Blugs Technote #3

What's New in Blugs 1.2

Updated Contact Information

Brian 'Moses' Hall

moses@blugs.com
http://www.blugs.com

702 LRDC
University of Pittsburgh
Pittsburgh, PA 15213
USA

Work: 412-624-7498
Home: 412-683-7779
Cell: 412-841-4693

Introduction

Welcome to Blugs 1.2! To prove that Blugs is still alive and well, and still fighting to beat DataBrowser at its own game, I am releasing a substantially modified API for the Blugs List Management Engine. This technote describes the changes in Blugs 1.2. There also some significant changes in distribution policy: changes which should make a lot of people very happy.

Open Source License

I decided that the LGPL license was too restrictive in its requirement that end users can recompile and relink the Blugs libraries; this is inappropriate in this case because the Blugs API is changing too quickly. Adherence to this licensing scheme might result in excessive version-complexity which I wish to very carefully avoid at this very volatile time in the history of Mac OS.

Note please that this license strategy may be abandoned in favor of the Perl Artistic License or one of its derivatives. The important thing is, I want to retain some control over the Blugs API

and project, but I want anyone to be able to use it in their own products without any restriction.

Blugs 1.2 is released under the ZLib license, reproduced below. You can inspect and modify the Blugs source code. This license allows you to use Blugs in a closed-source project. Please read the license and contact me if you have any questions or concerns:

Mach-O

Blugs compiles under ProjectBuilder and CodeWarrior as a Mach-O build. The source code has been heavily modified with `TARGET_RT_MAC_MACHO` conditionals.

Changed and New Structures and Constants

Constants in both the list flags and row flags enumerations have changed. There are also various additions to support the title bar and widget API enhancements.

List Flags

There are two removals and one addition to the list flags. The constants `blDrawColumnBorders` and `blDrawRowBorders` have been removed in Blugs 1.2. This functionality has been replaced by individual row and column flags, and the creation of the `BLSetDefaultRowFlags` and `BLSetDefaultColumnFlags` functions discussed below. `blPinRightColumn` has been promoted from a horizontal title bar property to a general list property, making it accessible to lists that lack a horizontal title bar.

```
enum
{
     blResizableHeight                = 0x00000001,
     blResizableWidth                 = 0x00000002,
     blHorizontalScroll               = 0x00000004,
     blVerticalScroll                 = 0x00000008,
     blLiveScroll                     = 0x00000010,
     blSmallScroll                    = 0x00000020,
     blAutodraw                       = 0x00000040,
     blHasGrow                        = 0x00000080,
     blDrawGrow                       = 0x00000100,
     blCanFocus                       = 0x00000200,
```

```
        blVisible                       = 0x00000400,
        blActive                        = 0x00000800,
        blInlineEditOnClick             = 0x00001000,
        blDrawColumnBorders             = 0x00002000, // GONE!
        blDrawRowBorders                = 0x00004000, // GONE!
        blDisclosure                    = 0x00008000,
        blBorderMetrics                 = 0x00010000,
        blDrawBorder                    = 0x00020000,
        blSortable                      = 0x00040000,
        blDrawSortButton                = 0x00080000,
        blTable                         = 0x00100000,
        blOnlyOne                       = 0x00200000,
        blUseSense                      = 0x00400000,
        blNoExtend                      = 0x00800000,
        blNoDisjoint                    = 0x01000000,
        blPinRightColumn                = 0x02000000 // NEW!
};
```

Constant Description

`blPinRightColumn`	Blugs tries to keep rightmost column pinned to the right edge.

Row Data Flags

The `blRowSelectAll` flag has been added. When a row has this property, clicking on any cell selects all cells in that row.

▲ **WARNING**
Setting up a list with rows having the `blRowSelectAll` flag, and setting up a list with a representative column – these are exactly opposite. In the former case you want one cell to select all, and in the latter case you want all cells to select one. Using both properties will undoubtedly give you strange results, particularly in marquee-selection. ▲

```
enum
{
        blRowHasChildren                = 0x0001,
        blRowIsExpanded                 = 0x0002,
        blRowIsTitleRow                 = 0x0004,
        blRowDrawBorder                 = 0x0008,
        blRowMarkedForMovement          = 0x0010,
        blRowSelectAll                  = 0x0020 // NEW!
};
```

Constant Descriptions

`blRowHasChildren`	The row has a disclosure triangle and zero or more children. Ignored if the `blDisclosure` list flag is clear.
`blRowIsExpanded`	The disclosure triangle points down. Ignored if the `blRowHasChildren` flag is clear, or if the `blDisclosure` list flag is clear.
`blRowIsTitleRow`	The row consists of a single cell that extends the full list width.
`blRowDrawBorder`	Draw a border at the bottom of this row.
`blRowMarkedForMovement`	Row will be moved in next call to `BLMoveMarkedRows`. Generally you will only use this flag in `BLSetRowFlags`;

	it doesn't make much sense to use it in a 'LiSt' resource, although you can if you want.
blRowSelectAll	When one cell in this row becomes selected, all cells in the row are selected.

Title Bar Info

This structure contains all of the title bar fields you may modify after the title bar has been created. The routines BLGetTitleBarInfo and BLSetTitleBarInfo use this kind of structure.

```
typedef struct
{
      BLContentType                 defaultContentType;
      UInt16                        thickness;
      UInt16                        whichTitleBarFlags;
      UInt16                        titleBarFlags;
      UInt16                        whichDefaultTitleFlags;
      UInt16                        defaultTitleFlags;
} BLTitleBarInfo;
```

Field Descriptions

defaultContentType	The content type to which new titles are initialized.
thickness	The title bar size. (Horizontal bar thickness is ignored under Aqua.)
whichTitleBarFlags	A mask indicating which of the titleBarFlags are applied.
titleBarFlags	New title bar flags; bits masked in by whichTitleBarFlags are applied.
whichDefaultTitleFlags	A mask indicating which of the defaultTitleFlags are applied.
defaultTitleFlags	The flags to which new titles are initialized; bits masked in by whichDefaultTitleFlags are applied.

Title Bar Info Field Flags

Use these flags to indicate to BLSetTitleBarInfo which fields of the BLTitleBarInfo structure you wish to apply to a title bar.

```
enum
{
      blSetTitleBarThickness          = 1 << 0,
      blSetTitleBarDefaultContentType = 1 << 1,
      blSetTitleBarFlags              = 1 << 2,
      blSetTitleBarDefaultTitleFlags  = 1 << 3
};
```

Constant Descriptions

blSetTitleBarThickness	Change the title bar thickness.
BlSetTitleBarDefaultContentType	Change the content type applied to new titles.
blSetTitleBarFlags	Change the bar's flags.
BlSetTitleBarDefaultTitleFlags	Change the flags applied to new titles.

Title Flags

These flags encode behavior of individual titles. A title bar contains a set of default title flags which are applied to new titles as rows or columns are added. You can modify individual titles by passing zero or more of these OR-combined flags to `BLSetTitleFlags`.

```
enum
{
      blTitleSelectable               = 0x0001
};
```

Constant Description

`blTitleSelectable`	Title bevel buttons can be clicked and selected by the user. When this feature is set, the bevel buttons have radio button behavior. By default titles cannot be selected.

Widget Info

This structure contains all of the widget fields you may modify after the widget has been created. The routines `BLGetWidgetInfo` and `BLSetWidgetInfo` use this kind of structure.

```
Typedef struct
{
      BLNotificationCommand           command;
      BLContentType                   content;
      SInt16                          size;
      UInt16                          whichWidgetFlags;
      UInt16                          widgetFlags;
} BLWidgetInfo;
```

Field Descriptions

`command`	The widget's notification command.
`content`	The widget's content type.
`size`	The widget's width or height.
`whichWidgetFlags`	A mask indicating which of the `widgetFlags` are to be applied.
`widgetFlags`	The new widget flags; bits masked in by `whichWidgetFlags` are applied.

Widget Info Field Flags

Use these flags to indicate to `BLSetWidgetInfo` which fields of the `BLWidgetInfo` structure you wish to apply to a widget.

```
enum
{
      blSetWidgetCommand              = 1 << 0,
```

```
        blSetWidgetContentType          = 1 << 1,
        blSetWidgetSize                 = 1 << 2,
        blSetWidgetFlags                = 1 << 3
};
```

Constant Descriptions

`blSetWidgetCommand`	Change widget's notification command.
`BlSetWidgetContentType`	Change widget's content type
`blSetWidgetSize`	Change widget's size.
`BlSetWidgetFlags`	Change widget's flags.

Deprecated and Deleted Routines

For various reasons these routines have been deprecated or removed in Blugs 1.2.

BLEnter (Deprecated under Carbon/OS X)

Explicitly checks environment and initializes Blugs.

```
OSErr BLEnter( void )
```

If you want to make sure the host machine can run Blugs, you can call `BLEnter` before you use other Blugs routines. If you do not explicitly call `BLEnter`, other routines (like `BLRegisterContentHandler`) will automatically call it. The routine is public because you may want to check its `OSErr` return value if your application may run on older machines/OSes. If you are developing exclusively for Carbon or (especially) OS X, you can let Blugs call this routine for you, since this degree of paranoia is probably unnecessary.

`BLEnter` makes some `Gestalt` checks: availability of Appearance Manager, Drag Manager, Control Manager version, and 32-bit `GWorld` capability. It then allocates a small hash table for storing content handler information.

If 32-bit `GWorld`s are not available, `BLEnter` returns `notInitErr`. *This will not happen with any PowerPC build.* If memory is so short that the hash table can't be allocated, it returns `memFullErr`, in which case your application is in serious memory trouble. If `BLEnter` returns an error, you must not use the Blugs API.

RESULT CODES

`notInitErr` (-900)	(68K only) Minimum system requirements not met.
`memFullErr` (-108)	Not enough memory.
`noErr` (0)	No error.

BLExit (Deleted)

`BLExit` has been removed because it was deemed unnecessary.

BLGetWidgetContentType, BLGetWidgetSize, BLGetWidgetFlags,

BLSetWidgetContentType, BLSetWidgetSize, BLGetWidgetFlags (Deleted)

These routines have been removed because the new routines `BLGetWidgetInfo` and `BLSetWidgetInfo` make the API simpler and more powerful.

BLGetHorizontalTitleBar, BLGetVerticalTitleBar (Deleted)

These routines, as well as the dummy definition of the `BLTitleBarRef`, have been removed. All references to title bars are now based on orientation (`Boolean inVertical`).

BLCredits (Deleted)

`BLCredits` has been removed because Blugs is going to kick DataBrowser's ass; why would anyone need any additional ego massage? :-)

Changed Routines

For various reasons these routines have been modified in Blugs 1.2.

BLNewTitleBar

Creates a title bar associated with a list.

```
OSErr BLNewTitleBar
   Boolean                inVertical
   const BLTitleBarInfo*  inInfo
   BlugsRef               inList
```

`inVertical`	Whether the bar is vertical or horizontal.
`inInfo`	A structure containing the title bar information. See the structure "Title Bar Info" above.
`inList`	The list which will contain the title bar.

`BLNewTitleBar` creates a title bar and associates it with a list. Blugs creates the title bar within the bounds of the list's view rectangle.

If `inList` already has a title bar of the same orientation, then the old title bar is deleted and replaced with the new one. If this happens, the content handler(s) for the old title bar's titles are called to dispose of data for individual titles.

RESULT CODES

`notInitErr` (-900)	Blugs is not initialized.
`nilHandleErr` (-109)	Bad list reference.
`memFullErr` (-108)	Could not allocate memory for title bar.
`noErr` (0)	No error.

BLSelectTitle

Selects a title in a title bar

```
OSErr BLSelectTitle
   Boolean              inVertical          Whether the bar is vertical or horizontal.
   UInt16               inTitle             The title to select.
   BlugsRef             inList              The list which contains the title bar.
```

Call `BLSelectTitle` to select `inTitle` and deselect any other selected title in the `inVertical` title bar. If `inVertical` is `false`, the list can be sorted, and the new selection was not selected before, Blugs makes `inTitle` the new primary sort column and sorts the list. This routine behaves exactly as if the user had clicked in `inTitle`. Note that it is not currently possible to pass zero for `inTitle`; you cannot do a "deselect all titles" operation.

RESULT CODES

`notInitErr` (-900)	Blugs is not initialized.
`inputOutOfBounds` (-190)	Title zero; nonexistent title.
`nilHandleErr` (-109)	Bad list or title bar reference.
`noErr` (0)	No error.

BLGetSelectedTitle

Returns the one-based index of the currently selected title in a title bar.

```
UInt16 BLGetSelectedTitle
   Boolean              inVertical          Whether the bar is vertical or horizontal.
   BlugsRef             inList              The list which contains the title bar.
```

`BLGetSelectedTitle` returns the one-based index of the currently selected title in the `inVertical` title bar. If the title bar does not contain a selection, `BLGetSelectedTitle` returns zero.

New Routines

These routines were added in Blugs 1.2.

BLGetDisclosureTriangleColumn

Returns the column in which disclosure triangles are drawn.

```
OSErr BLGetDisclosureTriangleColumn
   BlugsRef             inList              The list to query.
```

Call `BLGetDisclosureTriangleColumn` to determine the column in which Blugs draws disclosure triangles. The default is column one.

BLSetDisclosureTriangleColumn

Changes the column in which Blugs draws disclosure triangles

```
OSErr BLSetDisclosureTriangleColumn
   UInt16                  inColumn          The new disclosure triangle column.
   BlugsRef                inList            The list to change.
```

Call `BLSetDisclosureTriangleColumn` to set the column in which Blugs draws a list's disclosure triangles. This data is list-level, so disclosure triangles do not move with a column. The default is one, and if you delete columns such that the value goes out of range, Blugs resets the value to one.

▲ **WARNING**
A disclosure triangle will not be drawn in a title row if the list's disclosure triangle column is greater than one.

RESULT CODES

`notInitErr` (-900)	Blugs is not initialized.
`inputOutOfBounds` (-190)	Column does not exist.
`nilHandleErr` (-109)	Bad list reference.
`noErr` (0)	No error.

BLDrawHiliteRect

Draws a standard hilite rectangle.

```
void BLDrawHiliteRect
   Boolean                 inActive          Pass true if the host list is active, false
                                             if inactive.
   const Rect*             inRect            The rectangle in which Blugs draws
                                             hiliting.
```

Call `BLDrawHiliteRect` to have Blugs draw a standard hilite rectangle in the current port. Typically you will use this call from within a content handler, to draw a portion of cell contents in a selected/hilited state. Prior to Mac OS X 10.1, Blugs uses the `RGBColor` retrieved from `LMGetHiliteRGB` and draws a solid rectangle if `inActive` is `true`, or a rectangular outline if `inActive` is `false`. With Mac OS X 10.1 Blugs uses Theme brush constants (unless an error occurs) and always draws a solid rectangle in the appropriate color. Blugs uses the appropriate color as the background color, calls `EraseRect`, and restores the previous background color before returning.

By way of example, the content handler called "Finderesque" supplied with the Blugs distribution, has two tasks to do when drawing a selected cell. First it must draw a file icon, and then it must draw a file name. So when its cell is selected it must draw the icon in a selected state and then draw a hilited rectangle on which the file name is drawn. Rather than reinvent the wheel, Finderesque in Blugs 1.2 calls `BLDrawHiliteRect` to apply appropriate hiliting for the text sub-region of the cell.

BLDrawHiliteRgn

Draws a standard hilite region.

```
void BLDrawHiliteRgn
   Boolean              inActive        Pass true if the host list is active, false
                                        if inactive.
   RgnHandle            inRgn           The region in which Blugs draws
                                        hiliting.
```

Call `BLDrawHiliteRgn` to have Blugs draw a standard hilite region in the current port. Typically you will use this call from within a content handler, to draw a portion of cell contents in a selected/hilited state. Prior to Mac OS X 10.1, Blugs uses the `RGBColor` retrieved from `LMGetHiliteRGB` and draws a solid region if `inActive` is `true`, or an outline of the region if `inActive` is `false`. With Mac OS X 10.1 Blugs uses Theme brush constants (unless an error occurs) and always draws a solid region in the appropriate color. Blugs uses the appropriate color as the background color, calls `EraseRgn`, and restores the previous background color before returning.

BLSetDefaultRowFlags

Sets the flags that are copied into a new row when it is created.

```
OSErr BLSetDefaultRowFlags
   UInt16               inWhichFlags    A mask in which flag bits to be changed
                                        are set.
   UInt16               inFlags         The new set of flags.
   BlugsRef             inList          The list to change.
```

Call `BLSetDefaultRowFlags` to change the initial set of flags copied into a new row. Each list keeps track of its default row and column flags. These are initialized to zero. This function changes the row flags indicated by `inWhichFlags` to the settings in `inFlags`. For each bit in the `inWhichFlags` mask parameter, if the bit is set then the corresponding bit in the `inFlags` parameter is applied to Blugs' stored default value. Changing the default value makes it faster to add rows because you do not have to call `BLSetRowFlags` for each row.

Note
This function does not affect undocumented flags. All flag bits not documented here or in the interface files are reserved or used internally. ◆

RESULT CODES

`notInitErr` (-900)	Blugs is not initialized.
`nilHandleErr` (-109)	Bad list reference.
`noErr` (0)	No error.

BLSetDefaultColumnFlags

Sets the flags that are copied into a new column when it is created.

```
OSErr BLSetDefaultColumnFlags
   UInt16                inWhichFlags        A mask in which flag bits to be changed
                                             are set.
   Uint16                inFlags             The new set of flags.
   BlugsRef              inList              The list to change.
```

Call `BLSetDefaultColumnFlags` to change the initial set of flags copied into a new column. Each list keeps track of its default row and column flags. These are initialized to zero. This function changes the column flags indicated by `inWhichFlags` to the settings in `inFlags`. For each bit in the `inWhichFlags` mask parameter, if the bit is set then the corresponding bit in the `inFlags` parameter is applied to Blugs' stored default value. Changing the default value makes it faster to add columns because you do not have to call `BLSetColumnFlags` each time.

Note
This function does not affect undocumented flags. All flag bits not documented here or in the interface files are reserved or used internally. ◆

RESULT CODES

`notInitErr` (-900)	Blugs is not initialized.
`nilHandleErr` (-109)	Bad list reference.
`noErr` (0)	No error.

BLGetTitleBarInfo

Gets information on a title bar.

```
OSErr BLGetTitleBarInfo
   Boolean               inVertical          Whether the bar is vertical or horizontal.
   BLTitleBarInfo*       outInfo             A structure that will contain the title bar
                                             information. See the structure "Title Bar
                                             Info" above.
   BlugsRef              inList              The list which contains the title bar.
```

`BLGetTitleBarInfo` copies the title bar's information into a structure you provide.

RESULT CODES

`notInitErr` (-900)	Blugs is not initialized.
`nilHandleErr` (-109)	Bad list or title bar reference.
`noErr` (0)	No error.

BLSetTitleBarInfo

Sets information on a title bar.

```
void BLSetTitleBarInfo
   Boolean                inVertical      Whether the bar is vertical or horizontal.
   UInt16                 inWhichFields   A mask with 1-bits indicating which
                                          fields are applied. See the enumeration
                                          "Title Bar Info Field Flags" above.
   const BLTitleBarInfo*  inInfo          A structure containing the new title bar
                                          information. See the structure "Title Bar
                                          Info" above.
   BlugsRef               inList          The list which contains the title bar.
```

`BLSetTitleBarInfo` modifies zero or more of the title bar's internal fields with the data you pass. For each bit set in `inWhichFields`, the corresponding field is copied into the title bar's internal representation. Blugs updates the list onscreen if necessary.

RESULT CODES

`notInitErr` (-900)	Blugs is not initialized.
`nilHandleErr` (-109)	Bad list or title bar reference.
`noErr` (0)	No error.

BLGetWidgetInfo

Gets information on a widget.

```
OSErr BLGetWidgetInfo
   Boolean                inVertical      Whether the widget is vertical or
                                          horizontal.
   BLWidgetInfo*          outInfo         A structure containing the widget
                                          information. See the structure "Widget
                                          Info" above.
   BlugsRef               inList          The list which contains the widget.
```

`BLGetTitleBarInfo` copies the widget's information into the structure you provide.

RESULT CODES

`notInitErr` (-900)	Blugs is not initialized.
`nilHandleErr` (-109)	Bad list or widget reference.
`noErr` (0)	No error.

BLSetWidgetInfo

Sets information on a widget.

```
void BLSetWidgetInfo
```

`Boolean`	`inVertical`	Whether the widget is vertical or horizontal.
`UInt16`	`inWhichFields`	A mask with 1-bits indicating which fields are applied. See the enumeration "Title Bar Info Field Flags" above.
`const BLWidgetInfo*`	`inInfo`	A structure containing the new widget information. See the structure "Widget Info" above.
`BlugsRef`	`inList`	The list which contains the widget.

`BLSetWidgetInfo` modifies zero or more of the widget's internal fields with the data you pass in. For each bit set in `inWhichFields`, the corresponding field is copied into the widget's internal representation. Blugs updates the list onscreen if necessary.

RESULT CODES

`notInitErr` (-900)	Blugs is not initialized.
`nilHandleErr` (-109)	Bad list or widget reference.
`noErr` (0)	No error.

The `'LiSt'` Resource

With the advent of the new APIs covered above, and others (row and column [default] info) still in development, it is becoming increasingly clear that the `'LiSt'` resource format – while useful since its inception – was not sufficiently well-designed to handle the kinds of API changes Blugs is undergoing. The same may be said of almost *any* resource format.

So it is with some reluctance that I declare this resource format officially **deprecated**. All support for reading `'LiSt'` resources will be removed from future versions of Blugs, and no further modifications will be made to the current code base except to fix critical bugs.

Developers may find a cleaner solution by stripping the resource format down to its fixed-length header portion (the stuff that gets sent to `BLNew`) and then using auxiliary resources or data files to recreate variable-length fields such as cell data. OTOH, the Blugs API is becoming sufficiently powerful so that creating the entire list programmatically is not so painful. I hope to work with Blugs adopters to find the best solution – which will probably have something to do with XML!

What's on the Horizon

I am sitting on a huge set of desired features, many of which have been successfully implemented in the so-called Blugs 2.0 project. Most of these features will be rolled over into Blugs 1.X in order of desirability. If one of these ideas is particularly exciting to you, please drop me a line and we'll see if we can't work together to get it implemented quickly and elegantly.

- Default cell info – including content type (the spreadsheet/table distinction is going away!)
- Carbon control/HIView implementation (partially done!)
- Quartz support (partially done!)
- Content types predefined as `OSType`s by handler header files.
- Replace various common data infrastructures with CoreFoundation primitives.

- Retool content handler API to support CarbonEvents cleanlt, and as the preferred event handling architecture.
- 32-bit row and column numbers.
- Removal of "nth flavor data" APIs and handler messages.
- Idling via Carbon timers.
- Immediate, rather than deferred, inline editing.