Autodesk®
Scaleform®

Scaleform 4.2 AS3 Extensions Reference

This document describes ActionScript 3.0 extensions available in Scaleform 4.2.

Authors: Artem Bolgar, Prasad Silva
Version: 1.04
Last Edited: September 17, 2012

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.2

How to Contact Autodesk Scaleform:

Document	AS3 Extensions Reference
Address	Autodesk Scaleform Corporation 6305 Ivy Lane, Suite 310 Greenbelt, MD 20770, USA
Website	www.scaleform.com
Email	info@scaleform.com
Direct	(301) 446-3200
Fax	(301) 446-3199

Table of Contents

1 Introduction

The Autodesk Scaleform® SDK™ is a light-weight, high-performance rich media Adobe® Flash® vector graphics engine, built on a clean-room implementation specifically for console and PC game developers. Autodesk Scaleform combines the scalability and development ease of proven visual authoring tools, such as the Adobe Creative Suite®, with the latest hardware graphics acceleration that cutting-edge game developers demand.

Since Flash is primarily designed for the web and not game or application development, it has limited functionality in some areas such as focus handling, mouse support, text field support and handling of IME input. Autodesk Scaleform improves the basic functionality in these areas by adding the "extensions" to ActionScript classes. To enable extension support in Scaleform Player it is necessary to set scaleform.gfx.Extensions.enabled property to true.

```
scaleform.gfx.Extensions.enabled = true;
```

or

```
import scaleform.gfx.*;
Extensions.enabled = true;
```

It is also necessary to add a path *GFXSDK/Resources/AS3/CLIK* as "Source path" in Flash to make extension classes visible for the ActionScript 3.0 compiler (in Flash Studio*: 'Edit' -> 'Preferences' -> 'ActionScript' -> 'ActionScript 3.0 Settings…' -> 'Source path'*). In most cases, developers will want to add this statement to the first frame in the FLA file that will be using extensions. If this property is not set, Scaleform Player will ignore all references to extensions, attempting to achieve the maximum level of Flash compatibility.

Developers should understand that extension functionality described in this document will not work in the standard Flash Player, as it is only implemented in Scaleform. Associated with each extension is a Scaleform version number, identifying the release number of the player SDK in which the extension was added. Extensions will not work in the Scaleform Player with earlier version numbers.

Although Autodesk will make the best effort to keep extension APIs supported and consistent throughout different releases, we reserve the right to change, rename or remove extension APIs in the future Scaleform releases. If significant changes are made, we will try to associate them with major point releases and provide notice as early as possible.

2 Extensions

enabled static property

```
enabled:Boolean  [read-write]
```

Scaleform version: 4.0.12

Enables/disables the extensions.

noInvisibleAdvance static property

```
noInvisibleAdvance:Boolean  [read-write]
```

Scaleform version: 4.0.12

If set to true, this property turns off advancing of all invisible movie clips. This might be used to improve performance of SWFs that contain many hidden movie clips. Note, Flash advances invisible movie clips (it still executes timeline animation, invokes frame's ActionScript code and so on). Thus, setting this property to true may lead to differences in behavior between Scaleform and Flash.

Note: This property may stop working for some children in the case when the child's branch is re-parented from another parent with this flag not set, and other way around; it may continue to work for the child's branch when it is re-parented to the parent with the flag not set. In this case user should set/reset the flag manually for the parent after re-parenting children (and it will be propagated to all its children).

getTopMostEntity() static method

```
public function getTopMostEntity(x : Number, y : Number, testAll: Boolean) :
DisplayObject
```

Scaleform version: 4.0.13

Returns a topmost DisplayObject instance that can be found at (x, y) coordinates (stage coordinate space). This method also may differ between characters with button handlers set (e.g., CLICK, MOUSE_DOWN, ROLL_OVER, etc) and without them. This distinction may be useful to filter out characters that have no handlers to handle mouse events.

This method is inverse to MovieClip.hitTest since the hitTest checks if x and y coordinates are inside of the particular object and the getTopMostEntity returns the actual object at the specified x and y coordinates. Thus Extensions.getTopMostEntity(x, y).hitTest(x, y, true) == true.

Parameters

testAll : Boolean – Indicates to look only for characters with button handlers (false) or for any character (true). If this parameter is not specified then getTopMostEntity assumes it is true.
x : Number, y : Number – Stage coordinates at which to look for a character.

See Also:

getMouseTopMostEntity

getMouseTopMostEntity() static method

```
public function getTopMostEntity([testAll : Boolean],
   [mouseIndex : uint]) : DisplayObject
```

Scaleform version: 4.0.13

Returns a topmost DisplayObject instance that can be found at the current mouse cursor position. This function is similar to getTopMostEntity, but instead of explicit coordinates it uses mouse coordinates.

Parameters

testAll : Boolean – Indicates to look only for characters with button handlers (false) or for any character (true). If this parameter is not specified then getTopMostEntity assumes it is true.
mouseIndex : Number – Zero-based mouse index.

See Also:

getTopMostEntity

getMouseCursorType() static method

```
public function getMouseCursorType([mouseIndex : uint]) : String
```

Scaleform version: 4.0.13

This static method returns the current mouse cursor type for the specified mouse controller. This method is similar to flash.ui.Mouse.cursor property, the only difference is this method allows to change cursor for multiple mice.

To track and optionally prevent mouse cursor change use MouseCursorEvent.CURSOR_CHANGE extension event.

Parameters
mouseIndex : Number – Zero-based mouse index (zero by default).

Returns
Returns type of the cursor, see flash.ui.MouseCursor static constants, such as:

- `flash.ui.MouseCursor.ARROW : String = "arrow"`
- `flash.ui.MouseCursor.BUTTON : String = "button"`
- `flash.ui.MouseCursor.HAND : String = "hand"`
- `flash.ui.MouseCursor.IBEAM : String = "ibeam"`

See Also:
`setMouseCursorType`
`MouseCursorEvent`

setMouseCursorType() static method

```
public function setMouseCursorType(cursor : String, [mouseIndex : uint]) : void
```

Scaleform version: 4.0.13

This static method changes the mouse cursor according to the parameter cursor. This method is similar to flash.ui.Mouse.cursor property, the only difference is this method allows to change cursor for multiple mice.

To track and optionally prevent mouse cursor change use MouseCursorEvent.CURSOR_CHANGE extension event.

Parameters
cursor : uint – Type of the cursor, see flash.ui.MouseCursor static constants, such as:

- flash.ui.MouseCursor.ARROW : String = "arrow"
- flash.ui.MouseCursor.BUTTON : String = "button"
- flash.ui.MouseCursor.HAND : String = "hand"
- flash.ui.MouseCursor.IBEAM : String = "ibeam"

mouseIndex : Number – Zero-based mouse index (zero by default).

See Also:
`getMouseCursorType`

MouseCursorEvent

numControllers static property

```
numControllers:uint  [read]
```

Scaleform version: 4.0.12

Returns the number of controllers detected in the system.

setEdgeAAMode() static method

```
public function setEdgeAAMode(dispObj:DisplayObject, mode:uint):void
```

Scaleform version: 4.0.12

Sets the EdgeAA mode of a specific display object and its children. The following mode values are accepted:

- scaleform.gfx.Extensions.EDGEAA_INHERIT = 0;Inherit EdgeAA mode from parent. On by default.
- scaleform.gfx.Extensions.EDGEAA_ON = 1;Enable EdgeAA mode for the target display object and its children, unless disabled (see EDGEAA_DISABLE)
- scaleform.gfx.Extensions.EDGEAA_OFF = 2;Do not use EdgeAA for the target display object and its children.
- scaleform.gfx.Extensions.EDGEAA_DISABLE = 3;Disable EdgeAA for the target display object and its children, overriding an inherited EDGEAA_ON mode value.

Parameters
dispObj : DisplayObject – The target display object to apply the new EdgeAA mode value.
mode : uint – The EdgeAA mode value.

See Also:
getEdgeAAMode

getEdgeAAMode() static method

```
static public function getEdgeAAMode(dispObj:DisplayObject): uint
```

Scaleform version: 4.0.12

Retrieves the EdgeAA mode value from a specific display object.

Parameters

dispObj : DisplayObject – The target display object to retrieve the EdgeAA mode value.

See Also:

`setEdgeAAMode`.

3 DisplayObjectEx Extensions

setRendererString static method

```
static public function setRendererString(o:DisplayObject, s:String)
```

Scaleform version: 4.0.17

This method allows custom directives to be sent to the renderer from ActionScript for any MovieClip instance. If the method is set, the string value will be sent to the renderer as user data.

There is no default value.
Example:

```
import scaleform.gfx.*;

// m is a MovieClip on stage.

DisplayObjectEx.setRendererString(m, "Abc");
trace(DisplayObjectEx.getRendererString(m));
```

setRendererFloat static method

```
static public function setRendererFloat(o:DisplayObject, f:Number)
```

Scaleform version: 4.0.17

This method allows custom directives to be sent to the renderer from ActionScript for any MovieClip instance. If the method is set, the float value will be sent to the renderer as user data.

There is no default value.

Example:

```
import scaleform.gfx.*;

// m is a MovieClip on stage.

DisplayObjectEx.setRendererFloat(m, 17.1717);
trace(DisplayObjectEx.getRendererFloat(m));
```

disableBatching static method

```
static public function disableBatching(o:DisplayObject, b:Boolean)
```

Scaleform version: 4.0.17

This method disables batching of mesh generation for custom drawing.

Example

```
import scaleform.gfx.*;

// m is a MovieClip on stage.

DisplayObjectEx.disableBatching(batchDisable, true);
trace(DisplayObjectEx.isBatchingDisabled(batchDisable));
```

4 FocusEventEx Extensions

```
package scaleform.gfx
{
    import flash.events.FocusEvent;

    public final class FocusEventEx extends FocusEvent
    {
        public var controllerIdx : uint = 0;

        public function FocusEventEx(type:String) { super(type); }
    }
}
```

This event is an extension of the standard flash.events.FocusEvent. It adds a ‘controllerIdx’ member that indicates a zero-based index of the controller that caused the event. When Extensions.enabled property is set to true Scaleform always generates FocusEventEx events instead of standard FocusEvent. A user can check if the received event is an instance of the FocusEventEx and if so cast the event object to the extension type. Example:

```
import scaleform.gfx.*;
import flash.events.FocusEvent;

Extensions.enabled = true;

function ev(e: FocusEvent)
{
      if (e is FocusEventEx)
      {
            var ee: FocusEventEx = e as FocusEventEx;
            trace("    controllerIdx = "+ee.controllerIdx);
      }
}
stage.addEventListener(FocusEvent.MOUSE_FOCUS_CHANGE, ev);
```

controllerIdx property

```
controllerIdx : uint  [read]
```

Scaleform version: 4.0.12

Indicates which keyboard/controller is used for the event (zero-based index).

5 FocusManager Extensions

alwaysEnableArrowKeys static property

```
alwaysEnableArrowKeys:Boolean  [read-write]
```

Scaleform version: 4.0.12

This static property allows arrow keys to change focus even when the `_focusrect` property is set to `false` (applied when the focus is captured). By default, Flash does not allow you to use arrow keys to change focus if the yellow focus rectangle is disabled via `_focusrect = false`. To change this behavior, set the `alwaysEnableArrowKeys` property to `true`.

disableFocusKeys static property

```
disableFocusKeys:Boolean  [read-write]
```

Scaleform version: 4.0.12

This static property disables handling of all focus keys (TAB, Shift-TAB and arrow keys), thus, users may implement their own focus keys management.

moveFocus() static method

```
static public function moveFocus(keyToSimulate : String,
                          startFromMovie:InteractiveObject = null,
                          includeFocusEnabledChars : Boolean = false,
                          controllerIdx : uint = 0) : InteractiveObject
```

Scaleform version: 4.0.12

This static method is used to move a focus rectangle by simulating key pressing of one of focus keys: TAB, Shift-TAB or arrow keys. This method with cooperation of `disableFocusKeys` and `modalClip` properties may be used for implementing custom focus management.

Parameters

`keyToSimulate : String -` Name of key to simulate: “up”, “down”, “left”, “right”, “tab”, “shifttab”.
`startFromMovie:InteractiveObject –` Optional parameter that specifies a character; moveFocus will use it instead of the currently focused one as a start point. This property might be null or undefined, which means that currently focused character is used as a starting point; this might be useful to specify the third optional parameter.
`includeFocusEnabledChars : Boolean –` Optional flag that allows moveFocus onto characters with only the `focusEnabled` property set as well as onto characters with the `tabEnabled / tabIndex` properties set. If the flag is not specified or set to `false` then only characters with the `tabEnabled / tabIndex` properties set will participate in focus movement.
`controllerIdx : uint -` Index of the controller used for the operation. If not specified, then the default controller (controller 0) is used.

Returns
Returns next character to be focused or `null` if the character cannot be found.

See also:
`findFocus`
`disableFocusKeys`
`setModalClip`
`getModalClip`

findFocus() static method

```
static public function findFocus(keyToSimulate : String,
                          parentMovie:DisplayObjectContainer = null,
                          loop : Boolean = false,
                          startFromMovie:InteractiveObject = null,
                          includeFocusEnabledChars : Boolean = false,
                          controllerIdx : uint = 0) : InteractiveObject
```

Scaleform version: 4.0.12

This static method is used to find the next focus item by simulating key pressing of one of the following keys: TAB, Shift-TAB or arrow keys. This method with conjunction with the `disableFocusKeys` and `setModalClip/getModalClip` extensions may be used to implement custom focus management.

Parameters
`keyToSimulate : String -` Name of key to simulate: “up”, “down”, “left”, “right”, “tab”, “shifttab”.
`parentMovie:DisplayObjectContainer –` The movie clip that is used as a modal clip. The focus item search is performed only within this clip’s children. May be null.

`loop : Boolean -` Boolean flag to loop focus. For example, if the currently focused item is at the bottom and the key is “down”, then findFocus either returns “null” (if this flag is “false”) or the topmost focusable item (if the flag is “true”).
`startFromMovie:InteractiveObject –` Optional parameter that specifies a character that findFocus will use instead of the currently focused one as a start point. This property might be null or undefined, which means that the currently focused character is used as a starting point.
`includeFocusEnabledChars : Boolean –` Optional flag that allows moveFocus onto characters with only the `focusEnabled` property set as well as onto characters with the `tabEnabled / tabIndex` properties set. If the flag is not specified or set to `false` then only characters with the `tabEnabled / tabIndex` properties set will participate in focus movement.
`controllerIdx : uint -` A zero base index of the controller that is manipulating focus. This in conjunction with focus groups can be used to provide multi controller focus support.

Returns

Returns next character to be focused or `null` if the character cannot be found.

See also:

- `moveFocus`
- `disableFocusKeys`
- `setModalClip`
- `getModalClip`

setFocus() static method

```
static public function setFocus(obj:InteractiveObject, controllerIdx:uint = 0) : void
```

Scaleform version: 4.0.12

This static method does the same as assigning stage.focus property. The only difference is that it is possible to specify the index of the controller that should be associated with the operation.

Parameters

`obj:InteractiveObject -` Newly focused interactive object
`controllerIdx : uint -` Indicates which keyboard/controller is used for the event (zero-based index).

getFocus() static method

```
static public function getFocus(controllerIdx:uint = 0) : InteractiveObject
```

Scaleform version: 4.0.12

This static method returns the same value as the stage.focus property. The only difference is that it is possible to specify the index of the controller that should be associated with the operation.

Parameters

`controllerIdx : uint` - Indicates which keyboard/controller is used for the event (zero-based index).

Returns

Currently focused interactive object.

numFocusGroups static property

```
numFocusGroups() : uint  [read]
```

Scaleform version: 4.0.12

Returns the number of focus groups, set up by call to `setControllerFocusGroup`function. If focus groups 0 and 3 are active, numFocusGroups will return 2.

setFocusGroupMask() static method

```
public function setFocusGroupMask(obj:InteractiveObject, mask:uint) : void
```

Scaleform version: 4.0.12

This method sets a bitmask to a character and **ALL** of its children. This bitmask assigns focus group ownership to the character, meaning only the controllers denoted in the bitmask are able to move focus into and within the character. Focus groups can be associated with controllers by using `setControllerFocusGroup` extension method.

For example, let's assume that "button1" is to be focusable only by controller 0 and "movieclip2" by controllers 0 and 1. To achieve this behavior, associate focus groups with the controllers:

```
FocusManager.setControllerFocusGroup(0, 0);
FocusManager.setControllerFocusGroup(1, 1);

FocusManager.setFocusGroupMask(button1, 0x1); // bit 0 – focus group 0
FocusManager.setFocusGroupMask(movieclip2,0x1 | 0x2); // bits 0 and 1 – focus
                                                      //  groups 0 and 1
```

The "`focusGroupMask`" bitmask may be set to the parent movieclip. This will propogate the mask value to all of its children.

Parameters

`obj:InteractiveObject` - An interactive object
`mask:uint` - A focus group bitmask

See also:

`setControllerFocusGroup`
`getFocusGroupMask`

getFocusGroupMask() static method

```
static public function getFocusGroupMask(obj:InteractiveObject) : uint
```

Scaleform version: 4.0.12

Returns current focus group bitmask value (see `setFocusGroupMask` for details).

Parameters

`obj:InteractiveObject` - An interactive object

Returns

A focus group bitmask for the specified interactive object.

See also:

`setControllerFocusGroup`
`setFocusGroupMask`

setControllerFocusGroup() static method

```
static public function setControllerFocusGroup(controllerIdx:uint,
                                                focusGroupIdx:uint) :   Boolean
```

Scaleform version: 4.0.12

This static method associates the controller denoted by `controllerIndex` with a focus group. By default, all controllers are associated with focus group 0, which means that they are using the same focus. However, it is possible to make each controller work with their own focus. For example, if two controllers should have separate focus (in a split-screen use case) then `setControllerFocusGroup(1,1)` will create a separate focus group for the controller 1. Calling `setControllerFocusGroup(1,0)` will make controller 0 and 1 to share the same focus again.

Parameters
`controllerIdx:uint` - Zero-base index of the controller.
`focusGroupIdx:uint` - Zero-base index of the focus group.

Returns
Returns true if successful.

getControllerFocusGroup() static method

```
static public function getControllerFocusGroup(controllerIdx:uint) : uint
```

Scaleform version: 4.0.12

This static method returns the focus group index associated with the specified controller.

Parameters
`controllerIndex` – Zero-base index of the physical controller.

Returns
Zero-based index of focus group.

setModalClip() static method

```
static public function setModalClip(mc:Sprite, controllerIdx:uint = 0) : void
```

Scaleform version: 4.0.12

This static method sets the specified movie clip as a “modal” clip for focus management. This means TAB, Shift-TAB and arrow keys will move focus only inside the specified movie clip across all “tabable” children.

Parameters
`controllerIdx:uint` - A zero base index of the controller.
`mc:Sprite` - A modal clip.

getModalClip() static method

```
static public function getModalClip(controllerIdx:uint = 0) : Sprite
```

Scaleform version: 4.0.12

This static method returns the modal clip for the specified controller.

Parameters

`controllerIdx` - Zero base index of the controller.

Returns

A modal clip or undefined if not found.

getControllerMaskByFocusGroup () static method

```
public function getControllerMaskByFocusGroup(focusGroupIdx:uint) : uint
```

Scaleform version: 4.0.17

This static method returns a bitmask where each bit represents a controller that is associated with the specified focus group. Returns the state set by the `setControllerFocusGroup` function.

Parameters

`focusGroupIdx` - An index of focus group.

Returns

A bitmask of controllers.

6 GamePad Extensions

Control Constants

This class provides helper constants for generic game pad controls, such as triggers, analog sticks and buttons. They are a mirror reflection of the constants defined in SF_KeyCodes.h. Scaleform injects the correct values at runtime, therefore compiling SWFs that utilizes these constants will work as intended.

The Scaleform FxPlayer framework maps game pad controls to keyboard equivalents for convenience, however a custom integration or application may use these constants in conjunction with GFx::Movie::HandleEvent to provide a distinction between controllers and keyboards if necessary for AS3 key events.

The Scaleform FxPlayer framework does use these constants with GamePadAnalogEvents to provide appropriate feedback for such analog values. However, it is also conceivable that instead of using the GamePad constants, developers could also use keyboard codes instead. The choice of combining game pad events with keyboard events is left to the developers' discretion.

supportsAnalogEvents() static method

```
public static function supportsAnalogEvents() : Boolean
```

Scaleform version: 4.0.13

Returns true if game pad analog events (such as for triggers and thumb sticks) are supported by the Scaleform implementation for the underlying hardware platform. This value can be used to determine whether GamePadAnalogEvents are supported.

See Also

GamePadAnalogEvent

7 GamePadAnalogEvent Extensions

```
package scaleform.gfx
{
    import flash.events.Event;


    public final class GamePadAnalogEvent extends Event
    {
        public static const CHANGE:String   = "gamePadAnalogChange";

        public var code : uint          = 0;    // See scaleform.gfx.GamePad for
                                                // valid pad codes
        public var controllerIdx : uint = 0;
        public var xvalue : Number      = 0;    // Normalized [-1, 1]
        public var yvalue : Number      = 0;    // Normalized [-1, 1]

        public function GamePadAnalogEvent(bubbles:Boolean, cancelable:Boolean,
                                           code:uint, controllerIdx:uint = 0,
                                           xvalue:Number = 0, yvalue:Number = 0)
        {
            super(GamePadAnalogEvent.CHANGE, bubbles, cancelable);
            this.code = code;
            this.controllerIdx = controllerIdx;
            this.xvalue = xvalue;
            this.yvalue = yvalue;
        }
    }
}
```

This event is fired if Scaleform supports game pad analog events for a specific platform. It is only dispatched from the Stage and all listeners are expected to attach to the Stage object. The Scaleform FxPlayer framework fires these events with appropriate GamePad constants for the ‘code’ property, however developers may use other values (such as keyboard values) if necessary in their own integration of Scaleform. Example:

```
import scaleform.gfx.*;

trace(“GamePadAnalogEvents supported? “ + GamePad.supportsAnalogEvents());

function ev(e: GamePadAnalogEvent)
{
      trace("code = " + ev.code);
      trace("controllerIdx = " + ev.controllerIdx);
```

```
      trace("xvalue = " + ev.xvalue);
      trace("yvalue = " + ev.yvalue);
}
stage.addEventListener(GamePadAnalogEvent.CHANGE, ev);
```

code property

```
code : uint
```

Scaleform version: 4.0.13

Indicates which key/control was used to generate this event. The Scaleform FxPlayer framework will use game pad constants. See GamePad.

controllerIdx property

```
controllerIdx : uint
```

Scaleform version: 4.0.13

Indicates which keyboard/controller is used for the event (zero-based index).

xvalue property

```
xvalue : Number
```

Scaleform version: 4.0.13

Indicates the current value in the x-axis. The value will be normalized between -1 and 1, inclusive.

yvalue property

```
yvalue : Number
```

Scaleform version: 4.0.13

Indicates the current value in the y-axis. The value will be normalized between -1 and 1, inclusive.

8 InteractiveObjectEx Extensions

getHitTestDisable() static method

```
public function getHitTestDisable(o:InteractiveObject) : Boolean
```

Scaleform version: 4.0.13

Returns state of 'hitTestDisable' flag. When it is set to `true`, the `MovieClip.hitTest` function will ignore this interactive object during hit test detection. In addition, all other mouse events are not propagated to the object.

The default value is `false`.

Parameters

```
o      - An interactive object.
```

Returns

A Boolean value representing state of 'hitTestDisable' flag.

See also:

`InteractiveObjectEx.setHitTestDisable`

setHitTestDisable() static method

```
public function setHitTestDisable(o:InteractiveObject, f:Boolean) : void
```

Scaleform version: 4.0.13

Sets state of 'hitTestDisable' flag. When it is set to `true`, the `MovieClip.hitTest` function will ignore this interactive object during hit test detection. In addition, all other mouse events are not propagated to the object. The default value is `false`.

Parameters

```
o      - An interactive object.
```

f - A boolean value representing new state of 'hitTestDisable' flag.

See also

`InteractiveObjectEx.getHitTestDisable`

getTopmostLevel() static method

```
public function getTopmostLevel (o:InteractiveObject) : Boolean
```

Scaleform version: 4.0.13

Returns state of ‘topmostLevel’ flag. When it is set to `true`, this character is displayed on the top of all other ones regardless of its depth.

Parameters

`o - An interactive object.`

Returns

A boolean value representing state of ‘topmostLevel’ flag.

See also:

`InteractiveObjectEx.setTopmostLevel`

setTopmostLevel () static method

```
public function setTopmostLevel(o:InteractiveObject, f:Boolean) : void
```

Scaleform version: 4.0.13

Sets state of ‘topmostLevel’ flag. If it is set to `true` then this character is displayed on the top of all other ones regardless of its depth. This might be useful for implementing custom mouse cursors when the cursor should be drawn above objects from all levels. The default value is `false`.

In case of marking several characters as "topmostLevel", the draw order is the same as it would be without marking the characters topmost, i.e. if objectA was drawn underneath the objectB, then after making them topmost the objectA will still be under objectB, regardless of the order of setting "topmostLevel" property to `true`.
Note: Once a character is marked as "topmostLevel", the swapDepth ActionScript function will not have any effect on this character.

The default value is `false`.

Parameters

`o - An interactive object.`
`f` - A boolean value representing new state of ‘topmostLevel’ flag.

See also:

`InteractiveObjectEx.getTopmostLevel`

9 KeyboardEventEx Extensions

```
package scaleform.gfx
{
    import flash.events.KeyboardEvent;

    public final class KeyboardEventEx extends KeyboardEvent
    {
        public var controllerIdx : uint = 0;

        public function KeyboardEventEx(type:String) { super(type); }
    }
}
```

This event is an extension of the standard flash.events.KeyboardEvent. It adds a ‘controllerIdx’ member that indicates a zero-based index of the controller that caused the event. When Extensions.enabled property is set to true Scaleform always generates KeyboardEventEx events instead of standard KeyboardEvent. A user can check if the received event is an instance of the KeyboardEventEx and if so cast the event object to the extension type. Example:

```
import scaleform.gfx.*;
import flash.events.KeyboardEvent;

Extensions.enabled = true;

function ev(e: KeyboardEvent)
{
      if (e is KeyboardEventEx)
      {
            var ee: KeyboardEventEx = e as KeyboardEventEx;
            trace("    controllerIdx = "+ee.controllerIdx);
      }
}
stage.addEventListener(KeyboardEvent.KEY_DOWN, ev);
stage.addEventListener(KeyboardEvent.KEY_UP, ev);
```

controllerIdx property

```
controllerIdx : uint  [read]
```

Scaleform version: 4.0.12

Indicates which keyboard/controller is used for the event (zero-based index).

10 MouseCursorEvent Extensions

```
package scaleform.gfx
{
    import flash.events.Event;

    public final class MouseCursorEvent extends Event
    {
        public var cursor : String = "auto";
        public var mouseIdx : uint = 0;

        static public const CURSOR_CHANGE : String = "mouseCursorChange";

        public function MouseCursorEvent()
  {
      super("MouseCursorEvent", false, true);
  }
    }
}
```

This event serves to track and/or prevent mouse cursor change. It has the following Event's properties set:
`bubbles` – false, it does not bubble
`cancellable` – true, the default action (cursor change) may be prevented by calling `preventDefault()` method.

This event is useful to implement custom animated mouse cursor. Whenever Scaleform changes the shape of the mouse cursor (either by rolling over textfield or button, or by setting `flash.ui.Mouse.cursor` property, this event is fired **for a stage** (if Extensions.enabled is set to true). Example:

```
Extensions.enabled = true;

function e(e:MouseCursorEvent)
{
   trace(e.type + "  " + e.mouseIdx + "  "+ e.cursor);
   e.preventDefault();
}
stage.addEventListener(MouseCursorEvent.CURSOR_CHANGE, e);
```

Note: This event is fired ONLY when the listener is set on a stage.

cursor property

```
cursor : String  [read]
```

Scaleform version: 4.0.13

The property indicates the type of the cursor to be changed to. It contains one of the string values from flash.ui.MouseCursor class, such as:

- `flash.ui.MouseCursor.ARROW : String = “arrow”`
- `flash.ui.MouseCursor.BUTTON : String = “button”`
- `flash.ui.MouseCursor.HAND : String = “hand”`
- `flash.ui.MouseCursor.IBEAM : String = “ibeam”`

mouseIdx property

```
mouseIdx : uint  [read]
```

Scaleform version: 4.0.13

Indicates for which mouse/controller the event is generated (zero-based index).

11 MouseEventEx Extensions

```
package scaleform.gfx
{
    import flash.events.MouseEvent;

    public final class MouseEventEx extends MouseEvent
    {
        public var mouseIdx : uint = 0;
        public var nestingIdx : uint = 0;
        public var buttonIdx : uint = 0; // LEFT_BUTTON, RIGHT_BUTTON, ...

        public static const LEFT_BUTTON : uint  = 0;
        public static const RIGHT_BUTTON : uint = 1;
        public static const MIDDLE_BUTTON : uint = 2;

        public function MouseEventEx(type:String) { super(type); }
    }
}
```

This event is an extension of the standard flash.events.MouseEvent. It adds properties for multi-controllers and right/middle mouse buttons support. When Extensions.enabled property is set to true Scaleform always generates MouseEventEx events instead of standard MouseEvent. A user can check if the received event is an instance of the MouseEventEx and if so cast the event object to the extension type. Example:

```
import scaleform.gfx.*;
Extensions.enabled = true;

stage.doubleClickEnabled = true;

function ev(e:MouseEvent):void
{
      trace("!!!! EVENT. " + cnt++);
      trace("    eventType     = "+e.type);
      trace("    bubbles       = "+e.bubbles);
      trace("    eventPhase    = "+e.eventPhase);
      trace("    target        = "+e.target.name);
      trace("    currentTarget = "+e.currentTarget.name);
      if (e is MouseEventEx)
      {
            var ee:MouseEventEx = e as MouseEventEx;
            trace("    mouseIdx      = "+ee.mouseIdx);
            trace("    nestingIdx    = "+ee. nestingIdx);
```

```
            trace("    buttonIdx     = "+ee.buttonIdx);
      }
      trace(e);
}
stage.addEventListener(MouseEvent.MOUSE_DOWN, ev);
stage.addEventListener(MouseEvent.MOUSE_UP, ev);
stage.addEventListener(MouseEvent.CLICK, ev);
stage.addEventListener(MouseEvent.DOUBLE_CLICK, ev);
```

Note: Once extensions are enabled, the mouse event will be fired for right, middle, center, etc mouse button events and user must check the buttonIdx property to figure out which button generated the event.

buttonIdx property

```
buttonIdx : uint  [read]
```

Scaleform version: 4.0.13

Indicates for which button the event is generated (zero-based index). The value can be one of the following:

- MouseEventEx.LEFT_BUTTON : uint = 0 - left mouse button
- MouseEventEx.RIGHT_BUTTON : uint = 1 - right mouse button
- MouseEventEx.MIDDLE_BUTTON : uint = 2 - middle mouse button
- Any value greater than 2 is also legal in the case if mouse has more than 3 buttons.

mouseIdx property

```
mouseIdx : uint  [read]
```

Scaleform version: 4.0.13

Indicates for which mouse/controller the event is generated (zero-based index).

nestingIdx property

```
nestingIdx : uint  [read]
```

Scaleform version: 4.0.13

This property is optional for rollOver/Out, mouseOver/Out and dragOver/Out events. This parameter specifies the index of nested rollover/dragover event over the same character.
When nested rollOver/rollOut, mouseOver/Out and dragOver/dragOut events are generated (separately for each mouse cursor) then this parameter represents the zero-based index of nesting: the initial event will have 0 as the parameter; if the second cursor rolls over the same character, then the second rollOver/rollOut event will be fired with the member set to 1. If any of the cursors leaves the character then the rollOut/mouseOut/dragOut will be fired with the member set to 1; the last rollOut/mouseOut/dragOut event will be fired with the member set to 0.

12 TextEventEx Extensions

```
package scaleform.gfx
{
    import flash.events.TextEvent;

    public final class TextEventEx extends TextEvent
    {
        public var controllerIdx : uint = 0;

        public function TextEventEx(type:String) { super(type); }
    }
}
```

This event is an extension of the standard flash.events.TextEvent. It adds a ‘controllerIdx’ member that indicates a zero-based index of the controller that caused the event. When Extensions.enabled property is set to true Scaleform always generates TextEventEx events instead of standard TextEvent. A user can check if the received event is an instance of the TextEventEx and if so cast the event object to the extension type. Example:

```
import scaleform.gfx.*;
import flash.events.TextEvent;

Extensions.enabled = true;

function ev(e: TextEvent)
{
      if (e is TextEventEx)
      {
            var ee: TextEventEx = e as TextEventEx;
            trace("    controllerIdx = "+ee.controllerIdx);
      }
}
txf.addEventListener(TextEvent.TEXT_INPUT, ev);
```

controllerIdx property

```
controllerIdx : uint  [read]
```

Scaleform version: 4.0.12

Indicates which keyboard/controller is used for the event (zero-based index).

LINK_MOUSE_OVER/LINK_MOUSE_OUT events

```
public static const LINK_MOUSE_OVER:String = "linkMouseOver";
public static const LINK_MOUSE_OUT:String = "linkMouse";
```

Scaleform version: 4.0.14

Developers are now able to listen for mouse over/out events on TextField links (specified by HTML tags) using the TextFieldEx.LINK_MOUSE_OVER and TextFieldEx.LINK_MOUSE_OUT event extensions. These events behave as other AS3 events and can be listened to using the addEventListener paradigm.

13 System Extensions

actionVerbose static property

```
actionVerbose:Boolean  [read-write]
```

Scaleform version: 4.0.12

Enable/Disable opcode tracing.

getStackTrace() static method

```
public function getStackTrace() : String
```

Scaleform version: 4.0.12

Get current stack trace formatted as a string.

getCodeFileName() static method

```
public function getCodeFileName() : String
```

Scaleform version: 4.0.12

Get file name of currently executed code.

14 TextFieldEx Extensions

appendHtml() static method

```
public function appendHtml(textField:TextField, newHtml:String) : void
```

Scaleform version: 4.0.12

Appends the HTML specified by the newHtml parameter to the end of the text of the text field. This method is more efficient than an addition assignment (+=) on an `htmlText` property (such as `txt.htmlText += moreHtml`). The regular += on an `htmlText` property generates the HTML string, appends the new HTML portion and then parses the entire HTML from scratch. This function does incremental HTML parsing, i.e., it parses only the HTML from the `newHtml` string parameter (that is why the HTML in the `newHtml` parameter should be well-formed, which is not necessary for the += operator). It is particularly important for a text field that contains a large amount of content.

Note: This method will not work if a style sheet is applied to the text field.

Parameters
`textField:TextField` – A textfield to append HTML to.
`newHtml:String -` The string with HTML to append to the existing text.

setIMEEnabled() static method

```
static public function setIMEEnabled(textField:TextField, isEnabled:Boolean): void
```

Scaleform version: 4.0.12

Enables/disables IME for the specified textfield. If `isEnabled` is set to `false`, then IME is prevented from being activated in this text field. By default IME is enabled.

Parameters
`textField : TextField` – A textfield to work with.
`isEnabled : Boolean –` If `true` – IME enabled; disabled otherwise.

setVerticalAlign() static method

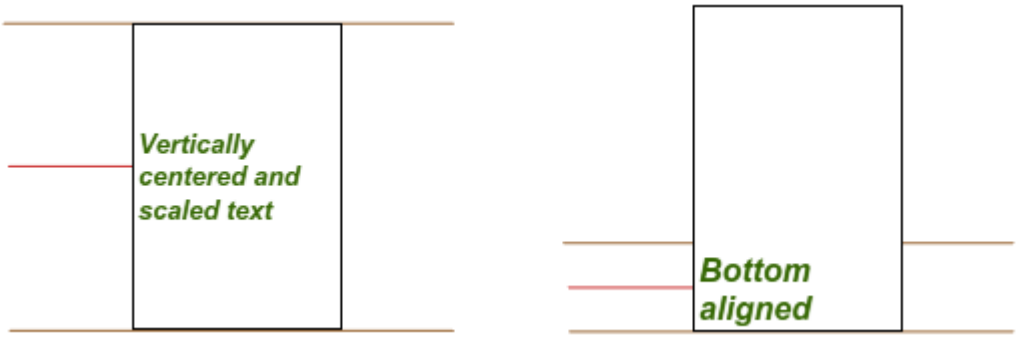
```
public function setVerticalAlign(textField:TextField, valign:String) : void
```

Scaleform version: 4.0.12

Sets the vertical alignment of the text inside the text box. Valid values for the property are the following constants declared in TextFieldEx class:

```
            public static const VALIGN_TOP:String     = "top";
            public static const VALIGN_CENTER:String  = "center";
            public static const VALIGN_BOTTOM:String  = "bottom";
```

If the property is set to `center` then text is centered inside the text box, if set to `bottom`, then text is at the bottom of the text box (see picture below):



The default value is `top`

Parameters

`textField:TextField` - A textfield to set vertical alignment
`valign:String` - Alignment value `("top", "center", "bottom")`.

setImageSubstitutions() static method

```
public function setImageSubstitutions(textField:TextField, substInfo:Object) : void
{}
```

Scaleform version: 4.0.17

Sets image substitutions for substrings to the text field.
Strings substitution works only with images embedded into a SWF; these images also should have assigned linkage in order to have an export name. To embed image into a SWF you need to:

1. Import a bitmap image to the library.
2. Right-click (Windows) or Control-click (Macintosh) the image in the library and select Linkage from the context menu.
3. Select Export for ActionScript and Export in first Frame and type the desired name (for example, myImage) in the Identifier text box.
4. Click OK to set the linkage identifier.

After the image is imported and linkage identifier is assigned, it is necessary to create a `BitmapData` instance. Here is the example of ActionScript code:

```
import flash.display.BitmapData;
var imageBmp:BitmapData = new myImage;
```

If more than one image to be used as a substitution you need to repeat these steps for each image, giving different linkage IDs.

The descriptor of the single substitution is the `Object` with the following members set:

`subString:String`
Specifies the sub-string that will be replaced by image; this member is mandatory. The maximum length of this sub-string is 15 characters.

`image : BitmapData`
Specifies the image; this is mandatory.

`width : Number`
Specifies the width of image on the screen, in pixels. Optional.

`height : Number`
Specifies the height of image on the screen, in pixels. Optional.

`baseLineY : Number`
Specifies the Y-offset of base line in the image, in pixels of original image (without transformation). Optional. By default, this value is equal to image's height; thus, the bottom of the image appears on a baseline.

`id : String`
Specifies the id of the substitution to use as a first parameter for the "updateImageSubstitution" call. Optional.

It is not necessary to keep a reference to the single descriptor object in ActionScript code after `setImageSubstitutions` is called; however, keep it if it is necessary to refer it somewhere in the ActionScript code, since there is no way to get the array of substitutions back from the text field.

Parameters

`textField:TextField` – The text field for image substitution.

`substInfo:Object` – A single substitution descriptor object (see above).

See also:

`updateImageSubstitution()`

Example:

```
var b1:BitmapData = new smile1;
var b2:BitmapData = new smile2;
var b3:BitmapData = new smile3;
var a = new Array;
a[0] = { subString:"=)", image:b1, baseLineY:35, width:20, height:20, id:"sm=)" };
a[1] = { subString:":-)", image:b2, baseLineY:20, id:"sm:-)" };
a[2] = { subString:":\\", image:b3, baseLineY:35, height:100 };
a[3] = { subString:":-\\", image:b1 };
TextFieldEx.setImageSubstitutions(t, a);
```

As soon as a text field contains a substring "=)", without quotes, this substring will be replaced by the image with "smile1" linkage identifier.

updateImageSubstitution () static method

```
public function updateImageSubstitution(textField:TextField, id:String,
image:BitmapData) : void
```

Scaleform version: 4.0.17

Replaces or removes the image for the text substitution previously created by the `setImageSubstitutions` function.

Parameters

`textField:TextField` – The text field for image substitution.

`id:String` – An ID of the substitution, same as `id` member of the descriptor object used for the `setImageSubstitutions` call.

`image:BitmapData` – Specifies the new image; if `null` then the substitution will be removed completely.

See also: `setImageSubstitutions()`

Example:
`TextFieldEx.updateImageSubstitution(t, "sm=)", b3`
The following is an example of animation of embedded images. Update may be done in the onEnterFrame handler or using setInterval. Note, no text reformatting occurs when `updateImageSubstitution` is called; thus, the size of new image should be the same as the old ones.

```
var phase = 0;
var b1a:BitmapData = new smile1a;
var b2a:BitmapData = new smile2a;

addEventListener(Event.ENTER_FRAME, function()
{
   if (phase % 10 == 0)
   {
      if (phase % 20 == 0)
      {
         TextFieldEx.updateImageSubstitution(t, "sm=)", b1);
         TextFieldEx.updateImageSubstitution(t, "sm:-)", b2);
      }
      else
      {
         TextFieldEx.updateImageSubstitution(t, "sm=)", b1a);
         TextFieldEx..updateImageSubstitution(t, "sm:-)",b2a);
      }
   }
   ++phase;
});
```