

Screen

COLLABORATORS

	<i>TITLE :</i> Screen		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		January 23, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	Screen	1
1.1	Screen	1
1.2	closescreen	2
1.3	createdualplayfield	2
1.4	findscreen	2
1.5	findfrontscreen	3
1.6	flashscreen	3
1.7	hidescreen	3
1.8	initscreen	3
1.9	openscreen	3
1.10	removedualplayfield	9
1.11	screenbarheight	10
1.12	screendepth	10
1.13	screenfontheight	10
1.14	screenheight	10
1.15	screenid	10
1.16	screenmousex	11
1.17	screenmousey	11
1.18	screenrastport	11
1.19	screenwidth	11
1.20	showscreen	12
1.21	usescreen	12
1.22	viewport	12
1.23	wbtoscreen	12

Chapter 1

Screen

1.1 Screen

Pure Basic Screen Library

Bildschirme (Screens) sind auf dem Amiga wohlbekannt, da sie die Grundlage der Bildschirmanzeige bilden. Das AmigaOS kann eine beliebige Anzahl an Bildschirmen zur gleichen Zeit verwalten und jeder Bildschirm kann seine eigenen Eigenschaften (Breite, Höhe, Tiefe...) haben. Dieselben Bildschirme können für beides, Spiele und Applikationen, benutzt werden.

Befehlsübersicht:

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

Beispiel:

Open screens

1.2 closescreen

SYNTAX

CloseScreen(#Screen)

STATEMENT

Schließt den angegebenen Bildschirm.

1.3 createdualplayfield

SYNTAX

CreateDualPlayField(BitMapID)

COMMAND

Erstellt ein sekundäres "Spielfeld" (Playfield) auf dem aktuellen Bildschirm mit der angegebenen 'BitMapID'. Das 'DualPlayField' ist eine Möglichkeit, den Bildschirm in zwei Zonen aufzuteilen, eine vordere und eine hintere Zone. Jede Zone ist total unabhängig von der anderen und hat jeweils ihre eigene BitMap. Sie können die BitMap des Hintergrunds mit dem Befehl 'ShowBackBitMap()' verändern. Die Farbe 0 ist die transparente Farbe. Die Bildschirmtiefe kann nicht größer als 3 (8 Farben) auf dem OCS/ECS Chip bzw. nicht größer als 4 (16 Farben) auf dem AGA Chipsatz sein. Jede Zone kann ihre eigene Palette haben, somit können Sie bis zu 32 Farben auf einem 16 Farben-Bildschirm verwenden. (Hübsch, nicht ? :-).

Sie müssen den Befehl 'RemoveDualPlayField()' aufrufen, bevor Sie den Bildschirm schließen oder das Programm beenden. Dies wird NICHT automatisch erledigt !

1.4 findscreen

SYNTAX

ScreenID.1 = FindScreen(#Screen, ScreenName\$)

COMMAND

Ermittelt den Standard Public-Screen (öffentlicher Bildschirm) und gibt dessen ScreenID Zeiger zurück. Ist die ScreenID gleich NULL, wurden keine Public-Screens gefunden.

Sie können einen ScreenName\$ angeben, die Funktion sieht dann in der Public-Screen Liste nach, ob dieser Bildschirm geöffnet wurde, und wenn ja wird dieser "gefangen"!

#Screen = Nummer, um den Bildschirm später zu identifizieren zu können.
ScreenName\$ = Name des zu findenden Bildschirms. Wurde als Name "" übergeben, wird der Standard (Default) Public-Screen zurückgegeben.

1.5 findfrontscreen

SYNTAX

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

COMMAND

Ermittelt den vordersten Bildschirm und gibt dessen ScreenID zurück.
Ist die ScreenID gleich NULL, wurden keine Bildschirme gefunden (!).

#Screen = Nummer, um den Bildschirm später zu identifizieren zu können.

1.6 flashscreen

SYNTAX

```
FlashScreen()
```

FUNCTION

Läßt den aktuellen Bildschirm aufblitzen.

1.7 hidescreen

SYNTAX

```
HideScreen()
```

STATEMENT

Legt den aktuellen Bildschirm hinter alle anderen geöffneten
Bildschirme.

1.8 initscreen

SYNTAX

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

FUNCTION

Initialisiert die Programmierumgebung für Screens zur weiteren Nutzung.
Sie müssen diese Funktion am Anfang Ihres Sourcecodes aufrufen, wenn
Sie weitere Befehle aus der Screen.library nutzen möchten. Sie können
das Ergebnis testen, um die korrekte Initialisierung der Bildschirm-
Umgebung zu testen.

#NumScreenMax : Maximale Anzahl zu verwaltender Bildschirme.

1.9 openscreen

SYNTAX

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

FUNCTION

Öffnet einen neuen Bildschirm und gibt dessen ScreenID zurück. Ist die ScreenID gleich NULL, konnte der Bildschirm nicht geöffnet werden. Der neu geöffnete Bildschirm wird automatisch der aktuell benutzte Bildschirm (es besteht also kein Bedarf für die UseScreen() Funktion).

#Screen = Nummer, um den Bildschirm später zu identifizieren zu können.

Hinweis: Die AmigaLibs.res Datei muß im Resident-Feld von Compiler/Optionen eingetragen sein, wenn Sie die Tags nutzen möchten.

Verfügbare 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.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 $1 < \text{depth}$, 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)

1.10 removedualplayfield

SYNTAX

`RemoveDualPlayField()`

COMMAND

Entfernt das 'DualPlayField' Feature vom aktuellen Bildschirm.

Sie müssen den Befehl 'RemoveDualPlayField()' aufrufen, bevor Sie den Bildschirm schließen oder das Programm beenden. Dies wird NICHT automatisch erledigt !

1.11 screenbarheight

SYNTAX

Result.b = ScreenBarHeight()

FUNCTION

Gibt die Höhe der Menü-/Titelzeile des aktuellen Bildschirms zurück. Nützlich um z.B. Fenster darunter zu positionieren.

1.12 screendepth

SYNTAX

Result.w = ScreenDepth()

FUNCTION

Gibt die Tiefe des aktuellen Bildschirms zurück.

1.13 screenfontheight

SYNTAX

Result.b = ScreenFontHeight()

FUNCTION

Ermittelt die Höhe des Zeichensatzes vom aktuellen Bildschirm.

1.14 screenheight

SYNTAX

height.w = ScreenHeight()

FUNCTION

Gibt die Höhe des aktuellen Bildschirms in Pixeln zurück.

1.15 screenid

SYNTAX

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

FUNCTION

Gibt den Intuition Screen Zeiger zurück. Sehr nützlich.

1.16 screenmousex

SYNTAX

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

FUNCTION

Gibt die X-Position der Maus in Pixeln, ausgehend vom linken Rand des aktuellen Bildschirms, zurück.

1.17 screenmousey

SYNTAX

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

FUNCTION

Gibt die Y-Position der Maus in Pixeln, ausgehend vom linken Rand des aktuellen Bildschirms, zurück.

1.18 screenrastport

SYNTAX

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

FUNCTION

Gibt den aktuellen Screen RastPort zurück. Dies erlaubt es Ihnen, 2D Zeichenfunktionen direkt auf der Bildschirm Bitmap auszuführen:

Beispiel:

```
DrawingOutput(ScreenRastPort()) ; Setzt das Ausgabeziel für Zeichen-  
                                ; Operationen auf den aktuellen  
                                ; Bildschirm
```

```
BoxFill(10, 10, 100, 100)      ; Dieser Kasten wird auf den Bildschirm  
                                ; gezeichnet.
```

1.19 screenwidth

SYNTAX

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

FUNCTION

Gibt die Breite des aktuellen Bildschirms in Pixeln zurück.

1.20 showscreen

SYNTAX

```
ShowScreen()
```

STATEMENT

Bringt den aktuellen Bildschirm nach vorne auf das Display.

1.21 usescreen

SYNTAX

```
UseScreen(#Screen)
```

STATEMENT

Macht den angegebenen #Screen zum aktuellen Bildschirm.

1.22 viewport

SYNTAX

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

FUNCTION

Gibt den Viewport des aktuellen Bildschirms zurück. Diese Funktion wurde geschrieben, um Ihnen Zugriff auf den ViewPort des Bildschirms zu geben und sollte nur von erfahrenen Programmierern genutzt werden, die Zugriff auf alle OS Funktionen haben möchten.

1.23 wbtoscreen

SYNTAX

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

FUNCTION

Versucht, den Workbench Bildschirm zu finden und gibt dessen ScreenID zurück. Ergibt die ScreenID gleich NULL, wurde der Workbench Bildschirm nicht gefunden.

#Screen = Nummer, um den Bildschirm später zu identifizieren zu können.
