

**Screen**

**COLLABORATORS**

	<i>TITLE :</i> Screen		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		February 8, 2025	

**REVISION HISTORY**

NUMBER	DATE	DESCRIPTION	NAME

# Contents

<b>1</b>	<b>Screen</b>	<b>1</b>
1.1	Screen V1.00 . . . . .	1
1.2	findscreen . . . . .	1
1.3	findfrontscreen . . . . .	2
1.4	wbtoscreen . . . . .	2
1.5	screen . . . . .	2
1.6	screenmousex . . . . .	8
1.7	screenmousey . . . . .	8
1.8	screendepth . . . . .	9
1.9	screenwidth . . . . .	9
1.10	screenheight . . . . .	9
1.11	showscreen . . . . .	9
1.12	hidescreen . . . . .	9
1.13	usescreen . . . . .	10
1.14	viewport . . . . .	10
1.15	screenbarheight . . . . .	10
1.16	screenfontheight . . . . .	10
1.17	wbordertop . . . . .	10
1.18	nwborderbottom . . . . .	11
1.19	closescreen . . . . .	11
1.20	initscreen . . . . .	11
1.21	screenid . . . . .	11
1.22	screenrastport . . . . .	11
1.23	flashscreen . . . . .	12

---

# Chapter 1

## Screen

### 1.1 Screen V1.00

Pure Basic Screen library V1.00

Screens are well know on the Amiga as it's the base of the display. The AmigaOS can handle any numbers of screen at the same time and any screens can have its own properties (width, height, depth...) Same screens can be used for both games or applications.

Commands summary:

- CloseScreen
- FindScreen
- FindFrontScreen
- FlashScreen
- HideScreen
- InitScreen
- Screen
- ScreenBarHeight
- ScreenDepth
- ScreenFontHeight
- ScreenHeight
- ScreenID
- ScreenMouseX
- ScreenMouseY
- ScreenRastPort
- ScreenWidth
- ShowScreen
- UseScreen
- ViewPort
- WbToScreen

Example:

Open screens

### 1.2 findscreen

---

## SYNTAX

```
ScreenID.l = FindScreen(#Screen, ScreenName$)
```

## COMMAND

Find the default public screen and return its ScreenID pointer.  
If the ScreenID is NULL, no public screens can be found.

You can specify a ScreenName\$, so it will look in the public screen list to show if the screen is opened, and if so, will catch it !

#Screen = Number to indentifie the screen later.

ScreenName\$ = Name of the screen to find. If null name passed "", it will return the default public screen.

### 1.3 findfrontscreen

## SYNTAX

```
ScreenID.l = FindFrontScreen(#Screen)
```

## COMMAND

Find the front most screen and return its ScreenID. If the ScreenID is NULL, no screens can be found (!).

#Screen = Number to indentify the screen later.

### 1.4 wbtoscreen

## SYNTAX

```
ScreenID.l = WbToScreen(#Screen)
```

## FUNCTION

Try to find the Workbench screen and return its ScreenID. If the ScreenID is NULL, the Workbench screen is not found.

#Screen = Number to indentify the screen later.

### 1.5 screen

## SYNTAX

```
ScreenID.l = Screen(#Screen, Width, Height, Depth, TagList)
```

## FUNCTION

Open a new screen and return its ScreenID. If the ScreenID is NULL, the screen can't be opened. The newly-opened screen becomes the currently used screen (no need for UseScreen() function).

#Screen = Number to indentify the screen later.

NOTE: The AmigaLibs.res file must be entered in the compiler/option

---

resident field if you want to use the Tags.

Availables Tags:

```
#SA_Left
#SA_Top
#SA_Width
#SA_Height
```

The defaults for the #SA\_Left, #SA\_Top, #SA\_Width, and #SA\_Height tags end up being a bit complex. If none of these tags are specified, and no NewScreen structure is used, then the left/top/width/height correctly match the display clip of your screen (see #SA\_DClip and #SA\_Overscan).

The difficulty comes with overscanned screens, because the normal value of #SA\_Left or #SA\_Top for such a screen may be non-zero. If a NewScreen structure is supplied, then the left/top/width/height come originally from there. If no NewScreen structure is supplied, but a non-default #SA\_Width (#SA\_Height) is specified, then #SA\_Left (#SA\_Top) defaults to zero instead. In these cases, the left and top edge may not be what you want.

If you need to specify explicit width or height, or supply a NewScreen, you must supply correct values for #SA\_Left and #SA\_Top. The correct normal values are the display clip rectangle's MinX and MinY values respectively. If you are using the #SA\_DClip tag, then you already have a rectangle to consult for these values. If you are using #SA\_Overscan to get one of the standard overscan types, you may use QueryOverscan() to get a rectangle for that overscan type.

```
#SA_Depth (defaults to 1)
#SA_DetailPen (defaults to 0)
#SA_BlockPen (defaults to 1)
#SA_Title (defaults to NULL)
#SA_Font (defaults to NULL, meaning user's preferred monospace font)
#SA_BitMap (whose existence also implies CUSTOMBITMAP).
```

Several tags are Booleans, which means that depending on whether their corresponding ti\_Data field is zero (FALSE) or non-zero (TRUE), they specify Boolean attributes. The ones corresponding to Boolean flags in the NewScreen.Type field are:

```
#SA_ShowTitle (defaults to TRUE)
#SA_Behind (equiv. to SCREENBEHIND) (defaults to FALSE)
#SA_Quiet (equiv. to SCREENQUIET) (defaults to FALSE)
```

The following tags provide extended information to Intuition when creating a screen:

```
#SA_Type: ti_Data corresponds to the SCREENTYPE bits of the
NewScreen.Type field. This should be one of PUBLICSCREEN or
CUSTOMSCREEN. The other bits of the NewScreen.Type field
must be set with the appropriate tags (#SA_Behind, #SA_Quiet,
```

etc.)

#SA\_DisplayID: ti\_Data is a 32-bit extended display mode ID, as defined in the <graphics/modeid.h> include file (V39 and up) or in <graphics/displayinfo.h> (V37 and V38).

#SA\_Overscan: ti\_Data contains a defined constant specifying one of the system standard overscan dimensions appropriate for the display mode of the screen. Used with the Width and Height dimensions STDSCREENWIDTH and STDSCREEN, this makes it trivial to open an overscanned or standard dimension screen. You may also hand-pick your various dimensions for overscanned or other screens, by specifying screen position and dimensions explicitly, and by using #SA\_DClip to explicitly specify an overscanned DisplayClip region.

The values for ti\_Data of this tag are as follows:

OSCAN\_TEXT - Text Overscan region. A region which is completely on screen and readable ("text safe"). A preferences data setting, this is backward equivalent with the old MoreRows, and specifies the DisplayClip and default dimensions of the Workbench screen. This is the default.

OSCAN\_STANDARD - Also a preferences setting, this specifies a rectangle whose edges are "just out of view." This yields the most efficient position and dimensions of on-monitor presentations, such as games and artwork.

OSCAN\_MAX - This is the largest rectangular region that the graphics library can handle "comfortably" for a given mode. Screens can smoothly scroll (hardware pan) within this region, and any DisplayClip or Screen region within this rectangle is also legal. It is not a preferences item, but reflects the limits of the graphics hardware and software.

OSCAN\_VIDEO - This is the largest region that the graphics library can display, comfortable or not. There is no guarantee that all smaller rectangles are valid. This region is typically out of sight on any monitor or TV, but provides our best shot at "edge-to-edge" video generation.

Remember, using overscan drastically effects memory use and chip memory bandwidth. Always use the smallest (standard) overscan region that works for your application.

#SA\_DClip: ti\_Data is a pointer to a rectangle which explicitly defines a DisplayClip region for this screen. See QueryOverscan() for the role of the DisplayClip region.

Except for overscan display screens, this parameter is unnecessary, and specifying a standard value using #SA\_Overscan is normally an easier way to get overscan.

#SA\_AutoScroll: this is a Boolean tag item, which specifies that this screens is to scroll automatically when the mouse pointer reaches the edge of the screen. The operation of this requires

that the screen dimensions be larger than its DisplayClip region.

#SA\_PubName: If this field is present (and ti\_Data is non-NULL), it means that the screen is a public screen, and that the public screen name string is pointed to by ti\_Data. Public screens are opened in "PRIVATE" mode and must be made public using PubScreenStatus( screen, 0 ).

#SA\_Pens: The ti\_Data field (if non-NULL) points to a UWORD array of pen specification, as defined for struct DrawInfo. This array will be used to initialize the screen's DrawInfo.dri\_Pens array.

#SA\_Pens is also used to decide that a screen is ready to support the full-blown "new look" graphics. If you want the 3D embossed look, you must provide this tag, and the ti\_Data value cannot be NULL. If it points to a "minimal" array, containing just the terminator ~0, you can specify "new look" without providing any values for the pen array.

The way the DrawInfo pens are determined is Intuition picks a default pen-array. Then, any pens you supply with #SA\_Pens override the defaults, up until the ~0 in your array.

If the screen is monochrome or old-look, the default will be the standard two-color pens.

If the screen is two or more planes deep, the default will be the standard four-color pens, which now include the new-look menu colors.

If the screen has the #SA\_LikeWorkbench property, the default will be the user's preferred pen-array, changeable through preferences.

The following two tag items specify the task and signal to be issued to notify when the last "visitor" window closes on a public screen. This support is to assist envisioned public screen manager programs.

#SA\_PubTask: Task to be signalled. If absent (and #SA\_PubSig is valid), use the task which called OpenScreen() or OpenScreenTagList().

#SA\_PubSig: Data is a UBYTE signal number (not flag) used to notify a task when the last visitor window closes on a public screen.

#SA\_Colors: ti\_Data points to an array of ColorSpec structures (terminated with ColorIndex = -1) which specify initial values of the screen's color palette.

#SA\_FullPalette: this is a Boolean attribute. Prior to V36, there were just 7 RGB color values that Intuition maintained in its user preferences (playfield colors 0-3, and colors 17-19 for the sprite). When opening a screen, the color

map for the screens viewport is first initialized by graphics (graphics.library/GetColorMap()) then these seven values are overridden to take the preferences values.

In V36, Intuition maintains a full set of 32 preferences colors. If you specify TRUE for #SA\_FullPalette, Intuition will override ALL color map entries with its full suite of preferred colors. (Defaults to FALSE).

#SA\_ErrorCode: ti\_Data points to a ULONG in which Intuition will stick an extended error code if OpenScreen[TagList]() fails. Values are of this include 0, for success, and:

- OSERR\_NOMONITOR - monitor for display mode not available.
- OSERR\_NOCHIPS - you need newer custom chips for display mode.
- OSERR\_NOMEM - couldn't get normal memory
- OSERR\_NOCHIPMEM - couldn't get chip memory
- OSERR\_PUBNOTUNIQUE - public screen name already used
- OSERR\_UNKNOWNMODE - don't recognize display mode requested
- OSERR\_TOODEEP - screen too deep to be displayed on this hardware (V39)
- OSERR\_ATTACHFAIL - An illegal attachment of screens was requested (V39)

NOTE: These values are not the same as some similar return values defined in graphics.library/ModeNotAvailable().

#SA\_SysFont: ti\_Data selects one of the system standard fonts specified in preferences. This tag item overrides the NewScreen.Font field and the #SA\_Font tag item.

Values recognized in ti\_Data at present are:

- 0 - old DefaultFont, fixed-width, the default.
- 1 - Workbench screen preferred font. You have to be very font sensitive to handle a proportional or larger than traditional screen font.

NOTE WELL: if you select sysfont 1, windows opened on your screen will not inherit the screen font, but rather the window RastPort will be initialized to the old-style DefaultFont (sysfont 0).

Attached screen tags: V39 supports attached screens, where one or more child screens can be associated with a parent screen. Attached screens depth-arrange as a group, and always remain adjacent depth-wise. Independent depth-arrangement of child screens is possible through the V39 ScreenDepth() call. If a child screen is made non-draggable through {#SA\_Draggable, FALSE}, then it will drag exclusively with the parent. Normal child screens drag independently of the parent, but are pulled down when the parent is. Use the #SA\_Parent, #SA\_FrontChild, and #SA\_BackChild tags to attach screens.

#SA\_Parent: If you wish to attach this screen to an already-open parent screen, use this tag and set ti\_Data to point to the parent screen. See also #SA\_FrontChild and #SA\_BackChild. (V39).

---

- #SA\_FrontChild:** If you wish to attach an already-open child screen to this screen, set `ti_Data` to point to the child screen. The child screen will come to the front of the family defined by the parent screen you are opening. See also `#SA_Parent` and `#SA_BackChild`. (V39)
- #SA\_BackChild:** If you wish to attach an already-open child screen to this screen, set `ti_Data` to point to the child screen. The child screen will go to the back of the family defined by the parent screen you are opening. See also `#SA_Parent` and `#SA_FrontChild`. (V39)
- #SA\_BackFill:** `ti_Data` is a pointer to a backfill hook for the screen's `Layer_Info`.  
(see `layers.library/InstallLayerInfoHook()`). (V39).
- #SA\_Draggable:** `ti_Data` is a boolean. Set to `FALSE` if you wish your screen to be non-draggable. This tag should be used very sparingly!. Defaults to `TRUE`. For child screens (see `#SA_Parent`, `#SA_FrontChild`, and `#SA_BackChild`) this tag has a slightly different meaning: non-draggable child screens are non-draggable with respect to their parent, meaning they always drag exactly with the parent, as opposed to having relative freedom. Also see `ScreenPosition()`. (V39)
- #SA\_Exclusive:** `ti_Data` is a boolean. Set to `TRUE` if you never want your screen to share the display with another screen. This means that your screen can't be pulled down, and will not appear behind other screens that are pulled down. Your screen may still be depth arranged, though. Use this tag sparingly! Defaults to `FALSE`. Starting with V40, attached screens may be `#SA_Exclusive`. Setting `#SA_Exclusive` for each screen will produce an exclusive family. (V39).
- #SA\_SharePens:** For those pens in the screen's `DrawInfo->dri_Pens`, Intuition obtains them in shared mode (see `graphics.library/ObtainPen()`). For compatibility, Intuition obtains the other pens of a public screen as `PENF_EXCLUSIVE`. Screens that wish to manage the pens themselves should generally set this tag to `TRUE`. This instructs Intuition to leave the other pens unallocated. Defaults to `FALSE`. (V39).
- #SA\_Colors32:** Tag to set the screen's initial palette colors at 32 bits-per-gun. `ti_Data` is a pointer to a table to be passed to the `graphics.library/LoadRGB32()` function. This format supports both runs of color registers and sparse registers. See the autodoc for that function for full details. Any color set here has precedence over the same register set by `#SA_Colors`. (V39).
- #SA\_Interleaved:** `ti_Data` is a boolean. Set to `TRUE` to request an interleaved bitmap for your screen. Defaults to `FALSE`. If the system cannot allocate an interleaved bitmap for you, it will attempt to allocate a non-interleaved one (V39).
-

#SA\_VideoControl: `ti_Data` points to a taglist that will be passed to `VideoControl()` after your screen is open. You might use this to turn on border-sprites, for example. (V39).

#SA\_ColorMapEntries: `ti_Data` is the number of entries that you wish Intuition to allocate for this screen's ColorMap. While Intuition allocates a suitable number for ordinary use, certain graphics.library features require a ColorMap which is larger than default. (The default value is 1<<depth, but not less than 32). (V39)

#SA\_LikeWorkbench: `ti_Data` is boolean. Set to TRUE to get a screen just like the Workbench screen. This is the best way to inherit all the characteristics of the Workbench, including depth, colors, pen-array, screen mode, etc. Individual attributes can be overridden through the use of tags. (#SA\_LikeWorkbench itself overrides things specified in the NewScreen structure). Attention should be paid to hidden assumptions when doing this. For example, setting the depth to two makes assumptions about the pen values in the DrawInfo pens. Note that this tag requests that Intuition ATTEMPT to open the screen to match the Workbench. There are fallbacks in case that fails, so it is not correct to make enquiries about the Workbench screen then make strong assumptions about what you're going to get. (Defaults to FALSE). (V39)

#SA\_MinimizeISG: `ti_Data` is boolean. For compatibility, Intuition always ensures that the inter-screen gap is at least three non-interlaced lines. If your application would look best with the smallest possible inter-screen gap, set `ti_Data` to TRUE. If you use the new graphics `VideoControl()` `VC_NoColorPaletteLoad` tag for your screen's ViewPort, you should also set this tag. (V40)

## 1.6 screenmousex

SYNTAX

```
x.w = ScreenMouseX()
```

FUNCTION

Returns the mouse position, in pixels, relative to the left of the current screen.

## 1.7 screenmousey

SYNTAX

```
y.w = ScreenMouseY()
```

FUNCTION

---

Returns the mouse position, in pixels, relative to the top of the current screen.

## 1.8 screendepth

SYNTAX

```
Result.w = ScreenDepth()
```

FUNCTION

Returns the depth of the current screen.

## 1.9 screenwidth

SYNTAX

```
width.w = ScreenWidth()
```

FUNCTION

Returns the width, in pixels, of the current screen.

## 1.10 screenheight

SYNTAX

```
height.w = ScreenHeight()
```

FUNCTION

Returns the height, in pixels, of the current screen.

## 1.11 showscreen

SYNTAX

```
ShowScreen()
```

STATEMENT

Bring the current screen to the front of the display.

## 1.12 hidescreen

SYNTAX

```
HideScreen()
```

STATEMENT

put the current screen to the back of the display.

---

## 1.13 usescreen

### SYNTAX

```
UseScreen(#Screen)
```

### STATEMENT

Change the current screen to the given #Screen.

## 1.14 viewport

### SYNTAX

```
Result.l = ViewPort()
```

### FUNCTION

Returns the viewport of the current screen. This function is written to enable you to get the screen's viewport and should be used only by advanced programmers who want access to all the OS functions.

## 1.15 screenbarheight

### SYNTAX

```
Result.b = ScreenBarHeight()
```

### FUNCTION

Return the used screen menu bar height. Useful to adjust windows just under it (for example).

## 1.16 screenfontheight

### SYNTAX

```
Result.b = ScreenFontHeight()
```

### FUNCTION

Returns the font height of the current screen.

## 1.17 wborder top

### SYNTAX

```
Result.b = WBorderTop()
```

### FUNCTION

Returns the window border-top which will be opened on this screen. The result includes the Font height, ie the border window with a title.

---

## 1.18 nwborderbottom

### SYNTAX

```
Result.b = NWBorderBottom
```

### FUNCTION

Returns the window border-bottom which will be opened on this screen.

## 1.19 closescreen

### SYNTAX

```
CloseScreen(#Screen)
```

### STATEMENT

Close the given screen.

## 1.20 initscreen

### SYNTAX

```
result.l = InitScreen(#NumScreenMax)
```

### FUNCTION

Init all the Screen environments for later use. You must put this function at the top of your source code if you want to use the NScreen commands. You can test the result to see if the Window environment is correctly initialized.

#NumScreenMax : Maximum number of screens to handle.

## 1.21 screenid

### SYNTAX

```
ScreenID.l = ScreenID()
```

### FUNCTION

Returns the Intuition Screen pointer. Very useful.

## 1.22 screenrastport

### SYNTAX

```
ScreenRP.l = ScreenRastPort()
```

### FUNCTION

Returns the current Screen RastPort. It allows you to use the 2D Drawing fonctions directly on the screen's bitmap:

Example

---

```
DrawingOutput(ScreenRastPort()) ; Set the drawing function output
                                ; on the current screen

BoxFill(10, 10, 100, 100)      ; This box will be drawn on the screen
```

## 1.23 flashscreen

### SYNTAX

```
FlashScreen()
```

### FUNCTION

```
Flash the current screen.
```

---