Autodesk®
Scaleform®

Scaleform 4.2 AS2 Extensions Reference

This document describes ActionScript 2.0 extensions available in Scaleform 4.2.

Authors: Artem Bolgar, Prasad Silva

Version: 4.03
Last Edited: September 17, 2012

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.2

How to Contact Autodesk Scaleform:

Document	Scaleform 4.0 AS2 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. 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. 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 `_global.gfxExtensions` variable to `true`.

```
_global.gfxExtensions = true;
```

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 variable 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 Scaleform 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 Mouse Class Extensions

In addition to supporting the full set of standard `Mouse` class methods, Scaleform also introduces extensions that allow tracking of multiple mouse cursors and detection of the RIGHT, MIDDLE and other mouse buttons (up to 16 at the moment). This chapter will first touch on some of the basic Mouse functionality and then provide a more detailed description of the Scaleform mouse extensions. For more detailed reference for the Mouse object, we recommend developers refer to the Flash Mouse Class documentation.

2.1 Multiple Cursor Support

Some platforms, such as Wii™, require support for more than one mouse cursor. Scaleform 3 and higher supports up to four mouse cursors. The method GFx::Movie::SetMouseCursorCount(unsigned n) sets the number of cursors. The method GFx::Movie::HandleEvent with GFx::MouseEvent object as a parameter should be used to feed mouse events into the Scaleform core. The constructors of GFx::MouseEvent and GFx::MouseCursorEvent have an extra parameter that specifies a zero-based index of mouse.

2.2 Button Events Extensions

Button events such as onRollOver, onRollOut, onDragOver, onDragOut, onPress, onRelease, onReleaseOutside receive two parameters if Scaleform extensions are on.
The first parameter is the zero-based index of mouse that caused the event. Thus, if onPress event was caused by the mouse cursor with an index of 1, then the parameter will contain a value of 1:

```
mc.onPress = function(mouseIdx:Number)
{
      if (mouseIndex == 0)
            . . .
      else if (mouseIndex == 1)
            . . .
      . . .
}
```

The second parameter for onPress and onRelease is numeric property that denotes whether the event was caused by a mouse/cursor or keyboard. The value is -1 if keyboard, and 0 if mouse/cursor. This property is useful for determining the source of the events:

```
mc.onPress = function(mouseIdx:Number, keyboardOrMouse:Number)
```

```
{
      if (keyboardOrMouse == 0)
            . . .
      else
            . . .
      . . .
}
```

The second parameter is optional for onRollOver/Out and onDragOver/Out events. This parameter specifies the index of nested rollover/dragover event over the same character. If this parameter is not declared for the function-handler, then onRollOver/onRollOut and onDragOver/onDragOut event pairs will be fired only once: onRollOver/onDragOver will be fired when the first cursor rolls over the character and onRollOut/onDragOut will be fired when the last cursor leaves it.
If the second parameter is declared for these handlers, then nested onRollOver/onRollOut and onDragOver/onDragOut events will be generated (separately for each mouse cursor) and the parameter represents the zero-based index of nesting: the initial onRollOver/onDragOver event will have 0 as the second parameter; if the second cursor rolls over the same character, then the second onRollOver/onRollOut event will be fired with the second parameter set to 1. If any of the cursors leaves the character then the onRollOut/onDragOut will be fired with the second parameter set to 1; the last onRollOut/onDragOut event will be fired with the second parameter set to 0.
Thus, the declaration of the second parameter changes the way how the onRollOver/onRollOut and onDragOver/onDragOut events are fired. In the case when the second parameter is declared, it is responsibility of user to take care of nested events.

```
mc.onRollOver = function(mouseIdx:Number, nestingIdx:Number)
{
      // do roll over animation only if nestingIdx == 0
      if (nestingIdx == 0)
            doOverAnimation();
      . . .
}
mc.onRollOut = function(mouseIdx:Number, nestingIdx:Number)
{
      // do roll out animation only if nestingIdx == 0
      if (nestingIdx == 0)
            doOutAnimation();
      . . .
}
```

Note, the old-style events such as on(rollOver), on(rollOut), etc. are not provided with any extra parameters, thus, there is no way to distinguish which mouse cursor caused the event. Thus, they are not recommended to be used with multiple mouse cursors.

To support mouse buttons other than the left button for onPress, onRelease, etc., new auxiliary versions of the Button Events have been added:

- onPressAux
- onReleaseAux
- onReleaseOutsideAux
- onDragOver
- onDragOut

These auxiliary event handlers will be invoked if they are defined on the event target. They will only be invoked for buttons with index != 0 (0 index denotes the left mouse button). The regular handlers are always invoked only for the left mouse button, regardless of the existence of the auxiliary event handlers. These handlers also have the same function signature as its standard counterparts, as defined earlier in this section. However, the auxiliary event handlers provide an extra parameter for the button index.

```
mc.onPressAux = function(mouseIdx:Number, keyboardOrMouse:Number, buttonIdx:Number)
{
      if (buttonIdx == 1)  // Right mouse button
            . . .
      . . .
}
```

2.3 Mouse Class Events

In ActionScript, mouse listeners can be installed to receive notifications about mouse movement, left mouse button press and mouse wheel events. In Flash these events are traditionally handled by using the `Mouse.addListener` method to install a listener object that implements the following methods:

- `onMouseMove  = function() { }`
- `onMouseDown  = function() { }`
- `onMouseUp    = function() { }`
- `onMouseWheel = function(delta : Number, targetPath : String) { }`

After the listener object is installed, its appropriate methods will be called whenever their corresponding events occur. In Flash, the `onMouseDown` and `onMouseUp` methods will only be called for the LEFT mouse button, no public interface is provided for receiving other mouse button events.

To address this limitation (and for supporting multiple mouse cursors) Scaleform extends these method calls by allowing them to take extra arguments when `_global.gfxExtensions` variable is set to true. The new function signatures are as follows:

- `onMouseDown = function(button : Number, [targetPath : String], [mouseIdx : Number], [x : Number], [y : Number], [dblClick : Boolean]) { }`
- `onMouseUp = function(button : Number, [targetPath : String], [mouseIdx : Number], [x : Number], [y : Number]) { }`

When the assigned listener function takes at least one extra argument, Scaleform will call that listener method for the RIGHT, MIDDLE and other mouse buttons as well, allowing your logic to detect those events. The value for LEFT is 1, RIGHT is 2 and MIDDLE is 3. Currently, up to 16 buttons are supported. For compatibility, if the function is declared to take no arguments, it will only be called for the LEFT mouse button, which is identical to Flash. Below is an example of how such handlers can be installed:

```
var mouseListener:Object = new Object;

mouseListener.onMouseDown = function(button, target)
{
   trace("mouseDown - button '" + button +
         "'  target = '" + target + "'");
}
mouseListener.onMouseUp = function(button, target)
{
   trace("mouseUp - button '" + button +
         "'  target = '" + target + "'");
}

Mouse.addListener(mouseListener);
```

The new mouse down/up handlers take at least two arguments: button and target. The first argument, button, describes the mouse button that was pressed; you can interpret it by comparing button to the `Mouse["LEFT"]`, `Mouse["RIGHT"]` and `Mouse["MIDDLE"]` extension constants (`Mouse["LEFT"]` syntax is used instead of `Mouse.LEFT` in order to quiet the ActionScript 2.0 compiler). The second argument, target, provides the path to the topmost object under the mouse cursor at the time mouse was pressed; it can be 'undefined' if no such path is available.

In Flash, all extra arguments will always be “undefined”, as the handlers are only called for the left mouse button. You can interpret the undefined value as the LEFT mouse button to ensure compatibility with the Flash player.

For supporting multiple mouse cursors, there are extra parameters for each handler:

```
▪ onMouseMove = function([mouseIdx : Number],
                         [x : Number], [y : Number]) { }
▪ onMouseDown = function(button : Number, [targetPath : String],
                         [mouseIdx : Number], [x : Number], [y : Number],
                         [dblClick : Boolean]) { }
▪ onMouseUp   = function(button : Number, [targetPath : String],
                         [mouseIdx : Number], [x : Number], [y : Number]) { }
▪ onMouseWheel = function(delta : Number, targetPath : String,
                         [mouseIdx : Number], [x : Number], [y : Number]) { }
```

The parameter `mouseIdx` contains the zero-based index of the mouse cursor caused the event. The `x` and `y` parameters contain the coordinates of the mouse cursor (in _root coordinate space). The `dblClick` parameter of `onMouseDown` handler indicates if a double-click occurred. In this case this parameter will contain a `true` value.

2.4 Changing the Mouse Cursor

In Flash there are three types of Mouse cursors:

- *Arrow* - used when the cursor is over most normal objects.
- *Hand* - used over buttons and links if useHandCursor property is true.
- *I-Beam* - used when the cursor is over top of text fields.

While Flash does not provide an easy way to detect when cursor changes occur, Scaleform offers both a C++ API and an ActionScript Mouse class extension that allow you to manage a custom cursor. On the C++ side, you can use the `GFx::StateBag::SetUserEventHandler` function to install an event handler that notifies you whenever mouse cursor changes need to take place. You can use this handler to change either the cursor image drawn by the game engine or the system cursor, as illustrated in FxPlayer.cpp. Alternatively, you can choose to implement a custom cursor fully in ActionScript, with the added benefit of animation and artist control. An example of how this is done is provided in the new *Mouse.fla*/*swf* sample shipped in the SDK.

The Mouse sample implements a customizable mouse cursor as a simple movie clip object on the stage, called 'Cursor_M'. The cursor has several frames identified by the "hand", "arrow" and "ibeam" frame labels; the cursor image can be changed by simply moving the timeline to one of those frames. Whenever mouse movement is detected, the movie clip _x and _y coordinates are adjusted to follow the mouse.

In order to change a cursor type based on the object underneath, the mouse sample overrides the `Mouse.setCursorType` function, which is a Scaleform extension. This function is called with `cursorType` and `mouseIndex` arguments whenever a cursor change needs to take place, allowing such changes to be detected. The mouse sample interprets this sample by comparing cursorType

with one of the mouse constants (`Mouse["ARROW"]`, `Mouse["HAND"]` and `Mouse["IBEAM"]`) and changing the cursor frame appropriately.

Below is the source code responsible for handling multiple custom cursors in the mouse sample. We recommend referring to the actual sample for more details.

```
_global.gfxExtensions = 1;
// Hide system cursor.
Mouse.hide();

var mouseListener:Object = new Object;
mouseListener.onMouseMove = function(mouseIdx, x,y)
{
    if (mouseIdx == undefined)
        mouseIdx = 0;
    if (x == undefined)
    {
        x = _xmouse;
        y = _ymouse;
    }
    var cmc = eval("_root.Cursor_M"+(mouseIdx+1));

    cmc._x = x;
    cmc._y = y;
}
Mouse.addListener(mouseListener);

Mouse["setCursorType"] = function(cursorType, mouseIdx)
{
   if (mouseIdx == undefined)
      mouseIdx = 0;
   var cmc = eval("_root.Cursor_M"+(mouseIdx+1));
   switch(cursorType)
   {
      case Mouse["HAND"]:
         cmc.Cursor.gotoAndPlay("hand");
         break;
      case Mouse["ARROW"]:
         cmc.Cursor.gotoAndPlay("arrow");
         break;
      case Mouse["IBEAM"]:
         cmc.Cursor.gotoAndPlay("ibeam");
         break;
      default: return;
   }
}
```

```
// Override Mouse.show and Mouse.hide functions so
// that they affect our cursor.
ASSetPropFlags(Mouse, "show,hide", 0, 7);

Mouse.show = function(mouseIdx)
{
    if (mouseIdx == undefined)
        mouseIdx = 0;
    var cmc = eval("_root.Cursor_M"+(mouseIdx+1));
    cmc._visible = true;
}
Mouse.hide = function(mouseIdx)
{
    if (mouseIdx == undefined)
        mouseIdx = 0;
    var cmc = eval("_root.Cursor_M"+(mouseIdx+1));
    cmc._visible = false;
}
```

2.5 Mouse Class ActionScript Extensions

Mouse ActionScript class has been extended in Scaleform by adding several static methods and properties. In ActionScript 1.0 it is possible to refer to extensions directly, as in `Mouse.setCursorType(…)`. ActionScript 2.0 is less tolerant and it will "complain" about such calls. To quiet the ActionScript 2.0 compiler you should use the alternative access syntax as follows: `Mouse["setCursorType"](..)`.

getButtonsState() static method

```
public function getButtonsState(mouseIndex : Number) : Number
```

Scaleform version: 2.2

Returns state of mouse buttons. The returned value is a bit-mask where the index of bit equals the zero-based mouse button index. If the corresponding bit is set then the button is pressed.

Parameters

`mouseIndex : Number` – Zero-based mouse index.

getTopMostEntity() static method

```
public function getTopMostEntity([mouseIndex : Number]) : Object
public function getTopMostEntity(testAll : Boolean,
                                 [mouseIndex : Number]) : Object
public function getTopMostEntity(x : Number, y : Number) : Object
public function getTopMostEntity(x : Number, y : Number, testAll: Boolean) : Object
```

Scaleform version: 1.2.34, multiple mouse cursors support since 2.2

This static method returns a target character underneath the mouse cursor or at the specified coordinates. This method also may differ between characters with button handlers set (e.g., `onPress`, `onMouseDown`, `onRollOver`) 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, `Mouse.getTopMostEntity(x, y).hitTest(x, y, true) == true`.

Parameters

`mouseIndex : Number` – Zero-based mouse index.

`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` – Alternative coordinates at which to look for a character.

Example:

```
var listenerObj = new Object;
listenerObj.onMouseMove = function()
{
   var target = Mouse["getTopMostEntity"]();
   trace("Mouse moved, target = "+target);
}
Mouse.addListener(listenerObj);
var target = Mouse["getTopMostEntity"](480, 10);
trace("The character at the (480, 10) is " + target);
```

getPosition static method

```
public function getPosition(mouseIndex : Number) : flash.geom.Point
```

Scaleform version: 2.2

This method returns coordinates of the corresponding mouse cursor, in _root coordinate space. The returned value is an instance of flash.geom.Point.

Parameters

`mouseIndex : Number` – Zero-based mouse index.

setCursorType() static method

```
public function setCursorType(cursorType : Number, [mouseIndex : Number]) : void
```

Scaleform version: 1.2.32

This static method changes the mouse cursor according to the parameter `cursorType`. See "Changing mouse cursor" for details.

Parameters

`cursorType : Number` – Indicates type of cursor, one of the following: Mouse.ARROW, Mouse.HAND, Mouse.IBEAM.

`mouseIndex : Number` – Zero-based mouse index.

3 Button Class Extensions

hitTestDisable property

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

Scaleform version: 2.1.51

When set to `true`, the `MovieClip.hitTest` function will ignore this button during hit test detection. In addition, all other mouse events are not propagated to the button.

The default value is `false`.

See also:

```
TextField.hitTestDisable
MovieClip.hitTestDisable
```

topmostLevel property

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

Scaleform version: 2.1.50

If the property is set to `true` then this character will be 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 as follows:

- Up to Scaleform 3.0.71, the draw order of characters marked as "topmostLevel" depend on the order of setting this property to `true` (thus, the character first marked as topmost will be drawn first);
- Starting from Scaleform 3.0.72, 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.

See also:

```
TextField.topmostLevel
MovieClip.topmostLevel
```

focusGroupMask property

`focusGroupMask : Number`

Scaleform version: 3.3.84

This property sets a bitmask to a stage 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:

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

button1.focusGroupMask = 0x1; // bit 0 – focus group 0
movieclip2.focusGroupMask = 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.

4 MovieClip Class Extensions

hitTest () method

```
public hitTest(x:Number, y:Number, [shapeFlag:Boolean],
               [ignoreInvisibleChildren:Boolean]):Boolean
```

Scaleform version: 2.1.51

The standard 3-parameters `hitTest` has an optional boolean parameter as an extension, indicating whether to ignore invisible children clips or not. The default Flash behavior of hitTest is to return `true`, even if the point at x, y coordinates belongs to an invisible child clip (i.e. its _visible property is set to `false`; child clips with _alpha set to zero are not treated as invisible).

Parameters

`ignoreInvisibleChildren:Boolean` – Should be set to `true` to ignore all invisible child clips.

hitTestDisable property

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

Scaleform version: 1.2.32

When set to `true`, the `MovieClip.hitTest` function will ignore this movie clip during hit test detection. In addition, all other mouse events are not propagated to the movie clip.

The default value is `false`.

See also:

`TextField.hitTestDisable`
`Button.hitTestDisable`

noAdvance property

```
noAdvance : Boolean
```

Scaleform version: 2.1.52

If set to `true`, this property turns off advancing of this movie clip and all of its children. This might be used to improve performance. Note: Flash always advances movie clips (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.

See also:

```
_global.noInvisibleAdvance
```

topmostLevel property

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

Scaleform version: 2.1.50

If the property is set to `true` then this character will be 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 as follows:

- Up to Scaleform 3.0.71, the draw order of characters marked as "topmostLevel" depend on the order of setting this property to `true` (thus, the character first marked as topmost will be drawn first);
- Starting from Scaleform 3.0.72, 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.

See also:

```
TextField.topmostLevel
Button.topmostLevel
```

rendererString property

```
rendererString:String  [read-write]
```

Scaleform version: 2.2.55

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

There is no default value.

Example:

```
myMovieInstance.rendererString = "SHADER_Blur"
```

rendererFloat property

`rendererFloat:Number  [read-write]`

Scaleform version: 2.2.55

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

There is no default value.

Example:

```
myMovieInstance.rendererFloat = 4; // eg: C++ enum value
```

disableBatching property

`disableBatching:Boolean`

Scaleform version: 4.0.17

This property disables batching of mesh generation for custom drawing.

focusGroupMask property

`focusGroupMask : Number`

Scaleform version: 3.3.84

This property sets a bitmask to a stage 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:

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

button1.focusGroupMask = 0x1; // bit 0 – focus group 0
movieclip2.focusGroupMask = 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.

5 NetStream Class Extensions

Scaleform adds functionality to the NetStream class, which allows manipulation with subtitles and additional audio tracks that are embedded into a video file.

onMetaData event handler

```
onMetaData = function(info:Object) { }
```

Scaleform version: 3.0.63

The event handler receives information on the video file being played and includes two additional object properties in Scaleform. These properties are used to retrieve information about subtitle and audio tracks encoded into a video file.

The object that is passed to `onMetaData` event handler has two additional read-only properties.

1. `audioTracks` – Array of objects describing audio tracks encoded into the video file
   Each object in `audioTracks` array has the following properties:
   - `channelsNumber` – Number of channels in the audio track .(1–mono, 2–stereo, 6–5.1 surround sound)
   - `totalSamples` – Number of sound samples in this track.
   - `trackIndex` – Track index number. This property is used to select an audio track by assigning it to `NetStream.audioTrack` property (see the example below).
   - `sampleRate` – Sound sample rate.

2. `subtitleTracksNumber` – Number of subtitle tracks available in the video file.

audioTrack property

```
audioTrack:Number  [read-write]
```

Scaleform version: 3.0.63

This property is used to set/get the currently playing audio track encoded into a video file.

Example:

```
var ns:NetStream = new NetStream(nc);
```

```
var audioTracks;

ns.onMetaData = function(info:Object)
{
      audioTracks = info.audioTracks;
}

if (audioTracks != undefined && audioTracks.length > 0)
      ns.audioTrack = audioTracks[0].trackIndex;
```

subtitleTrack property

```
subtitleTrack:Number  [read-write]
```

Scaleform version: 3.0.63

The property allows the setting and retrieving of the current subtitle track. To turn off the subtitle set this property to 0.

onSubtitle event handler

```
onSubtitle = function(msg:String) {}
```

Scaleform version: 3.0.63

This event handler is called when a subtitle message is ready to be shown.

Example:

```
var ns:NetStream = new NetStream(nc);
var subtitlesNumber = 0;

ns.onMetaData = function(info:Object)
{
      subtitlesNumber = info. subtitleTracksNumber;
}

ns.onSubtitle = function(msg:String)
{
      sbTextField.text = msg;
}
if (subtitlesNumber > 0)
      ns.subtitleTrack = 1;
```

setNumberOfFramePools function

```
public function setNumberOfFramePools(pools : Number) : Void
```

Scaleform version: 3.0.68

This function sets the number of internal video buffers. Video buffers are used to save video decoding output and are called “frame pools”. When the CPU load of decoding is unstable, having many frame pools helps smooth out playback. The default value is 1.

This method should be called before a video starts playing (i.e., before a NetStream.play() call).

setReloadThresholdTime function

```
public function setReloadThresholdTime(reloadTime : Number) : Void
```

Scaleform version: 3.0.68

This function sets the reload timing in seconds. The Scaleform video library submits a next file read request when the data size in the input buffer is less than the reload threshold. This threshold time is automatically determined according to the bitrate of the video file and the reloading time set. The default value of reload timing is 0.8 (seconds).

While reading game data during video playback, the number of seeking requests can be reduced by using this method. For example, when you set the Netstream buffer time (NetStream.setBufferTime() method) to 2 seconds and set the threshold time to 1 second, the game data can be read continuously for about 1 second. But the reading of game data should be completed until/before the reload timing occurs. When the game data is large, it may have to be read by chunks. And if the game data reading is not completed before the reload timing, the movie playback will “stutter,” as the video data buffer is empty.

This method should be called before a video starts playing (i.e., before a NetStream.play() call).

6 Selection Class Extensions

The Selection class, among its other functionalities, allows you to manage input focus among text fields, movie clips and buttons.

The Scaleform extends the focus functionality of the Selection class. In ActionScript 1.0 it is possible to refer to Selection extension functions directly, as in `Selection.captureFocus()`. ActionScript 2.0 is less tolerant and it will “complain” about such calls. To quiet the ActionScript 2.0 compiler you should use the alternative access syntax as follows: `Selection["captureFocus"]()`.

alwaysEnableArrowKeys static property

```
alwaysEnableArrowKeys : Boolean
```

Scaleform version: 1.2.34

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`.

Example:

```
Selection["alwaysEnableArrowKeys"] = true;
```

alwaysEnableKeyboardPress static property

```
alwaysEnableKeyboardPress : Boolean
```

Scaleform version: 3.0.63

This static property allows to fire onPress / onRelease events by pressing SPACE or ENTER keys, even when the `_focusrect` property is set to `false` (applied when the focus is captured). By default, Flash does not allow you to press buttons by keyboard if the yellow focus rectangle is disabled via `_focusrect = false;`. To change this behavior, set the `alwaysEnableKeyboardPress` property to `true`.

Example:

```
Selection["alwaysEnableKeyboardPress"] = true;
```

captureFocus() static method

```
public function captureFocus([doCapture:Boolean, controllerIdx:Number]) : void
```

Scaleform version: 1.2.34

This static method is used to capture or release keyboard focus programmatically.

The `captureFocus` with the `doCapture` parameter set to `true` (or without a parameter at all) captures keyboard focus ,and allows the arrow keys to change focus, which is similar to the first time the Tab key is pressed.

The `captureFocus` with the `doCapture` parameter set to `false` releases keyboard focus, and acts like a mouse move when the focus rectangle is on.

Parameters

`doCapture:Boolean` – Optional, indicates whether keyboard focus should be captured or released. If this parameter is omitted then `captureFocus` behaves like this parameter is set to `true`.

controllerIdx:Number – Optional, indicates which keyboard/controller is used for the capture operation. By default controller 0 is used.

Example:

```
Selection["captureFocus"](); // same as passing "true"
Selection["captureFocus"](false);
```

disableFocusAutoRelease static property

```
disableFocusAutoRelease : Boolean
```

Scaleform version: 1.2.34

This static property is used to control whether a mouse movement releases keyboard focus (the standard behavior in Flash) or not. By default, if focus is captured and the yellow rectangle is

displayed (allowing arrow keys to change the focus), the mouse move releases the focus. To prevent this behavior, set this static property to `true`.

Example:

```
Selection["disableFocusAutoRelease"] = true;
```

disableFocusKeys static property

```
disableFocusKeys : Boolean
```

Scaleform version: 2.2.58

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

Example:

```
Selection["disableFocusKeys"] = true;
```

See also:

```
moveFocus()
```

disableFocusRolloverEvent static property

```
disableFocusRolloverEvent : Boolean
```

Scaleform version: 2.0.37

This static property is used to disable rollover/out event firing if focus is changed by keys. By default, if focus is changed by pressing arrow keys rollover/rollout events are fired. To prevent this behavior, set this static property to `true`.

Example:

```
Selection["disableFocusRolloverEvent "] = true;
```

modalClip static property

```
modalClip : MovieClip
```

Scaleform version: 2.2.58

This static property sets the specified movie clip as a “modal” clip from focus management point of view. This means that focus keys (TAB, Shift-TAB and arrow keys) will move focus rectangle only inside the specified movie clip, i.e., only across “tabable” children of the movie clip.

Example:

```
Selection["modalClip"] = _root.mc;
...
Selection["modalClip"] = undefined;
```

See also:

`disableFocusKeys`
`moveFocus()`

moveFocus() static method

```
public function moveFocus(keyToSimmulate : String [, startFromMovie:Object,
                includeFocusEnabledChars : Boolean = false,
                controllerIdx :Number]) : Object
```

Scaleform version: 2.2.58

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:Object` – 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:Number – Index of the controller used for the operation. If not specified, then the default controller (controller 0) is used.

Example:

```
Selection["moveFocus"]("up");
Selection["moveFocus"]("tab", _root.mc);
Selection["moveFocus"]("tab", _root.mc, true);
Selection["moveFocus"]("tab", null, true);
```

Returns

Returns newly focused character or `undefined` if the character cannot be found.

See also:

`disableFocusKeys`
`modalClip`

findFocus() static method

```
public function findFocus(keyToSimmulate : String [, parentMovie:Object, loop :
                Boolean, startFromMovie:Object, includeFocusEnabledChars : Boolean,
                controllerIndex : Number]) : Object
```

Scaleform version: 3.3.84

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` - 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 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:Object` – 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.

`controllerIndex` - 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.

Example:

```
var a = Selection["findFocus"]("up");
```

Returns

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

See also:

`disableFocusKeys`
`setModalClip`
`getModalClip`

setModalClip() static method

```
public function setModalClip(modalClip : Object, controllerIndex : Number)
```

Scaleform version: 3.3.84

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

`modalClip` – A modal clip.

`controllerIndex` - A zero base index of the controller.

getModalClip() static method

```
public function getModalClip(controllerIndex : Number) : Object
```

Scaleform version: 3.3.84

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

Parameters

`controllerIndex` - Zero base index of the controller.

Returns

A modal clip or undefined if not found.

setControllerFocusGroup () static method

```
public function setControllerFocusGroup (controllerIndex : Number, focusGroupIdx :
                                                                    Number) : Boolean
```

Scaleform version: 3.3.84

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

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

`focusGroupIdx` – Zero-base index of the focus group.

Example:

```
Selection["setControllerFocusGroup"](0,0);
```

```
Selection["setControllerFocusGroup"](1,1);
Selection["setControllerFocusGroup"](2,1);
```

Returns

Returns true if successful.

getControllerFocusGroup () static method

```
public function getControllerFocusGroup (controllerIndex : Number) : Number
```

Scaleform version: 3.3.84

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.

getFocusArray() static method

```
public function getFocusArray(mc : Object) : Array
```

Scaleform version: 3.3.84

This static method returns an array of controller indices that currently has focus on the specified movieclip/button/text field.

Parameters

`mc` - A movie clip, button or text field.

Returns

Array of zero-based indices (Numbers).

getFocusBitmask() static method

```
public function getFocusBitmask(mc : Object) : Number
```

Scaleform version: 3.3.84

This static method returns a bitmask where each bit represents a controller that currently has focus on the specified movieclip/button/textfield.

Parameters

`mc` - A movie clip, button or text field.

Returns

A bitmask of controllers.

getControllerMaskByFocusGroup () static method

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

Scaleform version: 3.3.84

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.

numFocusGroups property

```
numFocusGroups : Number
```

Scaleform version: 3.3.84

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.

7 TextField Class Extensions

Scaleform introduces many extensions to the TextField class, making it more flexible and functional than the built-in TextField class in Flash 8. Some of the provided extensions replicate the same or similar behavior as TextField in Flash 9. Additional extensions provide functionality for better control of IME, alignment, text size and appearance, text filter effects (e.g., shadow, blur), embedded images, selection and clipboard operations.

7.1 General Functionality

This section describes general purpose extension properties and methods.

autoFit property

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

Scaleform version: 2.0.39

Sets on/off font auto-hinting. The default value is `false`.

appendText () method

```
public function appendText(newText:String):void
```

Scaleform version: 2.0.37

Appends the string specified by the `newText` parameter to the end of the text of the text field. This method is significantly more efficient than an addition assignment (+=) on a text property (e.g., `my_txt.text += appendingText`), especially for a text field that contains a large amount of text.

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

Parameters

`newText:String` - The string to append to the existing text.

See also:

`appendHtml()`

appendHtml () method

```
public function appendHtml(newHtml:String):void
```

Scaleform version: 2.0.37

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

`newHtml:String` - The string with HTML to append to the existing text.

See also:

`appendText()`

caretIndex property

```
caretIndex:Number  [read-only]
```

Scaleform version: 2.0.37

This property represents the index of the insertion point (caret or cursor) position. If the text field is not focused and caret position is not displayed (by a blinking cursor) it represents the last caret position before the text field loses the focus; or 0, if the text field has not had focus. This property contains the same value as the method `Selection.getCaretIndex()` returns; however, the `caretIndex` is accessible even if the text field is not focused, whereas the `Selection.getCaretIndex()` returns -1 in this case.

The caret index is zero-based (so, the first position is 0).

See also:

`Selection.getCaretIndex()`

getCharBoundaries () method

```
public function getCharBoundaries(charIndex:Number): flash.geom.Rectangle
```

Scaleform version: 2.0.37

This method returns a rectangle that is the bounding box of the character. Note, this method returns the rectangle, calculated with using advance values of each glyph. This means that the returned rectangle wouldn't be the exact one (see the picture; the red rectangles demonstrate boundaries for the 'a', 'w' and 'k'):

Parameters

`charIndex:Number` - The zero-based index value for the character (for example, the first position is 0, the second position is 1, and so on)

Returns

flash.geom.Rectangle - A rectangle with x and y minimum and maximum values defining the bounding box of the character. Coordinates are in the text field's coordinate space, i.e., the (0,0) point corresponds to the text field's top left corner.

See also:

`getExactCharBoundaries()`

getExactCharBoundaries () method

```
public function getExactCharBoundaries(charIndex:Number): flash.geom.Rectangle
```

Scaleform version: 2.0.37

This method returns a rectangle that is the exact bounding box of the character. Note: This method returns the rectangle, calculated with using real width of each glyph (in contrary to `getCharBoundaries()` that uses advance values). The height of the rectangle is the height of the entire line. See the picture: the red rectangles demonstrate exact boundaries for the 'a', 'w' and 'k':

Parameters

`charIndex:Number` - The zero-based index value for the character (e.g., the first position is 0, the second position is 1)

Returns

flash.geom.Rectangle - A rectangle with x and y minimum and maximum values defining the exact bounding box of the character. Coordinates are in the text field's coordinate space, i.e., the (0,0) point corresponds to the text field's top left corner.

See also:

`getCharBoundaries()`

getCharIndexAtPoint () method

```
public function getCharIndexAtPoint(x:Number, y:Number): Number
```

Scaleform version: 2.0.37

This method returns the zero-based index value of the character at the point specified by the `x` and `y` parameters.

Parameters

`x:Number` - The x coordinate of the character.

`y:Number` - The y coordinate of the character.

Returns

Number - The zero-based index value of the character (e.g., the first position is 0, the second position is 1). Returns -1 if the point is not over any character.

getFirstCharInParagraph () method

```
public function getFirstCharInParagraph(charIndex:Number): Number
```

Scaleform version: 2.0.37

This method returns the index of the first character in the paragraph that contains the character at the `charIndex` index.

Parameters

`charIndex:Number` - The zero-based index value of the character (e.g., the first character is 0, the second character is 1).

Returns

Number - The zero-based index value of the first character in the same paragraph.

getLineIndexAtPoint () method

```
public function getLineIndexAtPoint(x:Number, y:Number): Number
```

Scaleform version: 2.0.37

This method returns the zero-based index value of the line at the point specified by the `x` and `y` parameters.

Parameters

`x:Number` - The x coordinate of the line.

`y:Number` - The y coordinate of the line.

Returns

Number - The zero-based index value of the line (e.g., the first line is 0, the second line is 1). Returns -1 if the point is not over any line.

getLineLength() method

```
public function getLineLength(lineIndex:Number): Number
```

Scaleform version: 2.0.37

This method returns the number of characters in a specific text line.

Parameters

`lineIndex:Number` - The zero-based index value of the line (e.g., the first line is 0, the second line is 1).

Returns

Number - The number of characters in the line.

getLineMetrics() method

```
public function getLineMetrics(lineIndex:Number): Object
```

Scaleform version: 2.0.43

This method returns metrics information about a given text line. The returning object will contain the following members set:

`ascent : Number`
: The ascent value of the text is the length from the baseline to the top of the line height in pixels.

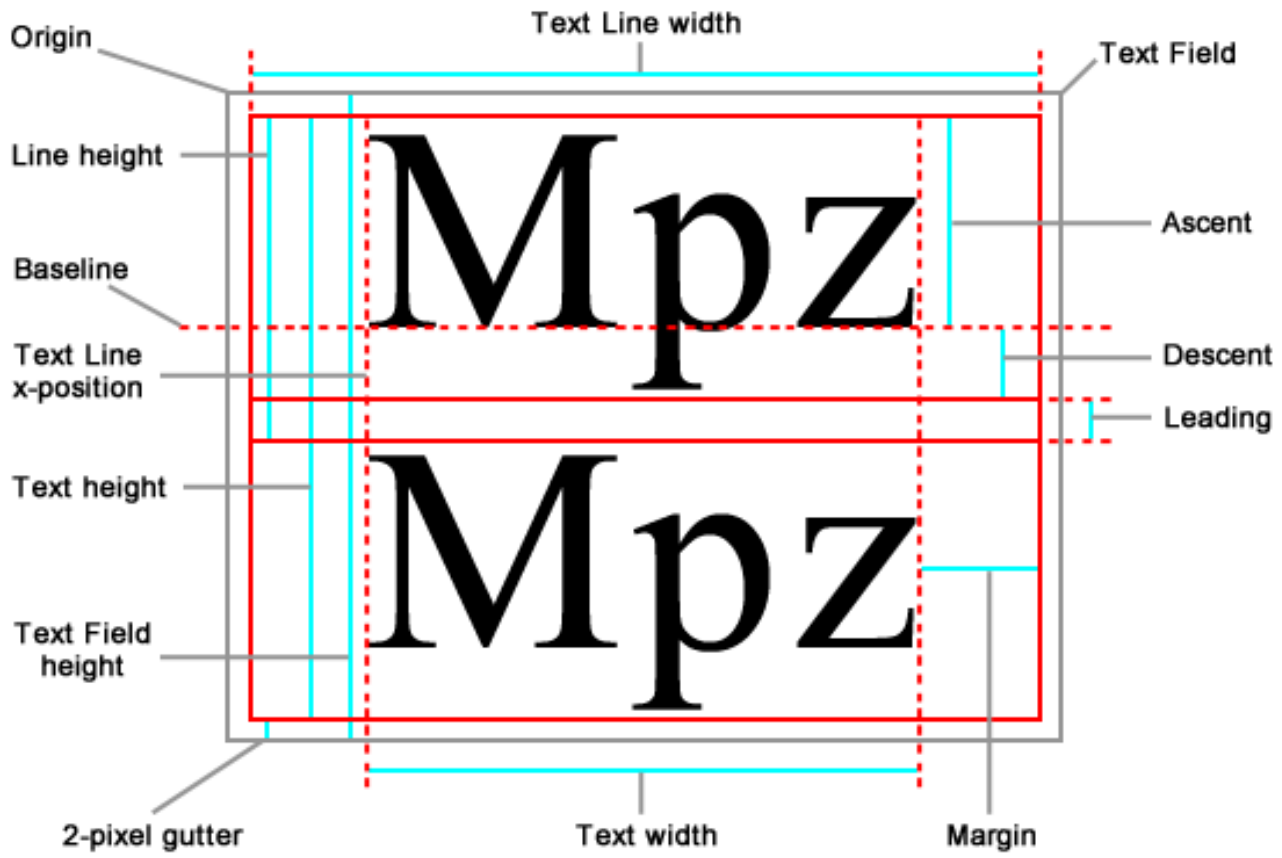
`descent : Number`
: The descent value of the text is the length from the baseline to the bottom depth of the line in pixels.

`height : Number`
: The height value of the text of the selected lines (not necessarily the complete text) in pixels.

`leading : Number`
: The leading value is the measurement of the vertical distance between the lines of text.

`width : Number`
: The width value is the width of the text of the selected lines (not necessarily the complete text) in pixels.

`x : Number`
: The `x` value is the left position of the first character in pixels.



Parameters

`lineIndex:Number` - The zero-based index value of the line (e.g., the first line is 0, the second line is 1).

Returns

Number – An object with metrics information.

getLineOffset () method

```
public function getLineOffset(lineIndex:Number): Number
```

Scaleform version: 2.0.37

This method returns the character index of the first character in the line that the `lineIndex` parameter specifies.

Parameters

`lineIndex:Number` - The zero-based index value of the line (e.g., the first line is 0, the second line is 1).

Returns

Number - The zero-based index value of the first character in the line.

getLineText () method

```
public function getLineText(lineIndex:Number): String
```

Scaleform version: 2.0.43

This method returns the text of the line specified by the `lineIndex` parameter.

Parameters

`lineIndex:Number` - The zero-based index value of the line (e.g., the first line is 0, the second line is 1).

Returns

String - The text string contained in the specified line.

hitTestDisable property

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

Scaleform version: 1.2.32

When set to `true`, the `MovieClip.hitTest` function will ignore this text field during hit test detection. In addition, all other mouse events are not propagated to the text field.

The default value is `false`.

noTranslate property

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

Scaleform version: 1.2.34

If set to true, prevents `GFxTranslator::Translate` callback from being called for the text field.

The default value is `false`.

numLines property

```
numLines:Number  [read-only]
```

Scaleform version: 2.0.43

This property represents the number of text lines in a multi-line text field. If the `wordWrap` property is set to `true`, the number of lines increases when text wraps.

topmostLevel property

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

Scaleform version: 2.1.50

If the property is set to `true` then this character will be displayed on the top of all other characters regardless of its depth. This property is useful for implementing custom mouse cursors and popups that need to be drawn above objects from all levels. The default value is `false`.

In case of marking several characters as "topmostLevel", the draw order is as follows:

- Up to Scaleform 3.0.71, the draw order of characters marked as "topmostLevel" depend on the order of setting this property to `true` (thus, the character first marked as topmost will be drawn first);
- Starting from Scaleform 3.0.72, 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.

See also:

```
MovieClip.topmostLevel
Button.topmostLevel
```

focusGroupMask property

```
focusGroupMask : Number
```

Scaleform version: 3.3.84

This property sets a bitmask to a stage 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:

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

button1.focusGroupMask = 0x1; // bit 0 – focus group 0
movieclip2.focusGroupMask = 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.

7.2 Selection and Clipboard Operations

This section describes extensions for selection and clipboard operations.

alwaysShowSelection property

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

Scaleform version: 2.1.45

By default, if the text field is not in focus, Scaleform does not highlight the selection. When this property is set to `true` and the text field is not in focus, Scaleform highlights the selection in the text field in gray. Colors of active and inactive selection might be overridden.

The default value is `false`.

See also:

- `selectionBkgColor`
- `selectionTextColor`
- `inactiveSelectionBkgColor`
- `inactiveSelectionTextColor`

copyToClipboard () method

```
public function copyToClipboard([richClipboard:Boolean], [startIndex:Number],
[endIndex:Number]):void
```

Scaleform version: 2.1.45

The method copies the text to clipboard. All parameters are optional.

Parameters

`richClipboard:Boolean` – If the value is true then text with styles will be copied to the clipboard. Default value is equal to the `useRichTextClipboard` property.

`startIndex:Number` – Start index of text segment being copied. Default value is equal to `selectionBeginIndex`.

`endIndex:Number` – End index of text segment being copied. Default value is equal to `selectionEndIndex`.

See also:

`useRichTextClipboard`
`selectionBeginIndex`
`selectionEndIndex`
`cutToClipboard()`
`pasteFromClipboard()`

cutToClipboard () method

```
public function cutToClipboard([richClipboard:Boolean], [startIndex:Number],
[endIndex:Number]):void
```

Scaleform version: 2.1.45

The method copies the text to clipboard and removes it from the text field. All parameters are optional.

Parameters

`richClipboard:Boolean` – If true then text with styles will be copied to the clipboard. Default value is equal to the `useRichTextClipboard` property.

`startIndex:Number` – Start index of text segment being copied. Default value is equal to `selectionBeginIndex`.

`endIndex:Number` – End index of text segment being copied. Default value is equal to `selectionEndIndex`.

See also:

```
useRichTextClipboard
selectionBeginIndex
selectionEndIndex
copyToClipboard()
pasteFromClipboard()
```

inactiveSelectionBkgColor property

```
inactiveSelectionBkgColor:Number  [read-write]
```

Scaleform version: 2.1.45

Specifies the background color of the inactive selection. It is actual only if the `alwaysShowSelection` property is set to `true`. The color is specified in the following format: 0xAARRGGBB, in which AA is the hexadecimal representation of the alpha channel component [0...255], RR is the hexadecimal representation of the red component [0...255], BB is the hexadecimal representation of the blue component [0...255] and GG is the hexadecimal representation of the green component [0...255].

Note, make sure the alpha channel is not set to 0, because in this case nothing will be drawn (since alpha set to 0 means complete transparency). Thus, this color is a little bit different from the regular colors used in Flash for a background color. For example, to set this property to full intense red color it is necessary to use the value 0xFFFF0000 (alpha and red set to 0xFF (255)), but for regular “backgroundColor” property it is enough to use the value 0xFF0000.

See also:

```
alwaysShowSelection
inactiveSelectionTextColor
selectionBkgColor
selectionTextColor
```

inactiveSelectionTextColor property

```
inactiveSelectionTextColor:Number  [read-write]
```

Scaleform version: 2.1.45

Specifies the text color of inactive selection. It is actual only if the `alwaysShowSelection` property is set to true. The color is specified in the following format: 0xAARRGGBB, in which AA is the hexadecimal representation of the alpha channel component [0...255], RR is the hexadecimal representation of the red component [0...255], BB is the hexadecimal representation of the blue component [0...255] and GG is the hexadecimal representation of the green component [0...255].

Note: make sure the alpha channel is not set to 0, because in this case nothing will be drawn (since alpha set to 0 means complete transparency). Thus, this color is a little bit different from the regular colors used in Flash for text color. For example, to set this property to full intense red color it is necessary to use the value 0xFFFF0000 (alpha and red set to 0xFF (255)), but for regular “textColor” property it is enough to use the value 0xFF0000.

See also:

`alwaysShowSelection`
`inactiveSelectionBkgColor`
`selectionBkgColor`
`selectionTextColor`

noAutoSelection property

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

Scaleform version: 2.1.45

If set to `true`, prevents auto selection when focus is transferred to a text field.

The default value is `false`.

pasteFromClipboard () method

```
public function pasteFromClipboard([richClipboard:Boolean], [startIndex:Number],
[endIndex:Number]):void
```

Scaleform version: 2.1.45

The method pastes the text from the clipboard. All parameters are optional.

Parameters

`richClipboard:Boolean` – If true then text with styles will be pasted from the clipboard. Default value is equal to `useRichTextClipboard` property.

`startIndex:Number` – Start index of the segment being replaced by the text from the clipboard. Default value is equal to `selectionBeginIndex`.

`endIndex:Number` – End index of the segment being replaced by the text from the clipboard. Default value is equal to `selectionEndIndex`.

See also:

`useRichTextClipboard`
`selectionBeginIndex`

```
selectionEndIndex
copyToClipboard()
cutToClipboard()
```

selectionBeginIndex property

```
selectionBeginIndex:Number  [read-only]
```

Scaleform version: 2.1.45

This property represents the zero-based character index value of the first character in the current selection. If no text is selected, this property is the value of `caretIndex`. This property contains the same value as the method `Selection.getBeginIndex()` returns; though, the `selectionBeginIndex` is accessible even if the text field is not focused, whereas the `Selection.getBeginIndex()` returns -1 in this case.

See also:

```
caretIndex
selectionEndIndex
Selection.getBeginIndex()
```

selectionEndIndex property

```
selectionEndIndex:Number  [read-only]
```

Scaleform version: 2.1.45

This property represents the zero-based character index value of the last character in the current selection. If no text is selected, this property is the value of `caretIndex`. This property contains the same value as the method `Selection.getEndIndex()` returns; though, the `selectionEndIndex` is accessible even if the text field is not focused, whereas the `Selection.getEndIndex()` returns -1 in this case.

See also:

```
caretIndex
selectionBeginIndex
Selection.getEndIndex()
```

selectionBkgColor property

```
selectionBkgColor:Number  [read-write]
```

Scaleform version: 2.1.45

Specifies the background color of the active selection. The color is specified in the following format: 0xAARRGGBB, in which AA is the hexadecimal representation of the alpha channel component [0..255], RR the hexadecimal representation of the red component [0..255], BB the hexadecimal representation of the blue component [0..255] and GG the hexadecimal representation of the green component [0..255].

Note: make sure the alpha channel is not set to 0, because in this case nothing will be drawn (since alpha set to 0 indicates complete transparency). Thus, this color is a little bit different from the regular colors used in Flash for background color. For example, to set this property to full intense red color it is necessary to use the value 0xFFFF0000 (alpha and red set to 0xFF (255)), but for the regular `BkgColor` property it is enough to use the value 0xFF0000.

See also:

```
inactiveSelectionTextColor
inactiveSelectionBkgColor
selectionTextColor
```

selectionTextColor property

```
selectionTextColor:Number  [read-write]
```

Scaleform version: 2.1.45

Specifies the text color of the active selection. The color is specified in the following format: 0xAARRGGBB, in which AA is the hexadecimal representation of the alpha channel component [0..255], RR the hexadecimal representation of the red component [0..255], BB the hexadecimal representation of the blue component [0..255] and GG the hexadecimal representation of the green component [0..255].

Note: make sure the alpha channel is not set to 0, because in this case nothing will be drawn (since alpha set to 0 indicates complete transparency). Thus, this color is a little bit different from the regular colors used in Flash for text color. For example, to set this property to full intense red color it is necessary to use the value 0xFFFF0000 (alpha and red set to 0xFF (255)), but for regular `textColor` property it is enough to use the value 0xFF0000.

See also:

```
inactiveSelectionBkgColor
```

```
inactiveSelectionBkgColor
selectionBkgColor
```

useRichTextClipboard property

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

Scaleform version: 2.1.45

Specifies whether to copy and paste the text formatting along with the text. When set to `true`, Scaleform will also copy and paste formatting (e.g., alignment, bold and italics) when you copy and paste between text fields. Both the origin and destination text fields for the copy and paste procedure must have `useRichTextClipboard` set to `true`.

The default value is `false`.

7.3 Text Size and Alignment

This section describes extensions that control the size of text and text alignment. In addition to standard alignment (left, right, center) Scaleform provides vertical alignment, as well as vertical auto-size functionality.

fontScaleFactor property

```
fontScaleFactor:Number  [read-write]
```

Scaleform version: 2.0.43

This property specifies the font scale factor for all fonts across the text field. All font sizes might be increased or decreased by multiplying the font size by the `fontScaleFactor` value.

Default value is 1.0.

textAutoSize property

```
textAutoSize:String  [read-write]
```

Scaleform version: 2.0.43

`textAutoSize` enables automatic resizing of text font size. Valid values for this property are `none`, `shrink`, and `fit`. If this mode is on (`shrink` or `fit`) and if text doesn't fit in a text field then size of the text will be decreased proportionally to fit whole text in the text field, thus, no scrolling is necessary. If text size becomes too small (font size is about 5 pt) then the default scrolling logic is still used and no further font size decrease is performed.

Setting textAutoSize to `fit` mode will enable increasing of the font size until the text field is filled with text. `Shrink` mode can only decrease the font size when text doesn't fit in the text field; the original font size is not increased.

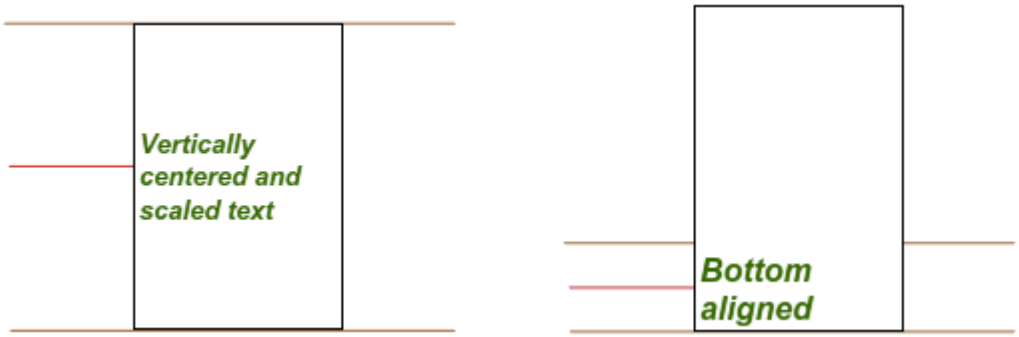
The default value is `none`.

verticalAlign property

`verticalAlign:String  [read-write]`

Scaleform version: 2.0.43

Sets the vertical alignment of the text inside the text box. Valid values for the property are: `none` (or `top` that is the same as `none`), `bottom` and `center`. 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 `none`.

See also:

`verticalAutoSize`
`TextField.align`

verticalAutoSize property

`verticalAutoSize:String  [read-write]`

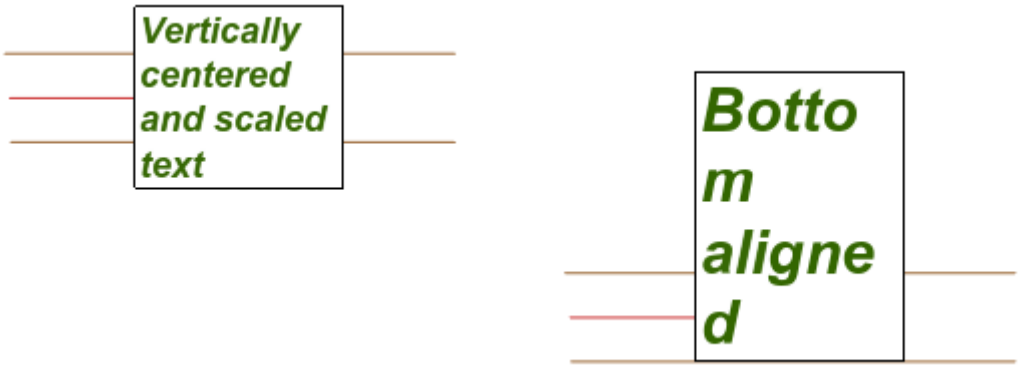
Scaleform version: 2.0.43

Controls automatic vertical sizing and alignment of text fields. Acceptable values are `none` (default), `top`, `bottom`, and `center`. If `verticalAutoSize` is set to `none` (the default) no resizing occurs.

If `verticalAutoSize` is set to `top`, the text field behaves similarly to the regular `TextField.autoSize` as if `wordWrap` is set to `true` (so, only the bottom of the text field is resized and the right side remains fixed).

If `verticalAutoSize` is set to `center`, the text field will be resized and any resizing of the text field is equally distributed to both the top and bottom margins. The anchor point is located in the middle point of the original text field's height (see picture below).

If `verticalAutoSize` is set to `bottom`, the text field will be resized to the top margin. The anchor point is located at the bottom of the original text field's box (see the picture below).



The default value is `none`.

See also:
`verticalAlign`
`TextField.autoSize`

7.4 HTML Extensions

This section describes extensions for HTML in Scaleform.

<FONT> tag, ALPHA attribute

ALPHA="#xx"

Controls alpha transparency for the text, in the range from 00 to FF (0 to 255). 0 = complete transparency, 255 = complete opaqueness. In combination with COLOR attribute, the ALPHA attribute helps to specify semi-transparent text in HTML.

<IMG> tag

At the moment, <IMG> tag in Scaleform may refer only to images imported into the library with assigned linkage identifier. Scaleform does not support loading images from external sources such as Web or file ("http://", "file:///" and other protocols are not supported). See section 6.6: Image Substitutions for how to import images and assign the linkage identifier.
Here is an example of HTML using the <IMG> tag to refer to the "myImage" imported image:

```
t.htmlText = "<p align='right'>abra<img src='myImage' width='20' height='30'
               align='baseline' vspace='-10'>bed232</p>";
```

Currently supported attributes for IMG tag:

`src` - Specifies the linkage identifier for an image symbol in the library. Only images are supported now. This attribute is required; all other attributes are optional. External files (JPEG, GIF, PNG, and SWF files) are not supported yet. User protocol “img://” can be used here instead of embedded image linkage id; the user defined virtual method GFx::ImageCreator:: LoadImage will be called in this case.

`width` - The width of the image being inserted, in pixels.

`height` - The height of the image being inserted, in pixels.

`align` - Specifies the horizontal alignment of the embedded image within the text field. The only supported value for now is `baseline`.

`vspace` - For `baseline` alignment it specifies the offset, in pixels, of the image relatively to baseline. A positive value raises the image above the baseline, and negative lowers the image.

Unsupported attributes:

> `id` - Specifies the name for the movie clip instance (created by Flash Player) that contains the embedded image file, SWF file, or movie clip.
>
> `align` - "left" and "right". Specifies the horizontal alignment of the embedded image within the text field.
>
> `hspace` - For "left" and "right" it specifies the amount of horizontal space that surrounds the image where no text appears.

Note, if the image is imported from another SWF file then this image should be used explicitly somewhere in the Flash file to be usable in an IMG tag. If an image is imported but never used then Flash drops out the reference on it completely from the resulting SWF file. One of the ways to prevent this is to create a movie clip, and drag and drop all imported images you need for IMG tags on it. Don't forget to export this movie clip, because if it is also not used explicitly and not exported, then Flash will drop it out from the resulting SWF file.

7.5 Shadow Effect Control

This section describes extensions that control different shadow effects (e.g., blur, knockout).

blurX, blurY properties

```
blurX:Number  [read-write]
blurY:Number  [read-write]
```

Scaleform version: 2.0.41

Gets or sets the blur radii of the text. Valid values are from 0 to 15.9 (floating point).

The default value is 0.

See also:

`blurStrength`

blurStrength property

```
blurStrength:Number  [read-write]
```

Scaleform version: 2.0.41

Gets or sets the blur strength of the text. Valid values are from 0 to 15.9 (floating point).

The default value is 0.

See also:
- `blurX`
- `blurY`

shadowAlpha property

```
shadowAlpha:Number  [read-write]
```

Scaleform version: 2.0.41

Controls the alpha transparency value for the shadow color. Valid values are 0.00 to 1.00 (e.g., .25 sets a transparency value of 25%).

See also:
- `shadowColor`

shadowAngle property

```
shadowAngle:Number  [read-write]
```

Scaleform version: 2.0.41

Controls the angle of the shadow, similar to `DropShadowFilter`. Valid values are 0 to 360˚ (floating point). The default value is 45.

See also:
- `DropShadowFilter`

shadowBlurX, shadowBlurY properties

```
shadowBlurX:Number  [read-write]
shadowBlurY:Number  [read-write]
```

Scaleform version: 2.0.41

Controls the blur radii of the shadow. Valid values are from 0 to 15.9 (floating point).

The default value is 0.

See also:

`shadowStrength`

shadowColor property

```
shadowColor:Number  [read-write]
```

Scaleform version: 1.1.31

Specifies the color of shadow (see `shadowStyle` for details). The color is specified in the following format: 0xRRGGBB, in which RR is the hexadecimal representation of the red component [0..255], BB the hexadecimal representation of the blue component [0..255] and GG the hexadecimal representation of the green component [0..255].

See also:

`shadowStyle`

shadowDistance property

```
shadowDistance:Number  [read-write]
```

Scaleform version: 2.0.41

Controls the offset distance for the shadow, in pixels, similar to `DropShadowFilter`.

See also:

`DropShadowFilter`

shadowHideObject property

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

Scaleform version: 2.0.41

Indicates whether or not the text is hidden. The value `true` indicates that the text itself is not drawn; only the shadow is visible. The default is `false` (show the text).

shadowKnockOut property

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

Scaleform version: 2.0.41

Controls a knockout effect (`true`), which effectively makes the object's fill transparent and reveals the background color of the document. The default is `false` (no knockout).

shadowQuality property

```
shadowQuality:Number  [read-write]
```

Scaleform version: 2.0.41

Controls the quality of the shadow or glow filter. Unlike Flash, the only values that make sense are 1 or 2. Value 1 means low quality, 2 or above means high quality.

shadowStrength property

```
shadowStrength:Number  [read-write]
```

Scaleform version: 2.0.41

Controls the blur strength of the shadow. Valid values are from 0 to 15.9 (floating point).

The default value is 0.

See also:

```
shadowBlurX
shadowBlurY
```

shadowStyle property

```
shadowStyle:String  [read-write]
```

Scaleform version: 1.1.31

Combined with `shadowColor`, controls shadow rendering for the text field. Specifically, the `shadowStyle` format string describes the set of text layers that will be applied, together with their coordinates. For example:

```
field.shadowStyle = "s{0,-1.5}{-1.4,1.2}{1.4,1.2}t{0,0}";
field.shadowColor = 0xff0000;
```

Within the string the "s" character delimits the shadow layer that will be drawn with `shadowColor` color value, while the "t" character delimits the front text layer drawn with the standard `TextField.textColor` color value. These delimiters are followed by the coordinate pairs describing offsets at which each text layer will be rendered; each coordinate unit maps to one pixel when identity transform is applied to the text. If the "t" delimiter and its coordinate pair are omitted from the style string, text will be drawn identically to when "t{0,0}" is specified.

Please note that in the current implementation using shadow layers will generate extra triangles and draw primitive calls, as text is rendered multiple times. It is strongly recommended to measure the resulting performance and to limit shadow use when possible.

See also:

```
shadowColor
TextField.textColor
```

7.6 Image Substitutions

This section provides description for extensions that control image substitutions.

setImageSubstitutions () method

```
public setImageSubstitutions(substInfoArr:Array) : void
public setImageSubstitutions(substInfo:Object) : void
public setImageSubstitutions(null) : void
```

Scaleform version: 2.0.38

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 = BitmapData.loadBitmap("myImage");
```

NOTE: Do NOT forget the import statement (`import flash.display.BitmapData;` or to use fully a qualified name - `flash.display.BitmapData`), otherwise the result will be "undefined"!

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.

The `substInfoArr` should be the array of such objects, as well as the `substInfo` should be the instance of the object. The version `setImageSubstitutions(substInfo:Object)` can set only a single substitutions whereas the version `setImageSubstitutions(substInfoArr:Array)` sets multiple ones.

Note, every call to `setImageSubstitutions` adds substitutions to the internal list. To clear all of them call the `setImageSubstitutions(null)`.

It is not necessary to keep a reference to the array with substitutions or 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

`substInfoArr:Array` –	An array of substitution descriptor objects (see above).
`substInfo:Object` –	A single substitution descriptor object (see above).
`null` –	Clear all substitutions.

See also:
`updateImageSubstitution()`

Example:

```
var b1 = BitmapData.loadBitmap("smile1");
var b2 = BitmapData.loadBitmap("smile2");
var b3 = BitmapData.loadBitmap("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 };
t.setImageSubstitutions(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 () method

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

Scaleform version: 2.0.38

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

Parameters

`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:

```
t.updateImageSubstitution("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.loadBitmap("smile1a");
var b2a = BitmapData.loadBitmap("smile2a");

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

7.7 IME Support

This section describes extensions for input method editor (IME) support.

disableIME property

`disableIME:Boolean  [read-write]`

Scaleform version: 2.1.50

If set to `true`, prevents IME from being activated in this text field.

The default value is `false`.

getIMECompositionStringStyle () method

`public function getIMECompositionStringStyle(categoryName:String): Object`

Scaleform version: 2.1.50

Returns the color settings descriptor object for the specified category of IME color settings. See `setIMECompositionStringStyle` for a detailed description of the `categoryName` and the descriptor object.

Parameters

`categoryName:String` – Category of IME color settings (see `setIMECompositionStringStyle`).

Returns

Object - The descriptor of color settings for the category (see `setIMECompositionStringStyle`).

See also:

`setIMECompositionStringStyle`
`disableIME`

setIMECompositionStringStyle () method

```
public function setIMECompositionStringStyle(categoryName:String,
                                    styleDescriptor:Object): Number
```

Scaleform version: 2.1.50

Sets the style to the appropriate category of IME color settings. The `categoryName` parameter may contain the following values:

`compositionSegment` – sets color settings for the whole composition string;
`clauseSegment` – sets color settings for clause segment;
`convertedSegment` – sets color settings for converted text segment;
`phraseLengthAdj` – sets color settings for phrase length adjustment;
`lowConfSegment` – sets color settings for low-confidence characters.

The `styleDescriptor` is an Object instance with the following optional members set:

`textColor : Number`
The color of text.

`backgroundColor : Number`
The background color.

`underlineColor : Number`
The color of underline.

`underlineStyle : String`
The style of underline. Valid values:

```
single
thick
dotted
ditheredSingle
ditheredThick
```

All colors are specified in the following format: 0xRRGGBB, in which RR is the hexadecimal representation of the red component [0..255], BB is the hexadecimal representation of the blue component [0..255] and GG is the hexadecimal representation of the green component [0..255].

Parameters

`categoryName:String` – Category of IME color settings (see description).
`styleDescriptor:Object` – A color setting descriptor object (see description).

See also:

`getIMECompositionStringStyle`
`disableIME`

8 TextFormat Class Extensions

TextFormat class represents the character formatting information, e.g., color, font size, and underlining.

alpha property

```
alpha:Number  [read-write]
```

Scaleform version: 2.0.41

Controls the alpha transparency value for the text as a percentage value. Valid values are 0 to 100 (%). Standard Flash `TextFormat.color` doesn't allow an alpha transparency to be set, so this extension could be used to make text partially or completely transparent.

See also:

```
TextFormat
HTML <FONT ALPHA='xx'>
```

9 Stage Class Extensions

In addition to supporting the full set of standard `Stage` class properties, Scaleform also introduces extensions that improve tracking of stage dimensions. These extensions include properties `visibleRect`, `safeRect` and `originalRect`, as well as an extra parameter for the `onResize` handler that represents currently visible frame rectangle. See below for details.

Note: in ActionScript 1.0 it is possible to refer to extensions directly, as in `Stage.visibleRect`. ActionScript 2.0 is less tolerant and it will "complain" about such references. To quiet ActionScript 2.0 compiler you should use the alternative access syntax as follows: `Stage["visibleRect"].`.

visibleRect property

```
visibleRect:flash.geom.Rectangle  [read-only]
```

Scaleform version: 2.2.56

This property contains the currently visible rectangle. This rectangle is being changed when you change the aspect ratio, scale mode, alignment and/or scale of Viewport and want to know which area is visible at the moment. This rectangle may have negative topLeft corner coordinates.

See also:

```
safeRect
originalRect
```

safeRect property

```
safeRect:flash.geom.Rectangle  [read-only]
```

Scaleform version: 2.2.56

This property contains the currently set safe rectangle. This rectangle should be set by user using `GFx::Movie::SetSafeRect` method. If it is not set, then it contains the same rectangle as the `visibleRect` property.

See also:

```
visibleRect
originalRect
```

originalRect property

```
originalRect:flash.geom.Rectangle  [read-only]
```

Scaleform version: 2.2.56

This property always contains the original SWF's rectangle with the original SWF's dimensions. So, if dimensions in "Document properties" in Flash Studio were set to 600px by 400px this rectangle will contain { 0, 0, {600, 400} } values.

See also:

`visibleRect`
`safeRect`

onResize event handler

```
onResize = function([visibleRect:flash.geom.Rectangle]) {}
```

Scaleform version: 2.2.56

The onResize event handler has an extra parameter as an extension. This parameter represents a currently visible rectangle, the same as returned by the `Stage.visibleRect` extension property.

See also:

`visibleRect`

translateToScreen () method

```
public function translateToScreen(pt:Object): Point
```

Scaleform version: 3.3.84

This method returns a point in the Stage coordinate system mapped to the screen coordinate system.

Parameters

`pt:Object` – An Object with x and y members representing the coordinates of the point to be transformed. The flash.geom.Point can be used instead of a generic Object.

Returns

Point – The input point mapped to the screen coordinate system.

10 Array Class Extensions

Locale-specific sorting

Flash's versions of `Array.sort` and `Array.sortOn` methods perform sorting according to Unicode values. However, this is not always in correct order, especially for languages such as French, Spanish, German, etc. In the case of Unicode sorting, all strings containing diacritic marks (e.g., umlaut, accent, acute) appear after the letter 'z'. Thus, on sorting the array ['á', 'z', 'a'] Flash returns ['a', 'z', 'á'] regardless to currently set locale. However, for French the result of locale-specific sorting should be ['a', 'á', 'z'].

To address this issue, Scaleform introduces one more option for `Array.sort`/`sortOn` methods: `Array.LOCALE`:

```
var arr = ['á', 'z', 'a'];
var newArr1 = arr.sort(); // returns ['a', 'z', 'á']
var newArrLoc = arr.sort(Array.LOCALE); // returns ['a', 'á', 'z']
var newArrRev = arr.sort(Array.LOCALE | Array.DESCENDING); // ['z', 'á', 'a']
```

Case insensitive locale-specific sorting is also supported:

```
var arr = ['Á', 'z', 'a'];
var a = arr.sort(Array.LOCALE | Array.CASEINSENSITIVE); // returns ['a', 'Á', 'z']
```

Note, this functionality might not work correctly on certain platforms, which do not support appropriate functionality in kernel. In this case `Array.LOCALE` sorting will perform as the regular one with Unicode values.

Scaleform version: 3.0.65

See also:

`String.localeCompare`

11 String Class Extensions

localeCompare() method

This function compares the sort order of two or more strings and returns the result of the comparison as a numeric value. This method compares in a locale-specific way. If the strings are equivalent, the return value is 0. If the original string value precedes the string value specified by the parameter, the return value is a negative value. If the original string value comes after one in parameter, the return value is a positive value.
See “Array Class Extensions” for more details about locale-specific string operations.
Note, this functionality might not work correctly on certain platforms, which do not support appropriate functionality in kernel. In this case this function will perform as a regular string comparison with Unicode values.

```
public localeCompare(other:String [, caseInsensitive:Boolean]) : Number
```

Scaleform version: 3.0.65

Parameters

`other:String` –	A string to compare with.
`caseInsensitive:Boolean` –	An optional parameter that might be used to perform case-insensitive comparison. If the parameter is not specified then the comparison is case-sensitive.

See also:
`Array.sort`

12 Video Class Extensions

In standard Flash only one Video object can receive data from a NetStream object at a time. Scaleform removes this limitation and allows for the multiple Video objects to be attached to the same NetStream object and receive the same video data.

clipRect property

```
clipRect: flash.geom.Rectangle  [read-write]
```

Scaleform version: 3.0.70

This property allows a video object to display only a part (region) of the original video frame.

Example:

```
video.clipRect = new flash.geom.Rectangle(10, 20, 100, 100);
```

13 3Di Extensions

Scaleform 3Di is implemented using a basic set of 3D AS2 extensions built into Scaleform. The following ActionScript extensions were added for 3Di.

Scaleform version: 3.2.82

_z

Sets the Z coordinate (depth) of the object (Movie clips, Text fields, Buttons), defaults to 0.

_zscale

Sets the scale of the object in the Z direction as a percentage, defaults to 100.

_xrotation

Sets the rotation of the object around the X (horizontal) axis, defaults to 0.

_yrotation

Sets the rotation of the object around the Y (vertical) axis, defaults to 0.

_matrix3d

Sets the complete 3D transform of the object using an array of 16 floats (4 x 4 matrix). If this value is set to NULL, all 3D transformations will be removed and the object will become 2D.

_perspfov

Sets the perspective Field Of View angle on the object, valid angles must be > 0 and < 180. If not set, the object inherits the root FOV angle, which defaults to 55.

The use of these new extensions requires the global variable, `gfxExtensions`, to be set to true in ActionScript.

14 Global Extensions

noInvisibleAdvance property

```
noInvisibleAdvance : Boolean
```

Scaleform version: 2.1.52

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.

See also:

`MovieClip.noAdvance`

imecommand function

```
imecommand(command:String, parameters:String) : Void
```

Scaleform version: 2.1.50

This function is similar to fscommand but is only used to communicate with Scaleform IME implementation. It is used internally by Scaleform.

getIMECandidateListStyle () function

```
public function getIMECompositionStringStyle(categoryName:String): Object
```

Scaleform version: 2.1.50

Returns the color settings descriptor object for an IME candidate list. See `setIMECandidateListStyle` for a detailed description of the descriptor object.

Returns

Object - The descriptor of color settings for the candidate list.

See also:

```
setIMECandidateListStyle
TextField.disableIME
```

setIMECandidateListStyle () function

```
public function setIMECandidateListStyle(styleDescriptor:Object)
```

Scaleform version: 2.1.50

Sets the candidate list style to that of the object passed as argument.

The `styleDescriptor` is an Object instance with the following optional members set:

```
styleObject.textColor                    = 0x5EADFF;
styleObject.selectedTextColor            = 0xFFFFFF;
styleObject.fontSize                     = 20;
styleObject.backgroundColor              = 0x001430;
styleObject.selectedTextBackgroundColor  = 0x2C5A99;
styleObject.indexBackgroundColor         = 0x12325A;
styleObject.selectedIndexBackgroundColor = 0x2C5A99;
styleObject.readingWindowTextColor       = 0xFFFFFF;
styleObject.readingWindowBackgroundColor = 0x001430;
styleObject.readingWindowFontSize        = 20;
```

`textColor : Number`
: The color of text.

`selectedTextColor : Number`
: The color of selected text.

`fontSize : Number`
: font size of the text in the row and row index.

`backgroundColor : Number`
: The background color of the text portion of unselected rows.

`seletedTextBackgroundColor : Number`
: The color of the text portion of the selected row.

`indexBackgroundColor : Number`
: The background color of the index portion of unselected rows.

`selectedIndexBackgroundColor : Number`
: The background color of the index portion of selected rows.

`readingWindowTextColor : Number`
: The color of the text in the reading window.

`readingWindowBackgroundColor : Number`
: Background color of the reading window.

`readingWindowFontSize : Number`
: Font size of the text in the reading window.

All colors are specified in the following format: 0xRRGGBB, in which RR is the hexadecimal representation of the red component [0..255], BB is the hexadecimal representation of the blue component [0..255] and GG is the hexadecimal representation of the green component [0..255].

See also:

`getIMECandidateListStyle`
`TextField.disableIME`

15 Standard Methods and Event Handlers Extensions

Key Class

```
Key.getCode(controllerIdx:Number)
Key.getAscii(controllerIdx:Number)
Key.isDown(controllerIdx:Number)
Key.isToggled(controllerIdx:Number)
```

The Key class methods take an optional parameter for the keyboard/controller. If not specified, then `controllerIdx` will default to 0.

```
Key.onKeyDown(controllerIdx:Number)
```

The Key listener method `onKeyDown` can receive an extra parameter for the keyboard/controller that generated the event.

Selection Class

```
Selection.setFocus(ch:Object, controllerIdx:Number)
Selection.getFocus(controllerIdx:Number)
Selection.getBeginIndex(controllerIdx:Number)
Selection.getEndIndex(controllerIdx:Number)
Selection.setSelection(start:Number, end:Number, controllerIdx:Number)
Selection.getCaretIndex(controllerIdx:Number)
```

The Selection class methods take an optional parameter for the keyboard/controller. If not specified, then `controllerIdx` will default to 0.

```
Selection.onSetFocus(old:Object, new:Object, controllerIdx:Number)
```

The Selection listener method `onSetFocus` can receive an extra parameter for the keyboard/controller that generated the event.

MovieClip/Button/TextField

```
MovieClip.onSetFocus(old:Object, controllerIdx:Number)
MovieClip.onKillFocus(new:Object, controllerIdx:Number)
Button.onSetFocus(old:Object, controllerIdx:Number)
```

```
Button.onKillFocus(new:Object, controllerIdx:Number)
TextField.onSetFocus(old:Object, controllerIdx:Number)
TextField.onKillFocus(new:Object, controllerIdx:Number)
TextField.onChanged(controllerIdx:Number)
```

These MovieClip/Button/TextField listener methods can receive an extra parameter for the keyboard/controller that generated the event.

System.capabilities

`System.capabilities.numControllers` – Returns the number of controllers detected in the system.