in

COLLABORATORS			
	TITLE : in		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 31, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

in

1.1 listviewclass.guide

```
 Search
TABLE OF CONTENTS

listviewclass/--background--
listviewclass/BASE_DRAGACTIVE
listviewclass/BASE_DRAGQUERY
listviewclass/BASE_DRAGUPDATE
listviewclass/BASE_DROPPED
listviewclass/BASE_FREEDRAGOBJECT
listviewclass/BASE_GETDRAGOBJECT
listviewclass/LVM_ADDENTRIES
listviewclass/LVM_ADDSINGLE
listviewclass/LVM_CLEAR
listviewclass/LVM_INSERTENTRIES
listviewclass/LVM_INSERTSINGLE
listviewclass/LVM_MOVE
listviewclass/LVM_REDRAW
listviewclass/LVM_REFRESH
listviewclass/LVM_REMENTRY
listviewclass/LVM_REMSELECTED
listviewclass/LVM_REPLACE
listviewclass/LVM_SORT
listviewclass/LVM_[UN]LOCKLIST
listviewclass/LVM_[xxx]ENTRY
listviewclass/LISTV_CompareHook
listviewclass/LISTV_CustomDisable
listviewclass/LISTV_DeSelect
listviewclass/LISTV_DisplayHook
listviewclass/LISTV_DropSpot
listviewclass/LISTV_Entry
listviewclass/LISTV_EntryArray
listviewclass/LISTV_EntryNumber
listviewclass/LISTV_LastClicked
listviewclass/LISTV_LastClickedNum
listviewclass/LISTV_LastColumn
listviewclass/LISTV_ListFont
listviewclass/LISTV_MakeVisible
listviewclass/LISTV_MinEntriesShown
```

1.2 listviewclass/--background--

NAME
Class: listviewclass
Superclass: baseclass
Include File: <libraries/bgui.h>

FUNCTION
To provide a gadget simular to the gadtools.library's listview kind. The lisview class does however have extended functionality like hooks for entry creation, entry comparrison, entry and title rendering. Also a multi-selection mode is available. Opposed to the gadtools version this class does not require the usage of list and nodes. All kinds of data can be added to the listview as entries providing that you supply hook routines to handle this data.

Objects from this class send out the following attribute pairs in their notification events:

GA_ID - Gadget object ID.
LISTV_Entry - Pointer to the selected entry.
LISTV_EntryNumber - Logical number of the selected entry.
LISTV_LastColumn - Last column clicked on.

NOTE
Most of the methods described below can also contain a pointer to a GadgetInfo structure. This pointer does not have to be valid. All actions will be done only if you want to let the action also be shown visually you need to pass a valid pointer to a GadgetInfo structure.

1.3 listviewclass/BASE_DRAGACTIVE

NAME
BASE_DRAGACTIVE -- This method overides the baseclass method. To show the user that the object is the active drop target it will

render a dotted box around the view area instead of a dotted box arround the hitbox area.

1.4 listviewclass/BASE_DRAGQUERY

NAME
BASE_DRAGQUERY -- This method overides the baseclass method. It will return BQR_ACCEPT when:

1) The request came from itself. I.E. It will only accept drops from itself.
2) The LISTV_ShowDropSpot attribute is set to TRUE.
3) The mouse location is inside the view area, not inside the scroller object.

Take a look at the supplied demo programs to see how you can overide this behaviour.

1.5 listviewclass/BASE_DRAGUPDATE

NAME
BASE_DRAGUPDATE -- This method overides the baseclass method. When the LISTV_ShowDropSpot attribute is set to TRUE and the user is dragging the entries over the object, the place at which they can drop the entries is continually updated by rendering a dotted line at that spot.

1.6 listviewclass/BASE_DROPPED

NAME
BASE_DROPPED -- This method overides the baseclass method. When the user has dropped the entries this method will move them to the location where they were dropped.

Take a look at the supplied demo programs to see how you can overide this behaviour.

1.7 listviewclass/BASE_FREEDRAGOBJECT

NAME
BASE_FREEDRAGOBJECT -- This method overides the baseclass method. It simply deallocated the stuff which was setup by the BASE_GETDRAGOBJECT method.

1.8 listviewclass/BASE_GETDRAGOBJECT

```
  NAME
BASE_GETDRAGOBJECT -- This method overides the baseclass method.  When
  the user starts dragging the selected entries this method will
  create a BitMap in which the dragged entries are displayed.

  Up to ten selected entries are shown in the dragged list. When
  there were more than ten entries the  dragged  list  will show
  the following entries:

  First Selected Entry
  -->
  Last Selected Entry
```

1.9 listviewclass/LVM_ADDENTRIES

```
  NAME
LVM_ADDENTRIES -- Add multiple entries.

  SYNOPSIS
err = DoMethod( obj, LVM_ADDENTRIES, gi, entries, how )

ULONG     err;
struct GadgetInfo      *gi;
APTR           *entries;
ULONG     how;

  FUNCTION
This method  can be used to add more than one entry after the listview
object has been created.

  INPUTS
gi  - A pointer to the GadgetInfo structure or NULL.
entries - This  must  point  to a NULL-terminated array of pointers to
  the entries to add.
how - Here  you  can  select  where  the  entries  are  added. The
  following positions are possible:

  LVAP_HEAD -- The entries are added at the top of the list.
  LVAP_TAIL -- The entries are added at the bottom of the list.
  LVAP_SORTED -- The entries  are added sorted according to  the
    sorting  method active. In the attributes  section  of
    this documentation you can find more about the sorting
    possibilities.

  RESULT
err - TRUE  uppon  succes  and FALSE if one or more of the entries
  failed to be added.

  SEE ALSO
LVM_ADDSINGLE, LVM_REMENTRY, LISTV_CompareHook
```

1.10 listviewclass/LVM_ADDSINGLE

```
  NAME
LVM_ADDSINGLE --  Add a single entry.

  SYNOPSIS
err = DoMethod( obj, LVM_ADDSINGLE, gi, entry, how, flags )

ULONG     err;
struct GadgetInfo      *gi;
APTR      entry;
ULONG     how;
ULONG     flags;

  FUNCTION
This method can be used to add a single entry to the  listview  object
after it has been created.

  INPUTS
gi  - A pointer to the GadgetInfo structure.
entry - This must point to the entry which needs to be added  to the
  listview object.
how - Please   refer   to  the  LVM_ADDENTRIES  section  for  more
  information on the ways you can add entries.
flags - Any of the following flags can be set here:

  LVASF_MAKEVISIBLE -- This tell's the lisview object to  scroll
    the  list  to  make  the  added  entry visible.

  LVASF_SELECT -- This  tell's  the  listview object to make the
    added entry selected.  This  will  also  automatically
    scroll  the  list  to  make the  added   entry visible
    unless the LVASF_NOT_VISIBLE flag is set.

  LVASF_MULTISELECT ** V40 ** -- This  flag only works on multi-
    select listviews.   When   set   the   added  entry is
    selected without  unselecting already selected entries
    in the list.   This flag will also scroll the list  to
    make   the    added    entry  visible    unless  the
    LVASF_NOT_VISIBLE flag is set.

  LVASF_NOT_VISIBLE ** V40 ** -- When  set  in  combination with
    the  LVASF_SELECT or LVASF_MULTISELECT flags the added
    entry  is  selected  but  not made visible.

  RESULT
err - TRUE uppon success, FALSE uppon failure.

  SEE ALSO
LVM_ADDENTRIES, LVM_REMENTRY
```

1.11 listviewclass/LVM_CLEAR

```
  NAME
```

```
LVM_CLEAR -- Delete all entries.

  SYNOPSIS
DoMethod( obj, LVM_CLEAR, gi )

struct GadgetInfo *gi;

  FUNCTION
This  method must be used to clear and delete all  entries  present in
the list.

  INPUTS
gi  - A pointer to the GadgetInfo structure.

  RESULT
Return code is not defined.
```

1.12 listviewclass/LVM_INSERTENTRIES

```
  NAME
LVM_INSERTENTRIES ** V40 ** -- Insert several entries.

  SYNOPSIS
err = DoMethod( obj, LVM_INSERTENTRIES, gi, pos, entries )

ULONG     err;
struct GadgetInfo      *gi;
ULONG     pos;
APTR           *entries;

  FUNCTION
This method is basically the same as the  LVM_ADDENTRIES  method  with
the  exception that you can specify at which position the entries will
be inserted  in the list.

  INPUTS
gi  - A pointer to the GadgetInfo structure.

pos - This must be the numeric position at which you want to  have
  the entries inserted. The  numbers  can  range from 0 to the
  amount of entries already in the list. If you specify a number
  larger than  the amount of  entries  already in the  list  the
  entries  will be appended to the existing entries.

entries - A  pointer  to  a NULL-terminated array of entry pointers to
  add to the list.

  RESULT
err - TRUE  uppon  succes  and FALSE if one or more of the entries
  failed to be added.

  SEE ALSO
LVM_ADDENTRIES, LVM_INSERTSINGLE, LVM_REMENTRY
```

1.13 listviewclass/LVM_INSERTSINGLE

```
  NAME
LVM_INSERTSINGLE ** V40 ** -- Insert a single entry.

  SYNOPSIS
err = DoMethod( obj, LVM_INSERTSINGLE, gi, pos, entry, flags )

ULONG     err;
struct GadgetInfo      *gi;
ULONG     pos;
APTR      entry;
ULONG     flags;

  FUNCTION
This  method  should  be  used  to  insert  a single  entry at a given
position in the list.  It is basically the same as  the  LVM_ADDSINGLE
method with the exception that the position of the entry is specified.

  INPUTS
gi  - A pointer to the GadgetInfo structure.

pos - This must be the  numeric  position  at  which you  want  to
  have  the entry inserted.  The numbers can range from 0 to the
  amount of entries already in the list. If you specify a number
  larger  than  the amount of entries already  in the  list  the
  entry will be appended to the existing entries.

entry - A pointer to the entry to insert.

flags - Special flags which  you  can use to make the inserted entry
  visible,  select it,  multi-select  it  etc.   For  a complete
  description  of  the  possibilities  read  the   LVM_ADDSINGLE
  section.

  RESULT
err - TRUE uppon succes and FALSE uppon failure.

  SEE ALSO
LVM_ADDSINGLE, LVM_INSERTENTRIES, LVM_REMENTRY
```

1.14 listviewclass/LVM_MOVE

```
  NAME
LVM_MOVE ** V38 ** -- Move an entry.

  SYNOPSIS
succ = DoMethod( obj, LVM_MOVE, gi, entry, dir, new )

ULONG     succ;
struct GadgetInfo      *gi;
APTR      entry;
ULONG     dir;
ULONG     new;
```

```
  FUNCTION
This method must be used to move entries in the list.

  INPUTS
gi  - A pointer to the GadgetInfo structure.

entry - This  can  point to the specific entry you want to move.  If
  you specify NULL here the  selected  entry is moved.

dir - Here you can specify the direction in  which the entry  must
  be moved.  The following directions are possible:

  LVMOVE_UP     -- Move the entry one place up.
  LVMOVE_DOWN   -- Move the entry one place down.
  LVMOVE_TOP    -- Move the entry to the list-top.
  LVMOVE_BOTTOM -- Move the entry to the list-bottom.
  LVMOVE_NEWPOS -- Move the entry to lvmm_NewPos. ** V40 **

new - This field was added in V40 of the library.  Do  not  use it
  on lower  versions.   This  field  must  contain  the  ordinal
  position number to where the entry is moved.  It  is  used  in
  combination with the LVMOVE_NEWPOS direction constant.

  NOTE
When  the  entry actually moved the class will send out a notification
message with the following attributes:

GA_ID     -- The ID of the object.
LISTV_NewPosition -- The new ordinal position of the entry.

  RESULT
succ  - TRUE when the entry moved and FALSE if not.
```

1.15 listviewclass/LVM_REDRAW

```
  NAME
LVM_REDRAW ** V40 ** -- Redraw the list contents.

  SYNOPSIS
DoMethod( obj, LVM_REDRAW, gi )

struct GadgetInfo *gi;

  FUNCTION
This method is basically the same as the LVM_REFRESH  method described
above  with  the  exception  that,  instead  of  refreshing  the whole
listview,  only  the  entries which are visible are refreshed.

  INPUTS
gi  - A pointer to the GadgetInfo structure.

  RESULT
Return code is not defined.
```

```
  SEE ALSO
LVM_REFRESH
```

1.16 listviewclass/LVM_REFRESH

```
  NAME
LVM_REFRESH -- Refresh the listview.

  SYNOPSIS
DoMethod( obj, LVM_REFRESH, gi )

struct GadgetInfo *gi;

  FUNCTION
This method must be used to refresh the listview object after some
changes have been made which where not visible. In some cases it might
be usefull to add entries without passing a GadgetInfo structure along
with the adding methods. This will speed up the adding and you can
show the changes when you are done by sending this method to the
listview object.

  INPUTS
gi - A pointer to the GadgetInfo structure. Should be valid
  otherwise this method is not really useful.

  RESULT
Return code is not defined.

  SEE ALSO
LVM_REDRAW
```

1.17 listviewclass/LVM_REMENTRY

```
  NAME
LVM_REMENTRY -- Remove an entry.

  SYNOPSIS
DoMethod( obj, LVM_REMENTRY, gi, entry )

struct GadgetInfo *gi;
APTR       entry;

  FUNCTION
This method must be used to remove a single entry from the listview
object.

  INPUTS
gi - A pointer to the GadgetInfo structure.
entry - This must point to the entry you want to remove.

  RESULT
Return code is not defined.
```

```
  SEE ALSO
LVM_ADDENTRIES, LVM_ADDSINGLE
```

1.18 listviewclass/LVM_REMSELECTED

```
  NAME
LVM_REMSELECTED ** V40 ** -- Remove selected entry.

  SYNOPSIS
succ = DoMethod( obj, LVM_REMSELECTED, gi )

ULONG      succ;
struct GadgetInfo *gi;

  FUNCTION
This method can  be used to remove the currently selected  entry  from
the list and automatically select the next or previous one.

  INPUTS
gi  - A pointer to the GadgetInfo structure.

  NOTE
This method  only operates on single-select listviews. On multi-select
listviews this method has no effect.

  RESULT
succ  - TRUE if an entry was removed and FALSE if not.
```

1.19 listviewclass/LVM_REPLACE

```
  NAME
LVM_REPLACE ** V39 ** -- Replace an entry by another.

  SYNOPSIS
rep = DoMethod( obj, LVM_REPLACE, gi, old, new )

APTR       rep;
struct GadgetInfo *gi;
APTR       old;
APTR       new;

  FUNCTION
This method allows you to replace an existing entry with another.

  INPUTS
gi  - A pointer to the GadgetInfo structure.

old - This must be a pointer to the entry you want to replace.

new - This must point to the new  data you want to replace the old
  entry by.
```

```
  RESULT
rep - A  pointer  to  the  new  entry uppon success and NULL uppon
  failure.
```

1.20 listviewclass/LVM_SORT

```
  NAME
LVM_SORT -- Sort all entries.

  SYNOPSIS
DoMethod( obj, LVM_SORT, gi )

struct GadgetInfo *gi;

  FUNCTION
Calling this method  will force a complete re-sorting of  the  entries
in the list. This can be handy  when your comparisson  hook (described
in  the  attributes  LISTV_CompareHook  section)  can handle different
kinds of comparissons.

  INPUTS
gi  - A pointer to the GadgetInfo structure.

  RESULT
Return code is not defined.
```

1.21 listviewclass/LVM_[UN]LOCKLIST

```
  NAME
LVM_LOCKLIST, LVM_UNLOCKLIST -- (Un)lock the listview.

  SYNOPSIS
DoMethod( obj, LVM_LOCKLIST )
DoMethod( obj, LVM_UNLOCKLIST, gi )

struct GadgetInfo *gi;

  FUNCTION
These methods must  be used to lock or unlock the list contents. When,
for example, you must change the text of a list entry you should  lock
it  using  the  LVM_LOCKLIST  method  and  when you are done unlock it
using the LVM_UNLOCKLIST method.

This locking is only necessary when you are manipulating  the contents
of  the  list  entries by hand. Changing the list contents with any of
the other listview methods do not require you to lock the list.

  INPUTS
gi  - A pointer to the GadgetInfo structure.

  RESULT
```

```
No return code is defined.
```

1.22 listviewclass/LVM_[xxx]ENTRY

```
  NAME
LVM_FIRSTENTRY, LVM_LASTENTRY, LVM_NEXTENTRY, LVM_PREVENTRY

  SYNOPSIS
entry = DoMethod( obj, LVM_FIRSTENTRY, NULL, flags )
entry = DoMethod( obj, LVM_LASTENTRY,  NULL, flags )
entry = DoMethod( obj, LVM_NEXTENTRY,  prev, flags )
entry = DoMethod( obj, LVM_PREVENTRY,  prev, flags )

APTR      entry;
APTR      prev;
ULONG     flags;

  FUNCTION
These methods must be used to  itterate  through  all  entries  in the
listview. You can itterate through the entries one by one or  only the
selected ones.

  INPUTS
prev  - For  the  LVM_FIRSTENTRY and LVM_LASTENTRY methods this must
  be NULL. For the LVM_NEXTENTRY and LVM_PREVENTRY  this  should
  point to the entry returned by a previous call to any of these
  methods.

flags - Any of the following flags may be set here:

  LVGEF_SELECTED -- The methods  will  only scan  for selected
    entries when this bit is set. All non-selected entries
    will simply be skipped.

  RESULT
entry - A pointer  to  the  entry or NULL  when  no more entries are
  available.


  EXAMPLE
/*
 *  Scan through all entries in
 *  the listview gadget starting
 *  at the first one.
 */
Object    *listview;
APTR     entry;

/*
 *  Get first entry.
 */
if ( entry = (APTR)DoMethod(
           listview, LVM_FIRSTENTRY, NULL, 0L )) {
  /*
   *  Loop through the rest of the list.
```

```
   */
  do {
    /*
     *  Print the entry...
     */
    printf( "Entry = %s\n", entry );

    /*
     *  Next entry...
     */
    entry = (APTR)DoMethod(
       listview, LVM_NEXTENTRY, entry, 0L );
  } while ( entry );
}
```

1.23 listviewclass/LISTV_CompareHook

```
  NAME
LISTV_CompareHook -- ( struct Hook * )

  FUNCTION
To add a hook routine  that will  compare two  entries with eachother.
As it is possible  to have  entries  which are  different  from simple
strings you can perform your own comparison here.  The comparison hook
is called each time an entry is added  sorted or when  the list is re-
sorted. The hook routine will be called as follows:

rc = hookFunc( hook, object, message );
D0         A0    A2      A1

LONG       rc;
struct Hook   *hook;
Object      *object;
struct lvCompare  *message;

The message argument is a pointer to the following data structure:

struct lvCompare {
  APTR    lvc_EntryA;
  APTR    lvc_EntryB;
};

lvc_EntryA, lvc_EntryB -- These  are the entries that must be compared
  to eachother.

The internal comparison routine simple  does a  stricmp()  on the  two
entry strings.

This hook must return -1 when entry a is smaller than entry b, 0  when
entry a is equal to entry b and 1 when entry a is bigger than entry b.

  DEFAULT
NULL (internal comparison routine).

  APPLICABILITY
```

```
(I).

  SEE ALSO
LISTV_ResourceHook, LISTV_DisplayHook
```

1.24 listviewclass/LISTV_CustomDisable

```
  NAME
LISTV_CustomDiable -- ( BOOL ) ** V40 **

  FUNCTION
To  tell  the  class  rendering routine that the custom rendering hook
will take care of the disabled rendering itself.  If  set to FALSE the
class itself will render a ghosting pattern over the list entries.

Note that this tag only affects objects which have a  custom rendering
hook installed with LISTV_DisplayHook.

  DEFAULT
FALSE.

  APPLICABILITY
(IS).

  SEE ALSO
LISTV_DisplayHook
```

1.25 listviewclass/LISTV_DeSelect

```
  NAME
LISTV_DeSelect -- ( ULONG ) ** V39 **

  FUNCTION
To deselect a selected entry. The data you pass is the ordinal  number
of the entry starting at 0 for the  first  entry in the list.  If  you
supply  a  value  of  ~0  (-1) all  selected  entries  in the list are
deselected.

  APPLICABILITY
(SU)

  SEE ALSO
LISTV_Select
```

1.26 listviewclass/LISTV_DisplayHook

```
  NAME
LISTV_DisplayHook -- ( struct Hook * )

  FUNCTION
```

To add a hook routine that will take care of rendering the entries. In some cases it is necessary to do your own rendering. This hook is called for each entry that needs to be rendered. The hook routine will be called as follows:

```
rc = hookFunc( hook, object, message );
D0         A0    A2      A1

VOID     rc;   /* No return code defined. */
struct Hook *hook;
Object    *object;
struct lvRender *message;
```

The message argument is a pointer to the following data structure:

```
struct lvRender {
  struct RastPort     *lvr_RPort;
  struct DrawInfo     *lvr_DrawInfo;
  struct Rectangle    *lvr_Bounds;
  APTR         lvr_Entry;
  UWORD        lvr_State;
  UWORD        lvr_Flags;
};
```

lvr_RPort -- This is a pointer to the RastPort in which the rendering must be done. Please note that the font you must use to render text is already set up for you. It is not recommendable to use another font than the one set in this RastPort because the height of the area you may render in is setup accoording to this font.

lvr_DrawInfo -- This can point to a DrawInfo structure as defined in <intuition/screens.h> in which the necessay information about the display environment is stored. Note that it is possible that this is NULL. It is not very likely but it is possible.

lvr_Bounds -- This is a struct Rectangle in which the area you should render in is defined. Do _not_ render outside the given bounds or you will seriously screw up the display! Also keep in mind that the area you are rendering into is not always cleared. In other words, the area may still show data from another entry. You must make sure you completely re-render the given bounds.

lvr_Entry -- This points to the entry data as setup by the entry creation hook or the built-in entry creation.

lvr_State -- This describes the state in which to render the entry. The state is one of the following possibilities:

LVRS_NORMAL -- Normal rendering. Render the entry in a normal, un-selected way.
LVRS_SELECTED -- Selected rendering. Render the entry in a selected way.
LVRS_NORMAL_DISABLED -- Normal, disabled rendering. Render the entry in a normal way but make it disabled. This is normally done by ghosting it with a pattern (see below).

```
  LVRS_SELECTED_DISABLED -- Selected, disabled rendering. Render
    the entry is a selected way but make it disabled. This
    is normally done by ghosting  it with  a  pattern (see
    below).

  Ghosting the entry is usually done like this:

  struct lvRender         *lvr;
  UWORD     *pens = lvr->lvr_DrawInfo->dri_Pens;
  UWORD      patt = { 0x2222, 0x8888 };

  SetAPen( lvr->lvr_RPort, pens[ SHADOWPEN ] );
  SetDrMd( lvr->lvr_RPort, JAM1 );
  SetAfPt( lvr->lvr_RPort, patt, 1 );

  RectFill( lvr->lvr_RPort, lvr->lvr_Bounds.MinX,
          lvr->lvr_Bounds.MinY,
          lvr->lvr_Bounds.MaxX,
          lvr->lvr_Bounds.MaxY );

  Please keep in mind that, although the above code doesn't show
  it, the lvr_DrawInfo field can be NULL.

lvr_Flags -- No flags are defined yet.

When this hook is not set the internal rendering  routine will  simply
render a string which is created in  the  LIST_ResourceHook.  When the
LISTV_RenderHook routine creates something other than a  simple string
pointer you must provide a display hook to render the entries.

Most  of  the  time  when  you  add  more  than a simple string to the
listview  object  the  data  you add is a structure which contains the
string  and  some  extra  data.  To prevent you from having to write a
display-hook  to  render the string your hook can also simply return a
pointer  to  the  string and the listviewclass will render it for you.
I.E.:

struct myStruct {
  UBYTE   *string;
  UWORD    some_more_data;
};

__saveds __asm hookFunc( register __a0 struct Hook     *hook,
       register __a2 Object        *lv_obj,
       register __a1 struct lvRender *lvr )
{
     return((( struct myNode * )lvr->lvr_Entry )->string );
}

This  hook  will  let the listviewclass dispatcher render the returned
string  for  you while keeping the extended data available for you. If
your  hook  returns  NULL  the listviewclass assumes you have done all
rendering required.

  DEFAULT
NULL (internal entry rendering).
```

APPLICABILITY
(I).

SEE ALSO
LISTV_ResourceHook, LISTV_CompareHook, LISTV_TitleHook

1.27 listviewclass/LISTV_DropSpot

NAME
LISTV_DropSpot -- (ULONG) ** V40 **

FUNCTION
To query the position at which the dragged entries were dropped. When the LISTV_ShowDropSpot was set to TRUE and the user has dragged some entries in this listview this attribute will hold the ordinal list position where the entries were dropped. This attribute is _only_ usefull when queried in a BASE_DROPPED method.

APPLICABILITY
(G).

SEE ALSO
baseclass.doc/BASE_DROPPED, LISTV_ShowDropPos

1.28 listviewclass/LISTV_Entry

NAME
LISTV_Entry -- (APTR)

FUNCTION
This tag is sent during notification. The data field is a pointer to the entry which triggered the notification.

APPLICABILITY
(N).

SEE ALSO
LISTV_EntryNumber

1.29 listviewclass/LISTV_EntryArray

NAME
LISTV_EntryArray -- (APTR *)

FUNCTION
To add a set of entries at initialization time. The data is a pointer to a NULL-terminated array of entries which need to be added to the listview object.

DEFAULT

```
NULL.

  APPLICABILITY
(I).

  SEE ALSO
LISTV_SortEntryArray
```

1.30 listviewclass/LISTV_EntryNumber

```
  NAME
LISTV_EntryNumber -- ( ULONG )

  FUNCTION
This  tag  is  sent during notification. The data field is the logical
number of the entry which triggered the notification.

  APPLICABILITY
(N).

  SEE ALSO
LISTV_Entry
```

1.31 listviewclass/LISTV_LastClicked

```
  NAME
LISTV_LastClicked -- ( APTR )

  FUNCTION
To get a pointer to the last selected entry.  This data can be used to
detect double-clicking and entry.

  EXAMPLE
Object    *listview;
ULONG    ds[2], dm[2], last = 0, clicked;

GetAttr( LISTV_LastClicked, listview, &clicked );
if ( clicked == last ) {
  CurrentTime( &ds[ 1 ], &dm[ 1 ] );
  if ( DoubleClick( ds[ 0 ], dm[ 0 ], ds[ 1 ], dm [ 1 ] )) {
    /* Double clicked */
    ...
  }
}
CurrentTime( &ds[ 0 ], &dm[ 0 ] );
last = clicked;

  APPLICABILITY
(G).

  SEE ALSO
LISTV_LastClickedNum
```

1.32 listviewclass/LISTV_LastClickedNum

```
  NAME
LISTV_LastClickedNum -- ( ULONG ) ** V38 **

  FUNCTION
To return the number of the last selected entry.

  APPLICABILITY
(G).

  SEE ALSO
LISTV_LastClicked
```

1.33 listviewclass/LISTV_LastColumn

```
  NAME
LISTV_LastColumn -- ( ULONG )  ** V41 **

  FUNCTION
To get the number of the last column clicked on.  This is useful if
you need to do different things depending on the column clicked on.

  APPLICABILITY
(G).

  SEE ALSO
LISTV_LastClicked, LISTV_LastClickedNum
```

1.34 listviewclass/LISTV_ListFont

```
  NAME
LISTV_ListFont -- ( struct TextAttr )

  FUNCTION
To set the font which is used to  render the  entries.  By default the
font  used  to  render  the  entries is the same font which is used to
render the object it's label. This font might be proportional. In some
cases  it might be useful to have a mono-space font for the entries or
even another proportional font.

  DEFAULT
NULL.

  APPLICABILITY
(IG)
```

1.35 listviewclass/LISTV_MakeVisible

```
  NAME
LISTV_MakeVisible -- ( ULONG )

  FUNCTION
To scroll the list to make the entry appear in the display area of the
listview object. The data required is the logical number of  the entry
in the list starting with 0 as the first entry.

  APPLICABILITY
(SU).
```

1.36 listviewclass/LISTV_MinEntriesShown

```
  NAME
LISTV_MinEntriesShown -- ( UWORD )

  FUNCTION
To specify how many entries should be visible at all times.  Note: The
larger this value the bigger the object it's minimum size.

  DEFAULT
3.

  APPLICABILITY
(I).
```

1.37 listviewclass/LISTV_MultiSelect

```
  NAME
LISTV_MultiSelect -- ( BOOL )

  FUNCTION
To make the listview a multi-selection object. Multi-selection objects
allow the user to select more than one entry from the list.

  DEFAULT
FALSE.

  APPLICABILITY
(ISU).
```

1.38 listviewclass/LISTV_MultiSelectNoShift

```
  NAME
LISTV_MultiSelectNoShift -- ( BOOL ) ** V39 **

  FUNCTION
To allow the user to multi-(de)select the entries in a multi-selection
object  without  having  to use the SHIFT key. This tag is only useful
```

when the LISTV_MultiSelect tag is set to TRUE.

DEFAULT
FALSE.

APPLICABILITY
(ISU).

SEE ALSO
LISTV_MultiSelect

1.39 listviewclass/LISTV_NewPosition

NAME
LISTV_NewPosition -- (ULONG) ** V38 **

FUNCTION
To notify the object it's targets of the entry it's new position number. When you move an entry with the LVM_MOVE method the object will send out a notification message with this attribute.

Since V40 of the library this attribute is also gettable with OM_GET. Please note however that getting this attribute will only give the desired result _after_ you moved an entry with LVM_MOVE.

APPLICABILITY
(NG).

SEE ALSO
LVM_MOVE

1.40 listviewclass/LISTV_NumEntries

NAME
LISTV_NumEntries (ULONG) ** V38 **

FUNCTION
To return the number of entries in the list.

APPLICABILITY
(G).

1.41 listviewclass/LISTV_ReadOnly

NAME
LISTV_ReadOnly -- (BOOL)

FUNCTION
To make the listview a read-only object. Read only objects have full functionality except for the entries which cannot be selected.

```
  DEFAULT
FALSE.

  APPLICABILITY
(I).
```

1.42 listviewclass/LISTV_ResourceHook

```
  NAME
LISTV_ResourceHook -- ( struct Hook * )

  FUNCTION
To add a hook routine that will build or delete a listview entry.  The
hook routine will be called as follows:

rc = hookFunc( hook, object, message );
D0         A0    A2      A1

APTR     rc;
struct Hook *hook;
Object    *object;
struct lvRender *message;

The message arguments is a pointer to the following data structure:

struct lvResource {
  UWORD   lvr_Command;
  APTR    lvr_Entry;
};

lvr_Command -- This can be LVRC_MAKE which means that the  hook should
  create an entry or it can be  LVRC_KILL  which  means that the
  hook must dispose of a previously created entry.

lvr_Entry -- When this is a LVRC_MAKE command  this contains  the data
  added  to  the  listview  by  one  of  the  adding  methods or
  attributes.  When  this  is a LVRC_KILL command this points to
  whatever LVRC_MAKE has created.

The default creating/deletion that is done by the listview expects the
entries to be simple string pointers. Internally these  strings  are
copied to an internal buffer when the entry is created. When the entry
is disposed of the string copy is simply  de-allocated.   If  you  add
entries to the listview which are not string pointers  you must supply
your own resource handling using this attribute.

  EXAMPLE

/*
 *  This example takes a PubScreenNode as input,
 *  copies the name and adds that to the listview.
 *  Uppon deletion it simply de-allocates the copy
 *  of the string.
 */
```

```
__saveds __asm APTR
hookFunc( register __a0 struct Hook   *hook,
    register __a2 Object      *object,
    register __a1 struct lvResource *lvr )
{
    struct PubScreenNode  *psn =
  ( struct PubScreenNode * )lvr->lvr_Entry;
    UWORD      len;
    APTR       rc = NULL;

    /*
     *      Built or dispose?
     */
    switch ( lvr->lvr_Command ) {
  case  LVRC_MAKE:
      /*
       *      Determine string size.
       */
      len = strlen( psn->psn_Node.ln_name ) + 1;

      /*
       *      Allocate and copy the string.
       */
      if ( rc = ( APTR )AllocVec( len, MEMF_ANY ))
    strcpy(( UBYTE * )rc, psn->psn_Node.ln_Name );
      break;

  case  LVRC_KILL:
      /*
       *      Simply de-allocate whats created above.
       */
      FreeVec( lvr->lvr_Entry );
      break;
    }
    /*
     *      'rc' will be a pointer to the created
     *      string copy or NULL which indicates a
     *      memory error with LVRC_MAKE. If rc is non-NULL
     *      the string is added to the list of entries.
     */
    return( rc );
}
```

The hook must return a pointer to the data created when the command is LVRC_MAKE. When the command is LVRC_MAKE and NULL is returned the entry will not be added to the list.

LVRC_KILL commands do not have a return code defined.

  DEFAULT
NULL (internal memory handling).

  APPLICABILITY
(I).

  SEE ALSO

```
LISTV_DisplayHook, LISTV_CompareHook
```

1.43 listviewclass/LISTV_Select[xxx]

```
  NAME
LISTV_Select, LISTV_SelectMulti ** V39 ** -- ( ULONG )

  FUNCTION
To select an entry in the list. The entry you select will also be made
visible   in   the   display   area.  The data required is the logical
number of the entry in the list starting with 0 as the first entry.

The following magic numbers are allowed in the tag it's data field:

LISTV_Select_First    -- Select the first entry. ** V38 **

LISTV_Select_Last     -- Select the last entry. ** V38 **

LISTV_Select_Next     -- Select the next  entry.  If there is no entry
  selected yet the first visible entry is selected. ** V38 **

LISTV_Select_Previous -- Select  the  previous entry. If there is no
  selected  entry  yet  the  first  visible  entry  is selected.
  ** V38 **

LISTV_Select_Top      -- Select the first visible entry. ** V38 **

LISTV_Select_Page_Up  -- Select the entry one page above  the current.
  If the currently selected entry is not the  top-entry  the top
  entry will be selected. Otherwise the entry one-page up  minus
  one is selected. When no entry is selected the  first  visible
  entry is selected. ** V38 **

LISTV_Select_Page_Down -- Select the entry one page below the current.
  If the currently selected entry is not the   bottom-entry  the
  bottom entry will be selected.  Otherwise  the  entry one-page
  down minus one is selected.  When  no  entry  is  selected the
  first visible entry is selected. ** V38 **

LISTV_Select_All -- Selects all entries in the list.  Please note that
  this  magic  number  will  only  work  on  listviews in multi-
  selection   mode   and   it   will   only   work   with   the
  LISTV_SelectMulti and LISTV_SelectMultiNotVisible  attributes.
  ** V39 **

LISTV_SelectMulti  will  select  the  entry  without  deselecting  any
previous selected items while LISTV_Select will deselect  any previous
selections.

  APPLICABILITY
(SU).

  SEE ALSO
LISTV_SelectNotVisible, LISTV_SelectMultiNotVisible, LISTV_DeSelect
```

1.44 listviewclass/LISTV_Select[xxx]NotVisible

```
  NAME
LISTV_SelectNotVisible, LISTV_SelectMultiNotVisible -- ( ULONG )
** V39 **

  FUNCTION
To select an entry in the list.  This attribute works exactly like the
LISTV_Select and LISTV_SelectMulti attributes with the exception  that
the  selected  entry  is  not  moved into the current view area of the
list.

  APPLICABILITY
(SU).

  SEE ALSO
LIST_Select, LISTV_SelectMulti
```

1.45 listviewclass/LISTV_ShowDropPos

```
  NAME
LISTV_ShowDropPos -- ( BOOL ) ** V40 **

  FUNCTION
When  set  to  TRUE  in  combination with the baseclass  BT_DropObject
and/or BT_DragObject attributes you will create a  Listview  object in
which entries can be dropped at a specific location. Example:

list = ListviewObject,
  BT_DragObject,    TRUE,
  BT_DropObject,    TRUE,
  LISTV_ShowDropSpot, TRUE,
EndObject;

This  creates  a  listview  object  in  which the entries can be moved
arround by means of drag and drop. Please look at the supplied example
programs to see the possibilities of Listview drag and drop.

  DEFAULT
FALSE.

  APPLICABILITY
(IS).

  SEE ALSO
baseclass.doc/BT_DragObject, baseclass.doc/BT_DropObject,
LISTV_DropPos
```

1.46 listviewclass/LISTV_SortEntryArray

```
  NAME
LISTV_SortEntryArray -- ( BOOL )
```

```
  FUNCTION
To sort the entries added at object create time. By default the
entries added with the LISTV_EntryArray attribute will ocure in the
list in the same order as they ocure in the array. When this attribute
is set to TRUE these entries will be sorted.

  DEFAULT
FALSE.

  APPLICABILITY
(I).

  SEE ALSO
LISTV_EntryArray
```

1.47 listviewclass/LISTV_ThinFrames

```
  NAME
LISTV_ThinFrames -- ( BOOL )

  FUNCTION
To make all listview object framing appear as thin frames. This will
help you to make an aspect-ratio dependant GUI.

  DEFAULT
FALSE.

  APPLICABILITY
(I).
```

1.48 listviewclass/LISTV_Title

```
  NAME
LISTV_Title -- ( UBYTE * )  ** V41 **

  FUNCTION
Set a title for the list.

To specify titles with multiple columns, place a tab character between
the title strings, like this: "Name\tSize\tDate".

  DEFAULT
NULL.

  APPLICABILITY
(ISG).
```

1.49 listviewclass/LISTV_TitleHook

```
  NAME
LISTV_TitleHook -- ( struct Hook * )

  FUNCTION
To  add  a hook to render a title for the list. Multi-column listviews
normally  have  a  title  entry which is rendered in the list area but
does  not scroll with the list. To support multi-column listviews this
hook can be defined which will keep room for a single entry at the top
of the list area which is reserved for this purpose.  The hook routine
is  called  exactly the same as the LISTV_DisplayHook routine with the
exception  that  the  lvr_Entry  field  of the lvRender structure will
contain a NULL pointer.

  DEFAULT
NULL (no title).

  APPLICABILITY
(I).

  SEE ALSO
LISTV_DisplayHook
```

1.50 listviewclass/LISTV_Top

```
  NAME
LISTV_Top -- ( ULONG )

  FUNCTION
Set the top-entry of the visible part of the list. This tag is  mostly
used by the prop object that is connected to  the listview  but it can
also  be  controlled by your program. The data of this tag must be the
number of the node to set at the top of the visible area.

  DEFAULT
0.

  APPLICABILITY
(ISGU).
```

1.51 listviewclass/LISTV_ViewBounds

```
  NAME
LISTV_ViewBounds -- ( struct IBox * ) ** V40 **

  FUNCTION
To query the bounds of the view area of the  listview  object.  Please
note that reading this attribute is only  valid  after the  object has
been rendered.

You will be passes a pointer (READ-ONLY) to a struct IBox in which the
bounds of the view area are described.
```

```
  APPLICABILITY
(G).
```

1.52 listviewclass/PGA_NewLook

```
  NAME
PGA_NewLook -- ( BOOL )

  FUNCTION
To make the scroller of the listview gadget appear in the new look.

  DEFAULT
FALSE.

  APPLICABILITY
(I).
```