ScriptX Post 1.5 Differences

Three of the ScriptX reference works have been given minor revisions, reflecting both editorial reworking of the documentation and changes to the application programming interface. The *ScriptX Language Guide* is untouched, except for editorial changes. This document is a complete summary of changes to the ScriptX API since the most recent printing of the ScriptX documentation.

Global Functions Suspend the Event System

Two global functions, `eventCriticalUp` and `eventCriticalDown`, were conceived as a means to prevent ScriptX from thrashing when too many events are passing through the queue. A low-end system can be flooded with events in a scene that a high-end system would handle easily. Use `eventCriticalUp` and `eventCriticalDown`, to block receipt of new mouse and keyboard events until the system finishes handling events that are already in the queue.

eventCriticalDown (global function)

`eventCriticalDown()` ⇨ *(none)*

Allows the event system to resume processing user input. This function is always paired with a prior call to `eventCriticalUp`, which suspends processing of keyboard and mouse events. A call to `eventCriticalDown` balances a call to `eventCriticalUp`, enabling the system to resume processing of these events. See `eventCriticalUp`, a global function defined below. *(Events component)*

eventCriticalUp (global function)

`eventCriticalUp()` ⇨ *(none)*

Suspends processing of keyboard and mouse events until the next call to `eventCriticalDown`. A call to `eventCriticalUp` precedes a critical segment of a code in which the system discards mouse and keyboard events. Of course, `eventCriticalUp` affects any class that maintains an interest in mouse and keyboard events, including `Menu`, `ScrollBar`, and `TextEdit`.

Within an event-critical segment, input devices continue to receive events from the underlying operating system, but they "swallow" these events rather than convert them into ScriptX events. An event-critical segment must be followed by a call to `eventCriticalDown`, which allows the event system to resume processing of these events. These two functions are designed so that a program can insure that the system finishes responding to a keyboard or mouse event before the next event is received.

The paired functions `eventCriticalUp` and `eventCriticalDown` are analogous to `threadCriticalUp` and `threadCriticalDown`. The system maintains a count of calls to `eventCriticalUp`, and each call must be matched by a corresponding call to `eventCriticalDown` before the system resumes processing user input. If the system is likely to be in an event-critical state for a long time, set the value of `pointerType`, an instance variable defined by `MouseDevice`, to `@wait`.

Use `eventCriticalUp` and `eventCriticalDown` cautiously and sparingly. An unbalanced call to `eventCriticalUp` can leave the system in a suspended state, from which the user may be unable to regain control of the system. The keyboard combinations command-period (MacOS) and control-break (Windows and OS/2) allow the user to escape from an event-critical state. *(Events component)*

Note – This function affects the entire runtime or authoring environment, including any other title or tool that is currently open. If a program requires user input, it must insure that event processing is restored.

Pointer Class Gives Developers Mouse Pointer Control

The new `Pointer` class allows ScriptX to control the mouse pointer, which can be any 16 x 16 x 1 `Bitmap` object. As with previous versions of the ScriptX Player, the mouse pointer is determined by the value of `pointerType`, an instance variable defined by `MouseDevice`. In previous versions, the value of `pointerType` was a name token that identified one of several standard system pointers. In the current version, the value of `pointerType` can be either a `NameClass` object that represents one of the standard system pointers or a `Pointer` object that represents a custom pointer. See the `Pointer` and `MouseDevice` classes for more information.

Global Functions Replace OpenPanel and SavePanel Classes

Three new global functions have been added, with two of them providing the functionality of the `OpenPanel` and `SavePanel` classes, and the third one adding the ability to generate a native system dialog box containing a user-supplied message.

The three classes `FilePanel`, `OpenPanel`, and `SavePanel` have been removed from ScriptX, which means that any code using the `OpenPanel` or `SavePanel` classes will need to be rewritten using the new, simpler global functions `presentOpenFilePanel` and `presentSaveFilePanel`.

presentOpenFilePanel (global function)

`presentOpenFilePanel` *typelist* ⇨ `Array` or `undefined`

typelist	`Array` object containing `NameClass` objects, which may include the following file types: `@title` - A ScriptX file containing a `TitleContainer` object `@library` - A ScriptX file containing a `LibraryContainer` object `@accessory` - A ScriptX file containing an `AccessoryContainer` object `@binary` - A file containing binary data `@text` - A file containing ASCII data `@unknown` - The file type is not specified, and any file type may be selected

Presents a platform-specific **Open** dialog box where the user can select a file; returns an array of strings representing the full path of the file selected, or `undefined` if no file is selected. The dialog box displays as options the file types specified in *typelist*. If *@unknown* or any unknown value is specified in *typelist*, the **Open** dialog box allows any file type to be selected. If the user clicks **Open** or **OK** (depending on the platform), the filename is returned; if the user clicks **Cancel**, the function returns `undefined`. In any case, it is up to the title to actually open the file.

presentSaveFilePanel (global function)

`presentSaveFilePanel` *prompt defaultName* ⇨ `Array` or `undefined`

prompt	String object
defaultName	String object

Presents a platform-specific **Save As** file dialog box, where the user can type in a filename, and which contains the specified prompt and default filename. If the user clicks **OK** or **Save**, the function returns an array containing the full pathname of the file named in the file entry field. If the user clicks **Cancel**, the function returns `undefined`. In any case, it is up to the title to actually save the file. The string of text supplied for *prompt* will appear next to the file entry field as a prompt; the string of text supplied as the default file name will appear in the dialog box's file entry field when it is first opened.

presentMessagePanel (global function)

`presentMessagePanel` *message icon button-list default-ix cancel-ix*
⇨ `ImmediateInteger`

message	`String` object, the text to display
icon	`NameClass` object, one of `@warning`, `@critical`, `@information`, or `@none`
button-list	`Array` object containing 1, 2, or 3 String objects to be used as button names
default-ix	`ImmediateInteger` object, the index of the button that is the default
cancel-ix	`ImmediateInteger` object, the index of the button that is to be returned by the cancel key or 0

Presents a native system dialog box containing the user-supplied message specified in *message*, the icon specified in *icon*, and buttons labeled with the strings specified in *button-list*; returns the index into *button-list* of the button that was pressed. One button must be the default (the button selected if the user hits the return key), and it is specified by giving its index into *button-list*. There must also be a button designated as the one to be returned if the user hits command -. (the command key plus a period) on a Macintosh or Ctrl-Break on a Windows machine. This button can be the same as the default key, but that is not necessarily the case.

The icons displayed in the message window are machine-dependent.

Example 1

```
presentMessagePanel "Nyuk, nyuk" @critical #("Moe", "Larry",
    "Curly") 2 1
```

Example 1 presents a dialog box with the message "Nyuk, nyuk." The dialog box contains an icon indicating that the message is "critical", and it has three buttons labeled "Moe", "Larry", and "Curly". "Larry" is the default button, and "Moe" is the cancel button.

Example 2

```
presentMessagePanel "Format your disk" @warning #("Format",
    "No") 2 2
```

Example 2 presents a dialog box with the icon that indicates a "warning" message and the message "Format your disk". It has two buttons, one labeled "Format" and one labelled "No". "No" is both the default button and the cancel button.

Three Classes Handle the System Menu

ScriptX gives title developers control of the system menu bar, the standard set of menus that is associated with the active application. The `SystemMenuBar` class has been modified extensively—it is now an array that maintains a list of `SystemMenu` objects. An instance of `SystemMenu` is an array of `SystemMenuItem` objects. Note the distinction between `Menu` objects, which are presenters, and are handled by the ScriptX compositor, and `SystemMenu` objects, which are handled by the underlying operating system. See the `SystemMenuBar`, `SystemMenu`, and `SystemMenuItem` classes for more information.

New Text Features

You can now double-click and shift-click for selecting text:

Double-clicking—double-click on any part of a word to select the whole word.

Shift-clicking—place the cursor at the beginning of the text you want to select, depress the shift key, and click at the end of the text you want to select. The selection will include everything between the initial cursor position and the position at which the mouse was clicked.

Miscellaneous changes

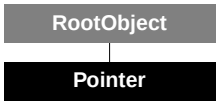
1. `clippingStencil` is now `clippingPresenter`. Even though the name has changed, this `ScrollingPresenter` instance variable has the same functionality and is still a `Stencil` object. Any code which uses `clippingStencil` will need to be changed so that `clippingPresenter` is used instead.

2. The global function `quit` now takes one argument, which can be any object. It ignores the argument. The `quit` function's signature was modified to make it compatible with the `selectAction`, an instance variable defined by the new `SystemMenuItem` class.

```
quit(true) -- the argument is ignored
```

3. The `systemQuery` global function has several new features. See the "Global Functions" chapter of the *ScriptX Class Reference*.

Pointer



Class type: Core class (concrete)
Resides in: ScriptX and KMP executables
Inherits from: `RootObject`
Component: Input Devices

`Pointer` is a concrete class that represents a mouse pointer on a ScriptX display surface. `Pointer` supplies a mouse device with a bitmap image and mask that visually represents the position of the mouse on the screen.

Creating and Initializing a New Instance

The following script creates a new instance of `Pointer` and sets the value of `pointerType` for an associated mouse device:

```
global bm := new bitmapSurface \
       bBox:(new Rect x2:16 y2:16) \
       bitsPerPixel:1
fill bm bm.bBox bm.bBox identityMatrix blackBrush
global myPointer := new Pointer \
       bitmap:bm \
       mask:bm \
       hotSpot:(new Point x:5 y:5)
global myMouseDevice := new MouseDevice
myMouseDevice.pointerType := myPointer
```

The global variable `myPointer` contains an initialized instance of `Pointer`. The `new` method uses keywords that are defined in `init`.

init

`init` *self* [`bitmap:`*bitmap*] [`mask:`*bitmap*] [`hotSpot:`*point*] ⇨ *(none)*

self	`InputDevice` object
`bitmap:`	`Bitmap` object
`mask:`	`Bitmap` object
`hotSpot:`	`Point` object

Initializes the `Pointer` *self*, applying the values supplied with the keywords to the instance variables of the same name.

If you omit an optional keyword, its default value is used:

```
bitmap:undefined
mask:undefined
hotSpot:(new Point x:1 y:1)
```

Instance Variables

bitMap

self`.bitmap` (read-write) `Bitmap`

Specifies a bitmap that represents the position of the pointer *self* on a display surface. This bitmap must be 16 x 16 in size, and 1 bit deep.

hotSpot

self.hotSpot (read-write) Point

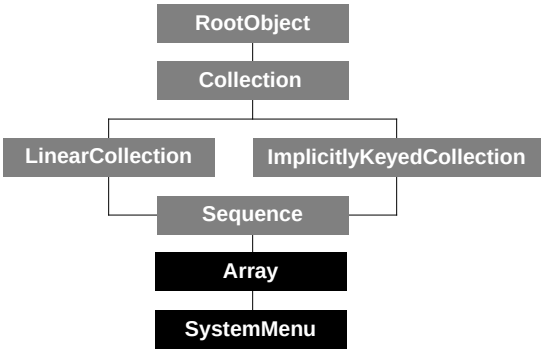
Specifies the hot spot or active spot for the pointer *self*. When the mouse is clicked, the mouse click occurs at this point in the two-dimensional coordinates of the display surface.

mask

self.mask (read-only) Bitmap

Specifies a bitmap that represents a mask for the pointer *self* on a display surface. This bitmap must be 16 x 16 in size, and 1 bit deep.

SystemMenu



Class type: Core Class (concrete)
Resides in: ScriptX and KMP executables
Inherits from: `Array`
Component: Title Management

The `SystemMenu` class provides a container for `SystemMenuItem` objects. A `SystemMenuBar` object acts as a container for `SystemMenu` objects. The standard menus that appear on all platforms, such as the **File** and **Edit** menus, are implemented as `SystemMenu` objects by ScriptX.

`SystemMenu` inherits from `Array`. The class specializes those methods that add, set, or delete the value of any element in the array, such as `addNth`, `setNth`, and `deleteNth`, to insure that elements are instances of `SystemMenuItem`, and that the menu bar is updated when any changes are made.

Creating and Initializing a New Instance

The following script creates a new instance of `SystemMenu`:

```
myMenuChoices := new SystemMenu \
       name:"Edit"
```

The variable `myMenuChoices` contains the initialized system menu. When `myMenuChoices` is instantiated, ScriptX applies the default values of `growable` and `initialSize`, keywords defined by the parent `Array` class.

init

`init` *self* [`name:` *string*] ⇨ *self*

self	`SystemMenu` object
`name:`	`String` object, used to set the name of the menu

Superclasses of `SystemMenu` apply the following keywords:

`initialSize:`	`Integer` object
`growable:`	`Boolean` object

Initializes the `SystemMenu` object *self*, setting the name that appears at the top of the menu to the value supplied for `name`. See the `Array` class for information on how `initialSize` and `growable` are applied.

Instance Variables

authorData

self.`authorData` (read-write) *(object)*

Specifies an object that is supplied as an argument when the system menu's preselect action or select action is called.

name

self.`name` (read-write) `String`

Specifies the name of the system menu *self*. The value of name is stored internally as a string, such as `"Edit"` or `"File"`. Any object that can be coerced to a string can be used to set the value of name.

preSelectAction

self.`preSelectAction` (read-write) *(function)*

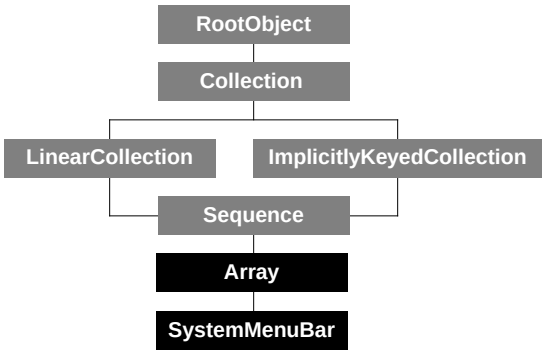
Specifies a function that is called automatically when the user clicks on the menu in the menu bar, but before a menu item is selected. This function is called with the one argument, for which the value of `authorData` is supplied.

selectAction

self.`selectAction` (read-write) *(function)*

Specifies a function that is called automatically when a menu item is selected. This function is called with the one argument, for which the value of `authorData` is supplied.

SystemMenuBar



Class type: Core class (concrete)
Resides in: ScriptX and KMP executables
Inherits from: `Array`
Component: Title Management

The `SystemMenuBar` class gives ScriptX the ability to control the appearance of the menu bar that is presented by the native operating system. Each title has its own system menu bar, as specified by the `systemMenuBar` instance variable defined by `TitleContainer`. You can hide or show the menu bar and enable or disable its menu items. One active title can show the menu bar, while another hides it. The menu bar automatically changes depending on which title has user focus.

The location and appearance of the system menu bar is platform-dependent, as shown in the following figure. With MacOS, the menu bar is always located at the top of the screen. In Windows and OS/2, the menu bar is located below the ScriptX title bar, which appears at the top of the screen only when the ScriptX window is "maximized."

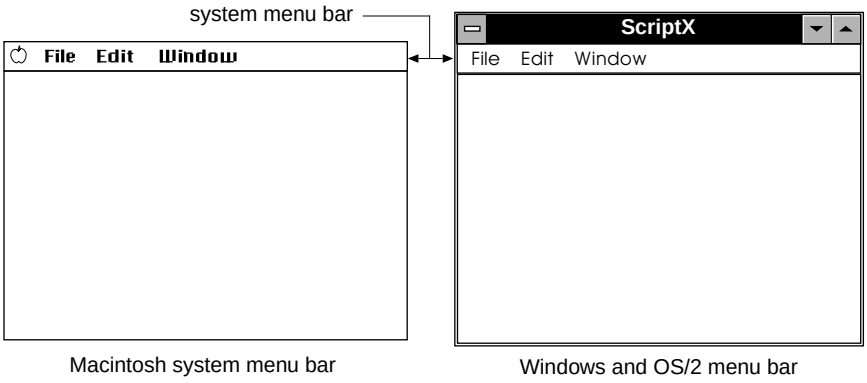


Figure 0-1: The menu bar's appearance is determined by the underlying platform.

A system menu bar is a container for system menus, and a system menu is a container for system menu items. `SystemMenuBar` inherits from `Array`, which it specializes to store only instances of `SystemMenu`. The class specializes those methods that add, set, or delete the value of any element in the array, such as `addNth`, `setNth`, and `deleteNth`, to insure that elements are instances of `SystemMenu`, and that the menu bar is updated when any changes are made.

The `SystemMenuBar` class provides access, via class variables, to internal functions that open and close titles, open accessories, bring up a new listener window, and bring up a page setup dialog box. This allows a programmer to invoke these functions as the select action of a menu item. Each of these functions takes one argument, which is ignored, for signature compatibility.

Creating and Initializing a New Instance

The following script creates a new instance of `SystemMenuBar`:

```
myMenus := new SystemMenuBar
```

The variable `myMenus` contains the initialized system menu bar. When `myMenus` is instantiated, ScriptX applies the default values of `growable` and `initialSize`, keywords defined by the parent `Array` class.

init

`init` *self* ⇨ *self*

self `SystemMenuBar` object

Superclasses of `SystemMenuBar` apply the following keywords:

`initialSize:`	`Integer` object
`growable:`	`Boolean` object

Initializes the `SystemMenuBar` object *self*. See the `Array` class for information on how `initialSize` and `growable` are applied.

Class Variables

closeTitleAction (SystemMenuBar)

self`.closeTitleAction` (read-only) *(function)*

Returns the internal function that closes a title. The associated function takes one argument, which is ignored.

newListenerAction (SystemMenuBar)

self`.newListenerAction` (read-only) *(function)*

Returns the internal function that opens a new Listener window. The associated function takes one argument, which is ignored.

openAccessoryAction (SystemMenuBar)

self`.openAccessoryAction` (read-only) *(function)*

Returns the internal function that opens an accessory. The associated function takes one argument, which is ignored.

openTitleAction (SystemMenuBar)

self`.openTitleAction` (read-only) *(function)*

Returns the internal function that opens a title. The associated function takes one argument, which is ignored.

pageSetupAction (SystemMenuBar)

self`.pageSetupAction` (read-only) *(function)*

Returns the internal function that opens a page setup dialog. The associated function takes one argument, which is ignored.

Instance Variables

hasUserFocus

self.`hasUserFocus` (read-only) `Boolean`

When `true`, the appearance of the system menu bar *self* responds to changes in its instance variables and methods; when `false`, it does not respond to changes. In the current release, since each title has exactly one system menu bar, this variable is set to `true` whenever a window in the title has user focus.

A title container manages focus on its system menu bar automatically—it stores a reference to its system menu bar in its `systemMenuBar` instance variable. It is possible for no window in a title to have focus while its system menu bar has focus. This can happen if two titles share the same menu bar—the background title does not have focus, but its menu bar has focus because it is used by the foreground title.

isVisible

self.`isVisible` (read-write) `Boolean`

When `true`, the system menu bar *self* is visible; when `false`, the system menu bar is hidden. Setting `isVisible` has the same effect as calling the methods `show` and `hide`.

Instance Methods

hide

`hide` *self* ⇨ *self*

Hides the system menu bar *self*. Has the same effect as setting `isVisible` to `false`.

setUserFocus

`setUserFocus` *self hasUserFocus* ⇨ `Boolean`

self	`SystemMenuBar` object
hasUserFocus	`Boolean` object

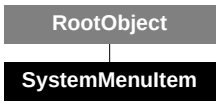
Sets user focus for the system menu bar *self*. This method is not usually called from the scripter, since the title container normally manages focus. It is visible to the scripter so that it can be specialized.

show

`show` *self* ⇨ *self*

Shows the system menu bar *self*. Has the same effect as setting `isVisible` to `true`.

SystemMenuItem



Class type: Core Class (concrete)
Resides in: ScriptX and KMP executables
Inherits from: `RootObject`
Component: Title Management

The `SystemMenuItem` class implements the behavior of individual menu items on a system menu

Creating and Initializing a New Instance

The following script creates a new instance of `SystemMenuItem`:

```
myItem := new SystemMenuItem \
       name: "Print List"
```

The variable `myItem` contains the initialized system menu item, which can be added to a system menu. It is displayed in a menu as a choice or menu command with the name "Print List." The `new` method uses keyword arguments defined by the `init` method.

init

`init` *self* ⇨ *self*

self	`SystemMenuItem` object
`name:`	`String` object

Initializes the `SystemMenuItem` object *self*, applying the keyword `name` to the instance variable of the same name.

If you omit an optional keyword, its default value is used:

`name:undefined`

Instance Variables

authorData

self`.authorData` (read-write) *(object)*

Specifies an object that is supplied as an argument when the system menu item's select action is called.

checked

self`.checked` (read-write) `Boolean`

Specifies whether the menu item *self* is checked.

enabled

self`.enabled` (read-write) `Boolean`

Specifies whether the menu item *self* is enabled.

name

self.name (read-write) String

Specifies the name of the system menu item *self*. The value of name is stored internally as a string, such as "Copy" or "Paste". Any object that can be coerced to a string can be used to set the value of name.

selectAction

self.selectAction (read-write) *(function)*

Specifies a function that is called automatically when a menu item is selected. This function is called with the one argument, for which the value of authorData is supplied.