textfield_gc

COLLABORATORS

	TITLE : textfield_gc	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		November 28, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1 textfield_gc **1**

1.1 textfield_gc.doc 1

1.2 textfield.gadget/textfield.gadget 1

1.3 textfield.gadget/TEXTFIELD_GetClass 17

Chapter 1

textfield_gc

1.1 textfield_gc.doc

```
    textfield.gadget()
    TEXTFIELD_GetClass()
```

1.2 textfield.gadget/textfield.gadget

```
  NAME
textfield.gadget -- multi-line text entry BOOPSI object (V2)

  LEGAL
textfield.gadget is © 1994 Mark Thomas
All rights reserved.

  FUNCTION
The textfield class allows you create an area on your screen for text
entry.  The class supports a number of features including unlimited
or limited size text entry, specifying the font and style to use,
specifying the colors for different parts (text, background, and
lines), two types of borders (with option to invert the borders for a
total of 4 types of borders) or no border, text left/center/right
justification, vertical centering, IntuiText labels, attached Images,
line spacing, limited character acceptance, insertion or block cursor,
cursor blinking speed or no blinking, read-only mode, modified buffer
flag, lined paper, cut/copy/paste/erase while editing or program-
matically, undo buffer, and settable word delimiter characters for
word wrapping.


SIZE ISSUE

The textfield class gadget should be used for relatively small areas
where you want to allow text entry.  Typically a size below 320 x 200
is a fairly reasonable limit, but larger sizes will work.  In other
words it is not intended to be a whole editor like Ed.  Please note
that the vertical size has more affect on speed than the horizontal
size.  Many people have ignored the size issue in previous versions,
```

and it doesn't seem to have caused problem. I keep this here until the day I can really optimize the gadget.

PROGRAMMING LANGUAGE

This gadget is written for a C language interface and is that way simply because the Amiga development is geared more towards the C language than any other. This does not mean that the gadget cannot be used with another language. In particular, support for Oberon is given in the form of two mod files. One file, TextField.mod provides the actual interface to the gadget class. It can be compiled and linked into an Oberon project. The other, TestClass.mod is a sample program, similar to the C version, that tests the class and shows how to use the class. Thanks goes to Stefan for taking the effort to convert the files.

If you use C, the important thing to know is that you need to include the header file <gadgets/textfield.h>. And depending on what you do you may need to include some other files: <intuition/gadgetclass.h>, <intuition/classes.h>, <intuition/classusr.h>, <intuition/icclass.h>, <intuition/imageclass.h>, <utility/tagitem.h>.

HOW TO OPEN THE TEXTFIELD GADGET LIBRARY (for non-SAS/C 6.50+ users)

To use the class you must open the "textfield.gadget" library like so: OpenLibrary("gadgets/textfield.gadget", 0). If that was successful, then you need to get the class pointer with the TEXTFIELD_GetClass() function. You do not have to check to make sure that the pointer returned by TEXTFIELD_GetClass() is valid (see TEXTFIELD_GetClass description). You do not have to return the pointer with any function, just do not use the pointer after you close the "textfield.gadget" library.

Here's a quick guide:

```
struct Library *TextFieldBase;
Class *TextFieldClass;

TextFieldBase = OpenLibrary("gadgets/textfield.gadget", 0)
if (TextFieldBase) {
    TextFieldClass = TEXTFIELD_GetClass();

    /* use the class */

    TextFieldClass = NULL;
    CloseLibrary(TextFieldBase);
}
```

HOW TO OPEN THE TEXTFIELD GADGET LIBRARY (for SAS/C 6.50+ users)

If you use SAS/C 6.50 or greater, then you can use the TextFieldAuto.c source file provided in the source draw to have the library open automatically at startup, and close automatically at termination so that you will not have to take care of these yourself. I highly

recommend using this feature. The file makes two variables available to the client:

```
extern struct Library *TextFieldBase;
extern Class *TextFieldClass;
```

Starting in main() these varaibles will automatically be defined to have the library base and the class pointer. If the library fails to open, SAS/C's auto-open feature will close down the program with a standard Amiga requester telling the user what happened.

NO PUBLIC CLASS NAME

Future versions of the textfield.gadget intend to offer a public class name that you can use, but that will have to wait until (or if) Commodore or CATS return from the dead since I have to register the class name. At which time that a public class name is offered, both the old TEXTFIELD_GetClass() and the class name can be used with the NewObject() function so that old and new programs will work.

WORKS ON OS 2.04 AND UP

The class will work on OS 2.04 and up, but for OS 2.04 through 2.1 there is no standard place to put .gadget libraries, so on these systems the gadget should be installed in a drawer named "Gadgets" in the same directory of the program that uses it. See the example program on how to open the class. For OS 3.0 and up, just install the textfield.gadget file in "SYS:Classes/Gadgets" like normal.

COMPARISON TO STRING GADGET

Unlike the Amiga's string gadget, you get a gadget up message from this gadget when you hit the right mouse button, or either Amiga keys and the right alt keys pressed together.

GADGET GRAPHIC ELEMENTS SUPPORTED

This gadget does support the GA_Image tag for rendering a linked list of images attached to the gadget. Borders structures are not supported. (Complain if you want it.)

As of version 2.0 the gadget also supports GA_IntuiText also. This means this gadget will render a linked list of IntuiText structures. Note that GA_Text and GA_LabelImage are not supported. (Like above, complain if you want it.)

This may or may not help you, but the render order for the elements of this gadget is as follows:

```
    Border          (if enabled)
    Lines of text
    Image           (from GA_Image)
    IntuiText       (from GA_IntuiText)
```

So, images do overwrite borders and the lines of text. Also note that the gadget will render at least one line of text, and at least one character per line, so restrict the width and height if this is a problem.

IMPORTANT NOTES ON ACCESSING THE BUFFER

You must follow these rules or bad things can happen.

The buffer pointer returned by TEXTFIELD_Text with the get method is read only. You must not modify the buffer at any time.

The buffer pointer returned by TEXTFIELD_Text with the get method is valid between any of the following:

```
OffGadget()
 valid
OnGadget()

GA_Disabled, TRUE
 valid
Disabled, FALSE

TEXTFIELD_ReadOnly, TRUE
 valid
TEXTFIELD_ReadOnly, FALSE
```

Anytime you are outside all of the above situations, the pointer is invalid and is guaranteed to change. In fact you will get a NULL pointer if you are outside all of the above situations.

The size of the buffer you have can be obtained by issuing a get method of TEXTFIELD_Size on the gadget inside the above situations. And if you get a size of 0, then you must not reference the pointer that is returned because it is NULL. Be very careful to never reference past the size given. The buffer is not NULL terminated. In C, that's buffer[0] through buffer[size - 1].

CHARACTERS ALLOWED IN THE BUFFER

The text you get from TEXTFIELD_Text only contains values from 0x0a, 0x20 - 0x7f, and 0xa0 - 0xff. In other words you get the printable characters, plus '\n'. No tabs are supported for now.

When characters are sent to the gadget, either by the application or by the user, only the characters listed above are allowed in. You can also make further restrictions with TEXTFIELD_AcceptChars and TEXTFIELD_RejectChars. There is one other character that is allowed in, 0x0d, but it is converted to 0x0a for you. When you read the buffer, it will return only 0x0a characters.

CHANGING ATTRIBUTES

When setting attributes, the gadget will always render the changes, except if the gadget's geometry is changed, in which case it will return a non-zero value to let you know changes need to be rendered by refreshing the gadget.

A geometry change means that the gadget moves in the window. Specifically the following attributes cause the geometry to change:

GA_Top, GA_Left, GA_Width, GA_Height, GA_RelWidth, GA_RelHeight, GA_RelBottom, GA_RelRight.

Here's how to handle this inside an application.

```
if (SetGadgetAttrs(gadget, window, requester, ...)) {
  /* gadget needs rendering */
  RefreshGList(gadget, window, requester, 1);
}
```

MOVING OR SIZING THE GADGET

If you want to move the gadget or change its size you should take note of the section above on changing attributes. In addition, the source file TestClass.c in the TestClass example shows how to move the gadget or change its size. You should clear out the old position, then set the new position and size, and finally refresh the gadget.

APPLICABILITY IN THE TAGS SECTION

In the TAGS section there is a statement with each attribute called applicability. It consists of a letters inside parentheses, like (ISGNU). These are standard BOOPSI letters that tell you which standard methods the attribute can be used. Here's what the letters refer to (see the BOOPSI reference in the RKRM: Libraries for further information):

```
    I - OM_NEW     (initializable)
    S - OM_SET     (settable)
    G - OM_GET     (gettable)
    N - OM_NOTIFY  (is passed in gadget's notify message)
    U - OM_UPDATE  (updatable)
```

SETTING ATTRIBUTES

Some of the attributes cause others to not work properly. Do not get discouraged. This only applies when sending then in one OM_SET method. It is unavoidable in the current release, so if you are seeing things not work right, or in particular if a later tag works but a former one does not, then split the attributes among two calls. The gadget is set up so that later tags have precedence over former tags. This particularly applies to attributes that change the cursor position and scrolling (changing top and such).

UNDO

The undo function is implemented as an alternate paste buffer. In certain situations list below text will be copied into an undo buffer. When the undo command is issued, the text in that buffer is pasted to the current cursor position. It does not really undo anything, it just saves otherwise lost text. When the undo is actually performed, the undo buffer becomes cleared so that you cannot undo the same text multiple times.

List of places where text is saved in the undo buffer:

- If you hit BACKSPACE or DELETE while some text is marked, the marked text is placed in the undo buffer before the text is deleted.

- If you hit SHIFT BACKSPACE or DELETE, the text that is deleted is placed in the undo buffer.

- If you hit ALT BACKSPACE or DELETE, the word that is deleted is placed in the undo buffer.

- If you hit CTRL X, the line that is deleted is placed in the undo buffer.

- If you type a character while some text is marked, the marked text is placed in the undo buffer before the text is delete.

- When you perform a normal paste while text is marked, the marked text is placed in the undo buffer before it is deleted and the paste occurs.

- If you perform an erase (RAMIGA E or programmatically), the whole buffer is placed in the undo buffer before it is deleted.

DOCS FOR USERS

You can mark text for cutting, copying, and erasing by simply clicking and dragging. Hitting alphanumeric keys replaces the text that is highlighted. Hitting cursor keys moves you to the front or end of the highlighted text.

While you drag to scroll, the farther away from the gadget your mouse pointer is, the faster the gadget will scroll.

For key sequences, the Amiga Style Guide was followed. Anywhere the undo buffer is mentioned, the statement is only valid if the UndoStream is supplied (see tag section below).

Key Sequence	Function
TAB	Activate next gadget (if GA_TabCycle)
SHIFT TAB	Activate previous gadget (if GA_TabCycle)
SHIFT CURSOR UP	Move to the top line in the current page, or scroll up one page if cursor is on top line

SHIFT CURSOR DOWN	Move to the bottom line in the current page, or scroll down one page if cursor is on top line
CTRL or SHIFT CURSOR RIGHT	Move to the right end of the current line
CTRL or SHIFT CURSOR LEFT	Move to the left end of the current line
SHIFT BACKSPACE	Delete all text to the left of cursor on the current line
SHIFT DELETE	Delete all text to the right of the cursor on the current line (in block cursor mode this also includes the highlighted character)
CTRL CURSOR UP	Move to the top line of the text
CTRL CURSOR DOWN	Move to the bottom line of the text
ALT CURSOR RIGHT	Move to the next word (using the delimiter characters provided by the programmer)
ALT CURSOR LEFT	Move to the previous word (using the delimiter characters provided by the programmer)
ALT CURSOR UP	Move to first character in gadget
ALT CURSOR DOWN	Move to last character in gadget
ALT BACKSPACE	Deletes the word to the left of the cursor starting at the current cursor position
ALT DEL	Deletes the word to the right of the cursor starting at the current cursor position
CTRL X	Deletes the whole line that the cursor is on
RAMIGA [	Switch to left justification (if TEXTFIELD_UserAlign is set)
RAMIGA \ or RAMIGA =	Switch to center justification (if TEXTFIELD_UserAlign is set)
RAMIGA]	Switch to right justification (if TEXTFIELD_UserAlign is set)
RAMIGA E	Erase all text in gadget (saved in undo buffer) (no read-only)
RAMIGA V	Paste text from clipboard to current cursor position (no read-only)

RAMIGA A	Mark all text
RAMIGA U	Undeletes (pastes) the last block of text marked, or recover from RAMIGA E (no read-only)

When text is highlighted the following keys have functions:

BACKSPACE	Erase marked text (saved in undo buffer)
DEL	Erase marked text (saved in undo buffer)
RAMIGA X	Cut marked text to clipboard (no read-only)
RAMIGA C	Copy marked text to clipboard
RAMIGA V	Replace marked text with text from clipboard (save marked text in undo buffer) (no read-only)
(any text key)	Replace marked text with that character

TAGS

GA_Left (WORD) -- Specifies the left edge of the gadget.

GA_Top (WORD) -- Specified the top edge of the gadget.

GA_Width (WORD) -- Specifies the width of the gadget. If a border is chosen, it will be rednered within this value.

GA_Height (WORD) -- Specifies the height of the gadget. If a border is chosen, it will be rendered within this value.

GA_RelRight (WORD) -- Specifies the gadget as being relative to the right border of whatever the gadget is attached to. See the BOOPSI Class Reference.

GA_RelBottom (WORD) -- Specifies the gadget as being relative to the bottom border of whatever the gadget is attached to. See the BOOPSI Class Reference.

GA_RelWidth (WORD) -- Specifies the gadget as being relative to the width of whatever it is attached to. See the BOOPSI Class Reference.

GA_RelHeight (WORD) -- Specifies the gadget as being relative to the height of whatever it is attached to. See the BOOPSI Class Reference.

GA_Image (struct Image *) -- Pass a pointer to a linked list of images and this class will render them relative to the gadget's upper left corner. See rendering order above.

GA_IntuiText (struct IntuiText *) -- Pass a pointer to a linked list of IntuiText structures and this class will render them relative to the gadget's upper left corner. See rendering order above.

(V2)

GA_Disabled (BOOL) -- TRUE disables the gadget, not allowing input, and FALSE enables the gadget for input. When the gadget is disables, it usually is ghosted, see TEXTFIELD_NoGhost for other conditions.

GA_TabCycle (BOOL) -- Turns on tab cycling. See the BOOPSI Class Reference.

TEXTFIELD_Text (char *) -- This set/replaces text. NULL means no change. To set the buffer empty pass "" (pointer to empty string, not a NULL pointer). When you use it to get text see special conditions under IMPORTANT above. The cursor position is reset to the beginning (0). The pointer passed must be a pointer to a NULL terminated string of characters. If you have read only mode turned on with TEXTFIELD_ReadOnly, this attribute still works.

Default for this tag is NULL. Applicability is (ISG U).

TEXTFIELD_InsertText (char *) -- This inserts text at current cursor position and move the cursor to the end of the inserted text. You must pass a NULL terminated pointer to characters. If you have read only mode turned on with TEXTFIELD_ReadOnly, this attribute still works.

Applicability is (S U).

TEXTFIELD_DeleteText (ULONG) -- This deletes the number of characters you pass in the data field, starting at the current cursor position. This attribute works even if read only mode is turned on with TEXTFIELD_ReadOnly.

Applicability is (S U). (V2)

TEXTFIELD_TextFont (struct TextFont *) -- Sets the font for the gadget to use. Pass the object a pointer to a TextFont structure. This supersedes TEXTFIELD_TextAttr below. Please do not close this font while the gadget is using it. :) The default font is your screen's current font.

Default for this tag is NULL. Applicability is (IS U).

TEXTFIELD_TextAttr (struct TextAttr *) -- Sets the font the gadget is to use. Pass the gadget a pointer to a TextAttr structure. This is superseded by TEXTFIELD_TextFont. The default font is the screen's current font.

Default for this tag is NULL. Applicability is (IS U).

TEXTFIELD_FontStyle (ULONG) -- The style will get set to what you pass here. The font style automatically gets reset when TEXTFIELD_TextFont or TEXTFIELD_TextAttr is set.

Default for this tag is FSF_PLAIN. Applicability is (IS U).

TEXTFIELD_Delimiters (char *) -- You get the default if you pass NULL. Words break after these and "\n". You will probably want at least " ", the space. This string is not copied, so do not get rid of it while the gadget is in use.

Default for this tag is ",)!@^&*_=+\|<>?/ ". Applicability is (IS U).

TEXTFIELD_AcceptChars (char *) -- Tells the textfield gadget which characters to accept. All others are rejected, and non-printable characters are automaically rejected. Some of these characters may not be accepted if they also appear in the reject string. This string is not copied, so do not get rid of it while the gadget is in use. A NULL means to accept all characters.

Default for this tag is NULL. Applicability is (IS U). (V2)

TEXTFIELD_RejectChars (char *) -- Tells the textfield gadget which characters to reject. Non-printable characters are automaically rejected. Characters specified with this are always rejected. This string is not copied, so do not get rid of it while the gadget is in use. A NULL means to reject no characters.

Default for this tag is NULL. Applicability is (IS U). (V2)

TEXTFIELD_BlinkRate (ULONG) -- This sets the number of microseconds between a cursor on-off, or off-on transition. A value of 0 means do not blink. Realistically, this should be set to 100000 or higher since BOOPSI objects don't get idle messages any faster than about once every 10th of a second, but any value between 0 and 100000 will just make the cursor blink as fast as it can. If you give the user an option of blink speed, suggest values: 0 for no blink, 750000 for a slow blink, 500000 for a medium blink, and 250000 for a fast blink.

Default for this tag is 0. Applicability is (IS U).

TEXTFIELD_BlockCursor (BOOL) -- Turn on/off block cursor mode. You should not use a block cursor if your font is italic because it looks weird.

Default for this tag is FALSE. Applicability is (IS U).

TEXTFIELD_CursorPos (ULONG/ULONG *) -- Get/Set the cursor position. The cursor position returned is always an exact offset into the buffer you get to read via TEXTFIELD_Text. 0 takes you to the first character in the gadget, and 0xFFFFFFFF takes you past the last character in the gadget. In general, any value you pass that is larger than what's returned by TEXTFIELD_Size will end up just past the last character in the gadget. Setting this attribute will scroll the text to the position you set the cursor to. If there is any text highlighted, highlighting is turned off. When you get the current cursor position, you must pass a pointer to a ULONG. The ULONG will then have the cursor position in it.

Applicability is (ISG U).

TEXTFIELD_Size (ULONG *) -- Returns the number of characters in the gadget's buffer, including \n characters. This gives you the size when you want to use TEXTFIELD_Text to read the text in the gadget. You must pass a pointer to a ULONG and then the ULONG will contain the value.

Applicability is (G).

TEXTFIELD_MaxSize (ULONG) -- Limit the size of text entered into the gadget. 0 means unlimited, otherwise limits the buffer size to what you pass. This includes \n characters.

Default for this tag is UNLIMITED. Applicability is (I).

TEXTFIELD_MaxSizeBeep (BOOL) -- This attribute lets you set whether the system DisplayBeep() function is called if the user attempts to enter text into the gadget and the gadget is full (i.e. the gadget's buffer size is equal to the limit set by TEXTFIELD_MaxSize). If TRUE the system beep is called. If FALSE, the system beep is not called.

Default for this tag is TRUE. Applicability is (IS U). (V2)

TEXTFIELD_Visible (ULONG *) -- Get the current number of visible lines. It always returns how many _could_ be displayed if there were enough characters to fill the display. Use for notifying a BOOPSI prop gadget. You actually pass a pointer to a ULONG and then the ULONG will contain the value. See example program.

Applicability is (GN).

TEXTFIELD_Lines (ULONG *) -- Get the total number of lines in the buffer of the gadget. Use this to also notify a BOOPSI prop gadget. You pass a pointer to a ULONG and then the ULONG will contain the value. See example program.

Applicability is (GN).

TEXTFIELD_Top (ULONG/ULONG *) -- Get or set ordinal value of the top line. 0 is the top most line. This is useful for ICA_MAP and ICA_TARGET when using the BOOPSI prop gadget and notifications. When you get the value you pass a pointer to a ULONG and then the ULONG will contain the value. See sample program for example.

Default for this tag is 0. Applicability is (SGNU).

TEXTFIELD_Partial (BOOL) -- When this flag is set to TRUE, partial lines will be shown at the bottom of the gadget. When this flag is set to false, then only whole lines will be shown in the gadget. Note that having both TEXTFIELD_VCenter, and TEXTFIELD_Partial on is not allowed and doesn't make sense. If both TEXTFIELD_VCenter and TEXTFIELD_Partial are turned on

at the same time, only TEXTFIELD_VCenter will get turned on.

Default for this tag is FALSE. Applicability is (IS U).

TEXTFIELD_NoGhost (BOOL) -- If TRUE, never ghost when gadget is disabled. If FALSE, then ghost when gadget is disabled. You can use this to make a read only multiline string gadget. It has a special purpose, though.

Normally you will want a gadget to be enabled when allowing text to be entered. However, when you need to read the text from the gadget, you have to disable it. But disabling a gadget ghosts it. So, with this option, you can pass GA_Disabled, TRUE, TEXTFIELD_NoGhost, TRUE at the same time and it will disable without ever showing the ghosted pattern. And likewise, passing the attributes GA_Disabled, FALSE, TEXTFIELD_NoGhost, FALSE will seamlessly reenable the gadget. While the gadget is disabled, read the text and then be on your way. Also note that most S and U attributes are settable while the gadget is disabled, notably TEXTFIELD_Top. This allows you to make a scrollable read-only multiline non-ghosted text, image capable, and border capable gadget. Sounds useful to me!

The read only mode is probably better. See below.

Default for this tag is FALSE. Applicability is (IS U).

TEXTFIELD_ReadOnly (BOOL) -- If TRUE, then the gadget does not allow text to be entered or deleted by the user. However, the user can still highlight so that text can be copied to the clipboard. If FALSE, then the gadget acts normally. If you want a read only mode without clipboard support, then use the GA_Disabled and TEXTFIELD_NoGhost mode mentioned in the TEXTFIELD_NoGhost description.

All but two of the normal keyboard editing commands are ignored while in read-only mode. RAMIGA-a highlights all text, and RAMIGA-c copies highlighted text to the clipboard. Other RAMIGA keys will get passed is TEXTFIELD_PassCommand is turned on.

So that the gadget acts more like it’s not active while in this mode, certain keys events get passed on to the window: F1-F10, Help, Cursor keys.

Note that non-RAWKEY events, like mouse moves, buttons, and timer events are not passed on so that normal highlighting and scrolling can occur.

There is no visual change in the gadget when it is read-only other than it does not have a cursor. You should set the border to an inverted bevel or inverted double-bevel. By the style guide, inverted borders represent read-only gadgets.

Default for this tag is FALSE. Applicability is (IS U). (V2)

TEXTFIELD_Modified (BOOL) -- This attribute is a flag that you can set and read to find out if a gadget's buffer has been modified in some way. For instance, if you wanted to load some text into the gadget, then when the program quits determine if the text was modified then you would load and set the flag like this: TEXTFIELD_Text, text, TEXTFIELD_Modified, FALSE. Note that the order matters, such that if they were reversed the text change would promptly set the modified flag to TRUE. You can read or set this flag at any time.

Default for this tag is FALSE. Applicability is (IS U). (V2)

TEXTFIELD_PassCommand (BOOL) -- This attribute is a flag to allow the gadget to pass unused right Amiga key sequences (command key sequences) on to the client application. If the flag is set to TRUE then the key sequences are passed to the client. There is a side effect that causes the gadget to become inactive (as if the user had clicked outside the gadget). The default behavior (set to FALSE) is to ignore these key sequences so that the gadget remains active.

Default for this tag is FALSE. Applicability is (IS U). (V2)

TEXTFIELD_Border (ULONG) -- Sets the border type. See defines below. The gadget offers a standard bevel, and standard double bevel. If you need another type, your could always create an image, link it to the gadget with GA_Image, and set its top and left edges above and to the left of this gadget (negative in the image structure), and make the width and height larger than this gadget.

Default for this tag is TEXTFIELD_BORDER_NONE. Applicability is (IS U).

TEXTFIELD_Inverted (BOOL) -- If this flags is TRUE, the border is drawn inverted, if there is a border. If FALSE, the border is drawn non-inverted. This option is here in case you want to give the textfield gadget a read-only look when used in conjunction with TEXTFIELD_NoGhost and GA_Disabled.

Default for this tag is FALSE. Applicability is (IS U).

TEXTFIELD_Up (ULONG) -- Moves the text up by one line. You can pass anything, but it will only move the text up by a line, if it's not at the top already. Useful BOOPSI notifications.

Applicability is (S U).

TEXTFIELD_Down (ULONG) -- Moves the text down by one line. You can pass anything, but it will only move the text down by a line, if it's not at the bottom already. Useful for BOOPSI notifications.

Applicability is (S U).

TEXTFIELD_Alignment (ULONG/ULONG *) -- Set/Get the line justification. This gadget offers left, center, and right justification. When you are getting the flags, you need to pass a pointer to a ULONG and then the ULONG will contain the flags. See defines below.

Default for this tag is TEXTFIELD_ALIGN_LEFT. Applicability is (ISG U).

TEXTFIELD_VCenter (BOOL) -- Turn on/off vertical centering. When on, the lines in the display are centered vertically. If the total number of lines is less than the visible number of lines then the smaller number of lines are centered. This allows you to center single lines of text within the gadget very easily. For normal text entry operation, it is best to leave this off. Also, check TEXTFIELD_Partial for possible conflicts when used with TEXTFIELD_VCenter.

Default for this tag is FALSE. Applicability is (IS U).

TEXTFIELD_UserAlign (BOOL) -- If this is set at creation, then the user will have control over the left/center/right justification of text through RIGHT-AMIGA [, =,] keyboard shortcuts. If you want to save what the user has set the justification to, then do a GetAttr() on TEXTFIELD_Alignment.

Default for this tag is FALSE. Applicability is (I).

TEXTFIELD_RuledPaper (BOOL) -- Lets you set whether the paper (background) has ruled horizontal lines under each line of text or not.

Default for this tag is FALSE. Applicability is (IS U).

TEXTFIELD_PaperPen (ULONG) -- This lets you specify the pen used for drawing the paper (background) of the gadget. A value of -1 means use default, which is BACKGROUNDPEN.

Default for this tag is -1. Applicability is (IS U).

TEXTFIELD_InkPen (ULONG) -- This lets you specify the pen used for drawing the text. A value of -1 means use the default, which is TEXTPEN. If this pen, and the TEXTFIELD_LinePen are different, then rendering speed is slowed down a bit. It is recommended that the line pen be left to -1.

Default for this tag is -1. Applicability is (IS U).

TEXTFIELD_LinePen (ULONG) -- This lets you specify the pen used for drawing the ruled lines, if TEXTFIELD_RuledPaper is TRUE. See TEXTFIELD_InkPen for possible speed problems when specifying this pen. A value of -1 means to use the same pen as TEXTFIELD_InkPen.

Default for this tag is -1. Applicability is (IS U).

TEXTFIELD_Spacing (UBYTE) -- Lets you set an extra amount of spacing between lines of text, for maybe doing 1-1/2 or double spacing. It's a pixel value between 0 and 255. The space is added to the top of each line. In other words, the baseline is moved down by the amount you specify.

Default for this tag is 0. Applicability is (IS U).

TEXTFIELD_ClipStream (struct ClipboardHandle *) -- This tag allows clipboard support in the gadget. Pass the pointer returned from the iffparse.library OpenClipboard() function. If a NULL is passed, the clipboard support is not allowed. Please supply this tag value. Don't leave users without clipboard support. It is recommended that the unit opened by OpenClipboard() be 0 or PRIMARY_UNIT, since that is the standard unit, but you can pick whatever unit you or your user wants. This stream can be safely given to multiple objects.

Default for this tag is NULL. Applicability is (I).

TEXTFIELD_ClipStream2 (struct ClipboardHandle *) -- ClipStream2 is used for the undo features of the textfield class. It is obtained from the iffparse.library OpenClipboard() function. You should probably use a clipboard unit other than 0 to avoid conflicts with normal clips. This stream can be safely passed to multiple objects. (See TEXTFIELD_UndoStream)

Default for this tag is NULL. Applicability (I).

TEXTFIELD_UndoStream -- This is a synonym for TEXTFIELD_ClipStream2. See it above.

TEXTFIELD_LineLength (ULONG *) -- Get the contents of the ULONG to a line number and that ULONG will be replaced with the length of that line in characters. If you are worried about the line lengths changing while you are reading them, then try removing the gadget from the window first. Please realize that if the gadget has a relative width or height then the lengths can change anytime the user resized the window.

Applicability is (G). (V2)

TEXTFIELD_SelectSize (ULONG/ULONG *) -- Set the number of characters to highlight starting from the current cursor position. If the value is 0, highlighting will be turned off. If the number is non-zero, that number of characters will be highlighted. You can also get the number of highlighted characters with this attribute by passing a ULONG *. If you get a 0 value, then highlighting is turned off. This can is useful for the copy, cut, and paste commands.

Applicability is (SG U). (V2)

TEXTFIELD_Copy -- When this attribute is specified, the gadget copies any highlighted text to the clipboard. Highlighting is turned off after issuing this. You can put Copy in a menu with this.

This does not take a parameter.

Applicability is (S U). (V2)

TEXTFIELD_CopyAll -- When this attribute is specified, the gadget copies all of the gadget's text to the clipboard. It does not affect the state of the highlight and the text does not need to be highlighted. You can put Copy All in a menu with this. This does not take a parameter.

Applicability is (S U). (V2)

TEXTFIELD_Cut -- When this attribute is specified, the gadget will cut any highlighted text to the clipboard. Cut text goes to the clipboard. You can put Cut in a menu with this. This does not take a parameter.

Applicability is (S U). (V2)

TEXTFIELD_Paste -- When this attribute is specified, the gadget will delete any highlighted text to the undo buffer, then it will paste the text from the clipboard to the gadget. You can put Paste in a menu with this. This does not take a parameter.

Applicability is (S U). (V2)

TEXTFIELD_Erase -- When this attribute is specified, the gadget will delete any highlighted text, otherwise do nothing. If text is deleted, it's placed in the undo buffer. You can put Erase in a menu with this. This does not take a parameter.

Applicability is (S U). (V2)

TEXTFIELD_Undo -- When this attribute is specified, the gadget will delete any highlighted text, then insert the text from the undo buffer. You can put Undo in a menu with this. This does not take a parameter.

Applicability is (S U). (V2)

BORDER REFERENCE

You can use the width and heights given when calculating window sizes and limits. To make the window's height minimal with respect to your font, use (window border top) + (window border bottom) + (num_lines * (font height)) + (gadget border height). Also, if you use TEXTFIELD_Spacing, you'll have to add that in too.

TEXTFIELD_BORDER_NONE	Border takes up: 0 width, 0 height
TEXTFIELD_BORDER_BEVEL	Border takes up: 8 width, 4 height
TEXTFIELD_BORDER_DOUBLEBEVEL	Border takes up: 12 width, 6 height

ALIGMENT REFERENCE

TEXTFIELD_ALIGN_LEFT	Cause text to be flush left
TEXTFIELD_ALIGN_CENTER	Cause text to be centered
TEXTFIELD_ALIGN_RIGHT	Cause text to be flush right

BUGS

```
What bugs?  Please let me know if you find any.
See the History file for a list of fixed bugs.

DEBUGGING

A version of the textfield.gadget library is provided with routines
that print debug information to the serial port using the current
settings for the serial device.  It could be helpful.  Just rename
the debug gadget (textfield.debug.gadget) to textfield.gadget and
place it in the normal gadgets place listed above.

  CONTACT
To contact me for reporting bugs or giving suggestions:

Mark Thomas
1515 Royal Crest Dr. #3259
Austin, TX  78741

or

mthomas@cs.utexas.edu

or

URL: http://www.cs.utexas.edu/~mthomas
```

1.3 textfield.gadget/TEXTFIELD_GetClass

```
  NAME
TEXTFIELD_GetClass -- Gets the pointer to the textfield class. (V1.0)

  SYNOPSIS
textfield_class = TEXTFIELD_GetClass();
D0

Class *TEXTFIELD_GetClass(void);

  FUNCTION
Obtains the pointer to the textfield.gadget class for use with
NewObject().  This function always returns a valid pointer so
you do not need to check it.  The reason is that if the library
opens fine, then the pointer returned is already setup.  (Of course
this implies that if opening the library fails, you shouldn't be
calling this.)

  INPUTS
None.

  RESULT
textfield_class - the pointer to the textfield.gadget class.

  BUGS
None.
```