Intelligence" ),
   ai = Slider( "a", -5, 20 ),
   Label( "Double _Buffering" ),
   HGroup,
      dbufing = CheckMark( "b" ),
      HVSpace,
   End,
End,
```

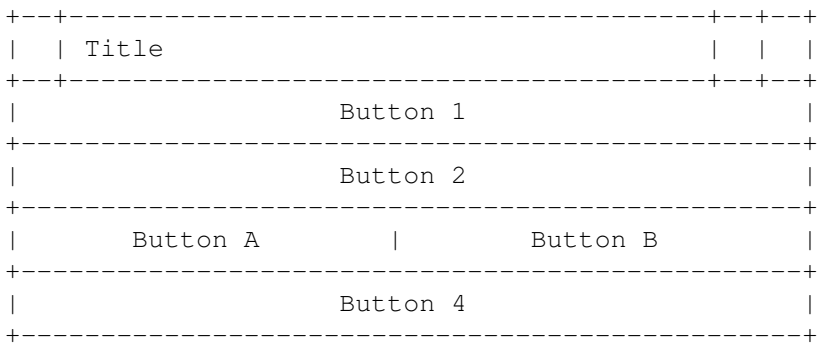
The strings "speed", "ai" and "dbufing" can now be used to identifiy gadgets for madblankersupport.library. See AutoDoc.

If you have understood the basics of MUI, you can also write your own MUI application with your knowledge. The code differs a bit (since you are programming in a real language, and don’t write instructions into a small file for Madhouse). Then, a big compiler goes through your code and compiles it, which differs from how Madhouse accesses the information. Where is the difference?! A compiler is a program which some people work on, for years. My interpreter for the MadhouseConfigEd was done by myself in four days. Please excuse me,

- I have not included every MUI-feature into the interpreter. The weight of every object is fixed to 100, you cannot use CustomClasses (why should you). But you can do a lot. And the MUI BlankerPrefs windows generated out of the gadget files look as good as ’real’ MUI applications.

o I was too lazy to handle ANY error in the gadget file. You can make thousands of errors, and Madhouse should not crash (I hope), but will not get an individual error message (or you get no one at all) for every mistake. You will recognize if something is wrong (the window will look strange), and it should not be difficult to find the error in such a small file.

1.4 The solution



1.5 The groups

Of course I won’t explain the basics of MUI groups again (see greenhorn chapter).

HGroup and HGroup("Grouptitle")

The HGroup (which places the objects inside horizontally) can have a frame around it and gets visible. The grouptitle will be shown in or above the frame. If you want to have a frame, use the syntax above.

You can do the same thing with VGroups.

ColumnGroup(x) and ColumnGroup("Grouptitle", x)

x stands for the amount of columns, if you use the second syntax you can have a grouptitle.

PageGroup("Title 1|Title 2|Title 3", 0)

This expression makes a register group, like the one the Stars blanker has. Every child of this group (it is common to use only groups as a child for a PageGroup) is placed on it’s own page. The first argument describes the titles of the page. You need to have a unique title for every child the group has. The titles are given in the same way cycle

gadgets get their entries, with a seperating ‘|’.

Please don’t forget the trailing 0 parameter, this is just for compatibility purposes.

For an example of a PageGroup, look at the gadget file of Stars.

1.6 The gadgets

Some standards in this chapter

- Key contains a non-captial letter which is used as the control key for the gadget. The user can control the gadget by pressing this key on the keyboard. Example: Slider("a", 5, 10) makes a slider which can be controlled with "a". You cannot use the chars "o", "c" and "t", because they are already used by the gadgets on the bottom of every BlankerPrefs window.
- Text contains text which will be displayed on the left side of the gadget. Example: LabelCycle("Color _selection", "s", "Red|Green|Blue") would draw "Color selection" next to the gadget, with the char "s" underlined. This is used to show the user the control key of the gadget.

Slider(Key, Minimum, Maximum)

creates a slider. It ranges from Minimum to Maximum.

LabelSlider(Text, Key, Minimum, Maximum)

like above, but has an additional label (text) on the left side.

CheckMark(Key)

creates a Checkmark.

Cycle(Key, Entries)

creates a cycle gadget with some entries. "Entries" have to contain all entries of the cycle gadget, separated by "|". Example: "RGB|HSV|CMYK". MUI gives the first entry the number 0, so if you select "RGB", madblankersupport.library / MBS_GetPrefs() will return 0.

LabelCycle(Text, Key, Entries)

like above, but has an additional label (text) on the left side.

String(Key, Length)

creates a string gadget which can contain max. ‘Length’ chars. The upper limit is 500 chars.

LabelString(Text, Key, Length)

like above, but has an additional label (text) on the left side.

Label(Text)

This object gives you the posibility to place your objects better. If you separate gadget and label (as you do it normally), the windows will look much better. Use this object and a gadget without Label... in front of its name.

LLabel(Text)

like above, but this object aligns the text left, not right. This is useful in placing extra text on the right side of a slider (e.g. "seconds", "minutes", "m", "Objects", things like that). An example of this object can be found in the gadget file of Waves.

HVSpace

The important dummy object, its usage was explained in the greenhorn chapter.

HBar

An elegant object which draws a horizontal bar to separate some gadgets. Should be used only in horizontal groups.

VBar

like above, but draws a vertical bar. An example can be found in the gadget file of Stars.

Font or Font(Key)

You will get a font popup in your BlankerPrefs window. As shown in the title, you can use two forms:

Font,
or

Font("f"),
The latter case enables you to give a hotkey for this gadget. The result can be asked with MBS_GetFontPrefs().

File or File(Key)

As above, but with a file requester instead of a font requester. Use MBS_GetStringPrefs() to obtain the prefs setting.

Dir or Dir(Key)

As above, but with a directory requester.

ScreenMode(Colors)

Makes a button for a screenmode requester. If Colors are not zero, there will be a slider enabling the user to select the number of colors for your screen. (If there is no prefs file available when opening the

BlankerPrefs window, the given value for Colors will be used as an initial value for the slider.)
Please don’t use the ScreenMode statement more than once in a gadget file.

1.7 The CHUNK:BLANKERINFO

WARNING: Madhouse reads the data in this chunk while reading the Blankers directory, NOT (!) while opening the BlankerPrefs window. If you change something in this chunk, Madhouse will use the old cpu loading and stack values until you use the ‘Update’ gadget!

```
CHUNK:BLANKERINFO
name
version
cpu-usage
stack
WBDisplay
Protocol
Category
```

- name = Your Name.
- version = The version of your blanker.
- cpu-usage = This value can be 0 or 1. You should set it to ‘0’, if your blanker needs none (or very little) CPU performance, otherwise ‘1’. Madhouse needs this information to check easily if it can run your blanker while a raytracing program is calculating, for example.
- stack = The stack depends on your compiler and on your program. Try 5000. If your blanker crashes or aborts, try more... (See the documatation for your compiler). Don’t use values that cannot be divided by four.
- WBDisplay can be 1 or 2. 2 means that your blanker copies the Workbench or the frontmost screen onto his own screen. 1 is right if you never do so. (Madhouse needs this information to know if it occasionally has to undimm and close it’s tiny black screen before starting your blanker; see Blankers Page, ‘Shows WB’.)
- Protocol has to be ‘Madhouse2’. (Please be sure to write Madhouse2, with a trailing 2, as ‘Madhouse’ works somewhat different (no library needed) and is not supported any more.
- Category: If you want, you can categorize your blanker. At the moment, this setting only influences the sorting of your blanker in the list. One of these keywords can be used in this line:
    - Anim, for blankers which use small, animated pictures (like FlyingToasters);
    - Clock, if your blanker has something to do with date and time;
    - Pyro, for all fireworks and fountains;
    - Text, if your blanker displays some text;
    - Fract, for effects which come out of fractals;
    - View, for blankers which display whole anims and pictures (in contrary to Anim!)
    - Algo, for algorhythmic blankers. Every program can only

```
      consist of algorhythms, so this category means cellular
      automations like Life;
   o  Sim, for simulations of rain or software failures and so on;
   o  3D, if your blankers draws 3D objects or moves them in a
      threedimensional way;
   o  Pixel, if the effect consists of single, isolated pixels, and
      it is not a firework ;-)
   o  Cycle, for effects which would look very boring without color
      cycling;
   o  WB, for all blankers which copy the frontmost or the Work-
      bench screen and do their work on it;
   o  Geo, if you are drawing geometrical objects (circles,
      rectangles, lines in special forms), this is the right
      category for you;
   o  Lines+Splines, if your blanker draws these well-known simple
      lines or splines;
   o  ?, if you don’t know at all, or if no category matches the
      character of your blanker.
```

1.8 The CHUNK:LOCALE

```
CHUNK:LOCALE
language1,language2,...
   While localizing (giving several languages to a program) Madhouse, I
   found out that I need to localize the BlankerPrefs windows, too. There-
   fore, the chunk MUI-WINDOWLAYOUT needs the gadget texts for every lang-
   uage, not just for one. Example:


CHUNK:MUI-WINDOWLAYOUT
ColumnGroup(2),
   Label( "_Mode,_Modus" ),
   Cycle( "m", "Bouncing Point|Wusel|Random,Springender Punkt|Wusel|Zufall" ),
   Label( "Wusel _lenght,Wusel_länge" ),
   Slider( "l", 1, 20 ),
   Label("_Sound,To_neffekt"),
   HGroup,
      CheckMark( "s,n" ),
      HVSpace,
   End,
End,

   The Locale chunk concerns all textdefinitions (recognizable with the
   "-char) in the gadget file. As you can see, every text has a ,-char now.
   The comma seperates the different languages. The left parts of the texts
   are the english texts (1st language), the right parts the german ones
   (2nd language). Other languages could follow. So the right CHUNK:LOCALE
   looks like this:


CHUNK:LOCALE
english,deutsch
```

Madhouse works like this:

- If the MadhouseConfigEd recognizes on startup, that this is not OS 2.1 or higher Amiga, it uses english as the default language. Otherwise it uses the language set with the Locale editor (from the Workbench), for example deutsch.
- Before opening a BlankerPrefs window, MadhouseConfigEd searches for the CHUNK:LOCALE. If there is none (this Chunk is optional!), it prints the text into the BlankerPrefs window without any change. If there is one, MadhouseConfigEd looks into the line with the languages and searches the user-defined language ("english" if this is an OS 2.0 Amiga). If it can find this language, it memorizes the amount of commas skipped, and will skip this amount of commas for every text definition. If it cannot find the language, it uses the first one (SHOULD BE "english"). If some text in your MUI-WINDOWLAYOUT has no comma at all, the text will be printed as it is.

As you can see above, in cyclegadget defintions the comma has a higher priority than the |-separator. Madhouse does the locale-parsing first, and then separates the cycle entries. The control keys are text too. You can define different keyboard "maps" for every language, perhaps "_Sound,To_neffekt" and "s,n". If a translation produces the same word ("_Level,_Level" and "l,l"), you don’t have to use the locale-feature and can write "_Level" and "l". Madhouse does no locale-parsing if it cannot find a comma in the string.

Use non-capital letters for the language names, and use the language itself to write the name of it. Examples: français, english, deutsch, ...

I hope I made all of this clear - ..

1.9 Index

Index

In this index, you will find all key words which I found useful or interesting. Inside the button, there is the headword. On the right, the chapter to which it leads.

If you click on a button, you will get the corresponding page. This AmigaGuide is designed so that it will scroll automatically to the line in which the headword occurs, if possible. Headwords are printed underlined there.

Checkmark
The gadgets

Child
for MUI-Greenhorns

Chunk-Header
The gadget file

Chunks
The gadget file

column groups
for MUI-Greenhorns

control key
The gadgets

cpu-usage
The CHUNK:BLANKERINFO

Cycle
The gadgets

cycle entries
for MUI-Greenhorns

cyclegadget
for MUI-Greenhorns

Dir
The gadgets

End
for MUI-Greenhorns

File
The gadgets

fixed height
for MUI-Greenhorns

Font
The gadgets

group
for MUI-Greenhorns

grouptitle
The groups

HBar
The gadgets

HGroup
The groups

```
horizontal bar
                        The gadgets

HVSpace
                              for MUI-Greenhorns

HVSpace
                              The gadgets

identifiers
                           for MUI-Greenhorns

indent
                               for MUI-Greenhorns

index
                                Index

keyboard
                              for MUI-Greenhorns

label
                                 for MUI-Greenhorns

Label
                                 The gadgets

LabelCycle
                            The gadgets

labels
                                 for MUI-Greenhorns

LabelSlider
                           The gadgets

LabelString
                           The gadgets

language
                              The CHUNK:LOCALE

LLabel
                                 The gadgets

Locale chunk
                          The CHUNK:LOCALE

MUI groups
                            The groups

MUI-windowlayout
                       for MUI-Greenhorns

name
                                    The CHUNK:BLANKERINFO
```

```
PageGroup
                              The groups

Protocol
                               The CHUNK:BLANKERINFO

register group
                         The groups

ScreenMode
                             The gadgets

slider
                                 for MUI-Greenhorns

Slider
                                 The gadgets

stack
                                  The CHUNK:BLANKERINFO

stretching-limitations
               for MUI-Greenhorns

String
                                 The gadgets

underline
                               for MUI-Greenhorns

underscore
                              for MUI-Greenhorns

VBar
                                    The gadgets

version
                                 The CHUNK:BLANKERINFO

VGroup
                                 The groups

WBDisplay
                               The CHUNK:BLANKERINFO
```