Autodesk®
Scaleform®

# Scaleform 4.3 AS3 擴展參考

本文介紹 Scaleform 4.3 中可以使用的 ActionScript 3.0 擴展。

作者： Artem Bolgar, Prasad Silva
版本： 1.05
上次編輯： 2013 年 2 月 26 日

Autodesk®
GAMEWARE

# 版权声明

**Autodesk® Scaleform® 4.3**

如何联系 Autodesk Scaleform：

| | |
|---|---|
| 文档 | AS3 扩展参考 |
| 地址 | Autodesk Scaleform Corporation<br>6305 Ivy Lane, Suite 310<br>Greenbelt, MD 20770, USA |
| 网站 | www.scaleform.com |
| 电邮 | info@scaleform.com |
| 电话 | (301) 446-3200 |
| 传真 | (301) 446-3199 |

# 目录

# 1 引言

Autodesk Scaleform® SDK™ 是一種羽量級、高性能的富媒體 Adobe® Flash®
向量圖形引擎,它建立在專門針對遊戲機和 PC 遊戲開發者的潔淨室實現基礎之上。Autodesk Scaleform
將業經證明的視覺化創作工具(如Adobe Creative
Suite®)與領先遊戲開發者所需的最新硬體圖形加速技術結合在一起。

由於 Flash 主要適用于 Web 而非遊戲或應用程式開發,在某些方面(如 IME
輸入的焦點處理、滑鼠支援、文本欄位支援和處理)具有有限的功能性。Autodesk Scaleform
通過將"擴展"(Extension) 添加到 ActionScript 類來改進這些方面的基本功能。為了在 Scaleform
播放機中實現擴展支援,有必要將 scaleform.gfx.Extensions.enabled 屬性設置為 true(真)。

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

或者　導入 `scaleform.gfx.*;`

```
Extensions.enabled = true;
```

還有必要添加一個路徑 GFXSDK/Resources/AS3/CLIK 作為 Flash 中的"源 (Source) 路徑",以使擴展類對
ActionScript 3.0 編譯器可見(在 Flash Studio 中:'Edit'(編輯) -> 'Preferences'(首選項) -> 'ActionScript' ->
'ActionScript 3.0 Settings…'(ActionScript 3.0 設置…) -> 'Source path'(Source
路徑))。在多數情況下,開發者會希望將此語句添加到將用到擴展的 FLA
檔中的第一個幀。如果不設置此屬性,Scaleform 播放機就會忽略對擴展的所有引用,試圖實現最高水準的
Flash 相容性。

開發者應明白,本文所述的擴展功能在標準 Flash 播放機中無效,因為僅在 Scaleform
中實現此擴展功能。與每個擴展關聯的是一個 Scaleform 版本號,用以標識添加了相應擴展的播放機 SDK
的版本號。本文所述擴展在具有以前的版本號的 Scaleform 播放機中無法工作。

儘管 Autodesk 將竭盡全力使擴展 API 在所有不同版本中得到支援並保持一致,我們還是要保留在未來的
Scaleform 版本中更改、重命名或取消擴展 API
的權利。如果進行重大更改,我們會儘量在重點版本中進行,並會儘早發出通知。

# 2 擴展

## enabled 靜態屬性

enabled:Boolean [read-write]

Scaleform版本： 4.0.12

啟用/禁用擴展。

## noInvisibleAdvance 靜態屬性

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

Scaleform 版本： 4.0.12

如果設置為
true,此屬性會關閉所有看不見的電影剪輯的進展情況。這可能會用來提高包含許多隱藏的電影剪輯的 SWF 的性能。注意,Flash 推進看不見的電影剪輯(它仍然執行時間線動畫,調用幀的 ActionScript 代碼,等等)。因此,將此屬性設置為 true 可能會導致 Scaleform 與 Flash 之間出現行為差異。

## getTopMostEntity() 靜態方法

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

Scaleform 版本： 4.0.13

返回一個可在 (x, y) 座標(stage 座標空間)中找到的最上面的 DisplayObject 實例。此方法在設置和未設置按鈕處理常式(例如,CLICK、MOUSE_DOWN、ROLL_OVER 等)的字元之間可能有所不同。此區別對於篩選出沒有處理常式來處理滑鼠事件的字元來說可能會非常有用。

此方法與 MovieClip.hitTest 相反,因為 hitTest 檢查 x 和 y 座標是否在特定物件的內部,而且 getTopMostEntity 返回指定的 x 和 y 座標處的實際物件。因此,Extensions.getTopMostEntity(x, y).hitTest(x, y, true) == true。

**參數**
`testAll :Boolean` – 指明僅查找帶有按鈕處理常式的字元 (false),或者查找任意字元 (true)。如果不指定此參數,則 getTopMostEntity 假設其為 true。

`x :Number, y :Number` –用來查找一個字元的 Stage 座標。

**另請參見：**

`getMouseTopMostEntity`

## getMouseTopMostEntity() 靜態方法

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

Scaleform 版本： 4.0.13

返回可在當前滑鼠游標位置找到的一個最上面的 DisplayObject 實例。此函數類似于 getTopMostEntity,但它不使用顯式座標,而是使用滑鼠座標。

**参数**

`testAll :Boolean` –指明僅查找帶有按鈕處理常式的字元 (false),或者查找任意字元 (true)。如果不指定此參數,則 getTopMostEntity 假設其為 true。

`mouseIndex :Number` –基於零的滑鼠索引。

**另请参见：**

`getTopMostEntity`

## getMouseCursorType() 靜態方法

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

Scaleform 版本： 4.0.13

此靜態方法返回用於指定滑鼠控制器的當前滑鼠游標類型。此方法類似于 flash.ui.Mouse.cursor 屬性,唯一的區別是此方法允許對多個滑鼠更改游標。

要跟蹤並隨意阻止滑鼠游標變化,請使用 MouseCursorEvent.CURSOR_CHANGE 擴展事件。

**参数**

`mouseIndex :Number` – 基于零的鼠标索引（默认情况下为零）。

**返回**

返回游標的類型,参見 flash.ui.MouseCursor 靜態常數,例如:

- `flash.ui.MouseCursor.ARROW : String = “arrow”`

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

**另请参见：**

setMouseCursorType
MouseCursorEvent

## setMouseCursorType() 静態方法

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

Scaleform 版本： 4.0.13

此靜態方法根據參數 cursor 更改滑鼠游標。此方法類似于 flash.ui.Mouse.cursor 屬性,唯一的區別是此方法允許對多個滑鼠更改游標。

要跟蹤並隨意阻止滑鼠游標變化,請使用 MouseCursorEvent.CURSOR_CHANGE 擴展事件。

**参数**

cursor :uint –游標的類型,參見 flash.ui.MouseCursor 靜態常數,例如：

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

mouseIndex :Number – 基于零的鼠标索引（默认情况下为零）。

**另请参见：**

getMouseCursorType
MouseCursorEvent

## numControllers 靜態屬性

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

Scaleform 版本： 4.0.12

返回系統中檢測到的控制器數量。

## setEdgeAAMode() 靜態方法

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

Scaleform 版本： 4.0.12

設置某個具體顯示物件及其子物件的 EdgeAA 模式。接受下列模式值:

- scaleform.gfx.Extensions.EDGEAA_INHERIT = 0;Inherit EdgeAA 模式（来自父对象）。默认情况下为 On。
- scaleform.gfx.Extensions.EDGEAA_ON = 1;Enable EdgeAA 模式，用于目标显示对象及其子对象 – 除非禁用（参见 EDGEAA_DISABLE）
- scaleform.gfx.Extensions.EDGEAA_OFF = 2;Do not use EdgeAA，用于目标显示对象及其子对象。
- scaleform.gfx.Extensions.EDGEAA_DISABLE = 3;Disable EdgeAA，用于目标显示对象及其子对象，替代一个继承来的 EDGEAA_ON 模式值。

**参数**

`dispObj :DisplayObject` –目標顯示物件,應用新的 EdgeAA 模式值。
`mode :uint` – EdgeAA 模式值。

**另请参见：**

getEdgeAAMode

## getEdgeAAMode() 靜態方法

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

Scaleform 版本： 4.0.12

從特定顯示物件檢索 EdgeAA 模式值。

**参数**

`dispObj :DisplayObject` –目標顯示物件,檢索 EdgeAA 模式值。

另请参见：

setEdgeAAMode

## visibleRect 靜態屬性

visibleRect:Rectangle [read]

Scaleform版本: 4.0.14

此屬性包含當前可見的矩形。當您更改長寬比、縮放模式、對齊和/或視口縮放比例並想知道當時哪個區域可見時,就會更改此矩形。此矩形可能有負的左上角座標。

## isScaleform 靜態屬性

isScaleform:Boolean [read]

Scaleform版本: 4.0.14

假如 SWF 在 GFx 內而非其他 Flash 播放機內運行時,返回「真」(true)。

# 3 DisplayObjectEx擴展

## setRendererString 靜態方法

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

Scaleform版本: 4.0.17

本屬性允許在任何MovieClip實例中通過ActionScript發送自定義變數到渲染器。如果本屬性置位元，字串值將作爲用戶資料傳遞給渲染器。

無預設值。

示例:

```
import scaleform.gfx.*;

// m is a MovieClip on stage.

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

## getRendererString() 靜態方法

```
static public function getRendererString(o:DisplayObject) : String
```

Scaleform版本: 4.0.17

返回作為使用者資料被發送到渲染器的字串值。

## setRendererFloat 靜態方法

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

Scaleform 版本: 4.0.17

本屬性允許在任何MovieClip實例中通過ActionScript發送自定義變數到渲染器。如果本屬性置位元，浮點值將作爲用戶資料傳遞給渲染器。

無預設值。

示例：

```
import scaleform.gfx.*;

// m is a MovieClip on stage.

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

## getRendererFloat()静态方法

```
static public function getRendererFloat(o:DisplayObject) : Number
```

Scaleform版本: 4.0.17

返回作為使用者資料被發送到渲染器的浮點值。

## disableBatching 靜態方法

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

Scaleform版本: 4.0.17

此屬性禁用針對自訂繪圖的網格生成批次處理。

示例

```
import scaleform.gfx.*;

// m is a MovieClip on stage.

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

## isBatchingDisabled() 静态方法

```
static public function isBatchingDisabled(o:DisplayObject) : Boolean
```

Scaleform 版本: 4.0.17

檢查是否禁用針對自訂繪製的網格生成批次處理。

# 4 FocusEventEx 擴展

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

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

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

此事件是標準 flash.events.FocusEvent 的一個擴展。它添加一個 'controllerIdx'
成員,該成員指明導致該事件的控制器的一個基於零的索引。當把 Extensions.enabled 屬性設置為 true 時,Scaleform 始終生成 FocusEventEx 事件,而不是標準 FocusEvent。使用者可以檢查收到的事件是不是 FocusEventEx 的一個實例,如果是,就把該事件物件歸到擴展類型中。示例：

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

Extensions.enabled = true;

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

## controllerIdx 屬性

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

Scaleform 版本： 4.0.12

指明哪個鍵盤/控制器用於該事件(基於零的索引)。

# 5 FocusManager 擴展

## alwaysEnableArrowKeys 靜態屬性

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

Scaleform 版本： 4.0.12

此靜態屬性允許方向鍵更改焦點 -- 即使把 _focusrect 屬性設置為
false(捕獲焦點時應用)。預設情況下,如果通過 _focusrect = false 禁用黃色焦點矩形,Flash
就不允許您使用方向鍵更改焦點。要更改此行為,請將 alwaysEnableArrowKeys 屬性設置為 true。

## disableFocusKeys 靜態屬性

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

Scaleform 版本： 4.0.12

此靜態屬性禁用處理所有焦點鍵(TAB、Shift-TAB 和方向鍵),因此,使用者可以實施自己的焦點鍵管理。

## moveFocus() 靜態方法

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

Scaleform 版本： 4.0.12

此靜態方法用來通過對以下焦點鍵之一進行類比按鍵操作來移動焦點矩形:TAB、Shift-
TAB或方向鍵。將此方法與 disableFocusKeys 和 modalClip 屬性一起使用,可實施自訂焦點管理。

**參数**

`keyToSimulate :String` - 要類比的鍵的名稱：“向上”、“向下”、“向左”、“向右”、“tab”、
“shifttab”。

`startFromMovie:InteractiveObject` –可選參數,用來指定一個字元;moveFocus
將它(而非當前聚焦的參數)用作一個起點。此屬性可能是空 (Null)
或未定義,這意味著當前聚焦的字元用作起點;這可能會對指定第三個可選參數有用。

`includeFocusEnabledChars :Boolean` –可選標誌,允許將 moveFocus 放在僅設置 focusEnabled 屬性的字元以及設置了 tabEnabled / tabIndex 屬性的字元上。如果不指定該標誌,或者將該標誌設置為 false,那麼只有設置了 tabEnabled / tabIndex 屬性的字元才能參加焦點移動。
`controllerIdx :uint` –用於該操作的控制器的索引。如果不指定,那就是用預設控制器(控制器 0)。

**返回**
返回下一個要聚焦的字元,或者如果找不到該字元則返回 null。

**另请参见：**

findFocus
disableFocusKeys
setModalClip
getModalClip

## findFocus() 靜態方法

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

Scaleform 版本： 4.0.12

此靜態方法用來通過對以下焦點鍵之一進行類比按鍵操作來查找下一個焦點項:TAB、Shift-TAB或方向鍵。將此方法與 disableFocusKeys 和 setModalClip/getModalClip 擴展相配合,可用來實施自訂焦點管理。

**参数**
`keyToSimulate :String` - 要類比的鍵的名稱：“向上”、“向下”、“向左”、“向右”、“tab”、“shifttab”。
`parentMovie:DisplayObjectContainer` –
電影剪輯,用作模式剪輯。焦點項搜索僅在此剪輯的子項內執行。可以為 null。
`loop :Boolean` - 要迴圈焦點的 Boolean 標誌。例如,如果當前聚焦的專案位於底部,而鍵為“向下”鍵,那麼 findFocus 要麼返回 “null”(如果此標誌為 “false”)或者返回最上面的可聚焦專案(如果此標誌為 “true”)。
`startFromMovie:InteractiveObject` –可選參數,用來指定一個字元;moveFocus 將它(而非當前聚焦的參數)用作一個起點。此屬性可能為 null 或未定義,這意味著當前聚焦的字元用作一個起點。

includeFocusEnabledChars :Boolean –可選標誌,允許將 moveFocus 放在僅設置 focusEnabled 屬性的字元以及設置了 tabEnabled / tabIndex 屬性的字元上。如果不指定該標誌,或者將該標誌設置為 false,那麼只有設置了 tabEnabled / tabIndex 屬性的字元才能參加焦點移動。
controllerIdx :uint –
正在操縱焦點的控制器的一個零基索引。將此與焦點組一起使用,可提供多控制器焦點支援。

**返回**

返回下一個要聚焦的字元,或者如果找不到該字元則返回 null。

**另请参见：**

moveFocus
disableFocusKeys
setModalClip
getModalClip

## setFocus() 静态方法

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

Scaleform 版本： 4.0.12

此靜態方法與指定 stage.focus 屬性的操作相同。唯一不同的是,可以指定應與此操作關聯的控制器的索引。

**参数**

obj:InteractiveObject - 新聚焦的互動式物件
controllerIdx :uint - 表明哪個鍵盤/控制器用於該事件(基於零的索引)。

## getFocus() 靜態方法

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

Scaleform 版本： 4.0.12

此靜態方法返回與 stage.focus 屬性相同的值。唯一不同的是,可以指定應與此操作關聯的控制器的索引。

**参数**

controllerIdx :uint - 指明哪個鍵盤/控制器用於該事件(基於零的索引)。

**返回**

當前聚焦的互動式物件。

## numFocusGroups 靜態屬性

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

Scaleform 版本： 4.0.12

返回焦點組數量,通過調用 setControllerFocusGroup 函數進行設置。如果焦點組 0 和 3 處於活動狀態,numFocusGroups 將返回 2。

## setFocusGroupMask() 靜態方法

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

Scaleform 版本： 4.0.12

此方法將一個位元遮罩設置為一個字元以及全部其子項。此位元遮罩將焦點組所有權指定給該字元,意思是只有該位元遮罩中表示的控制器才能將焦點移動到該字元內。使用 setControllerFocusGroup 擴展方法,可以將焦點組與控制器關聯。

例如,我們假定"按鈕 1"只能通過控制器 0 聚焦,而"電影剪輯2"可通過控制器 0 和 1 聚焦。要實現此行為,請將焦點組與控制器進行關聯：

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

FocusManager.setFocusGroupMask(button1, 0x1); // 位 0 – 焦点组 0
FocusManager.setFocusGroupMask(movieclip2,0x1 | 0x2); // 位 0 和 1 – 焦点
// 组 0 和组 1
```

"focusGroupMask" 位元遮罩可設置為母電影剪輯。這將會把此遮罩值傳播到所有其子項。

**参数**

`obj:InteractiveObject` - 一個互動式物件
`mask:uint` - 一個焦點組位元遮罩

**另请参见：**

setControllerFocusGroup
getFocusGroupMask

## getFocusGroupMask() 靜態方法

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

Scaleform 版本： 4.0.12

返回當前焦點組位元遮罩值(有關詳細資訊,請參見 `setFocusGroupMask`)。

**参数**

`obj:InteractiveObject` - 一個互動式物件

**返回**

一個用於指定的互動式物件的焦點組位元遮罩。

**另请参见：**

setControllerFocusGroup
setFocusGroupMask

## setControllerFocusGroup() 靜態方法

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

Scaleform 版本： 4.0.12

此靜態方法將 controllerIndex 表示的控制器與一個焦點組相關聯。預設情況下,所有控制器均與焦點組 0 關聯,這意味著它們正在使用同一焦點。不過,可以使每個控制器與其自己的焦點一起工作。例如,如果兩個控制器應有單獨的焦點(在分屏使用情況下),則 setControllerFocusGroup(1,1) 將為其控制器 1 創建一個單獨的焦點組。調用 setControllerFocusGroup(1,0) 將使控制器 0 和 1 再次共用同一焦點。

**参数**

`controllerIdx:uint` - 控制器的零基索引。
`focusGroupIdx:uint` –焦點組的零基索引。

**返回**

如果成功,就返回 true。

## getControllerFocusGroup() 静态方法

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

Scaleform 版本： 4.0.12

此方法返回與指定控制器關聯的焦點組索引。

**參数**

`controllerIndex` – 物理控制器的零基索引。

**返回**

焦點組的基於零的索引。

## setModalClip() 靜態方法

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

Scaleform 版本： 4.0.12

此靜態方法將指定的電影剪輯設置為用於焦點管理的"模式"(modal) 剪輯。這意味著 TAB、Shift-TAB 以及方向鍵將僅在所有 "tabable" 子項之間的指定電影剪輯內移動焦點。

**參数**

`controllerIdx:uint` - 控制器的一個零基索引。

`mc:Sprite` - 一個模式剪輯。

## getModalClip() 靜態方法

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

Scaleform 版本： 4.0.12

此靜態方法返回用於指定控制器的模式剪輯。

**參数**

`controllerIdx` – 控制器的零基索引。

**返回**

一个模式剪輯，如果未找到，則表明未定义。

# 6 GamePad 扩展

**控制項常數**

此類為一般遊戲手柄控制項(如觸發器、類比搖杆和按鈕)提供輔助常數。他們是 SF_KeyCodes.h 中定義的常數的鏡面反射。Scaleform 在運行時插入正確的值,因此,編譯利用這些常數的 SWF 的操作就會按預期進行。

為了方便起見,Scaleform FxPlayer 框架將遊戲手柄控制項映射到鍵盤當量,然而,自訂的集成或應用程式可將這些常數與 GFx::Movie::HandleEvent 一起使用,如果對於 AS3 鍵盤事件有必要的話,在控制器與鍵盤之間產生某種區別。

Scaleform FxPlayer 框架確實將這些常數與 GamePadAnalogEvents 一起使用,為這樣的類比值提供適當回饋。不過,也可以想到的是,開發者也可以不使用 GamePad 常數,而是使用鍵盤代碼。是否選擇將遊戲手柄事件與鍵盤事件結合在一起,留給開發者自行做出判斷。

## supportsAnalogEvents() 靜態方法

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

Scaleform 版本： 4.0.13

假如基本硬體平臺的 Scaleform 實現支援遊戲手柄類比事件(如對於觸發器和拇指棒),就返回 true。此值可用來確定是否支援 GamePadAnalogEvents。

**另请参见：**

GamePadAnalogEvent

## 7 GamePadAnalogEvent 擴展

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

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

        public var code :uint          = 0;    // 参见 scaleform.gfx.GamePad，了解
// 有效手柄代码
        public var controllerIdx : uint = 0;
        public var xvalue :Number      = 0;    // 标准化 [-1, 1]
public var yvalue :Number      = 0;    // 标准化 [-1, 1]

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

如果 Scaleform 支援針對某個特定平臺的遊戲手柄類比事件,就觸發此事件。此事件只從 Stage 發出,而所有偵聽者都會附加到 Stage 物件。Scaleform FxPlayer 框架通過用於 'code' 屬性的適當 GamePad 常數觸發這些事件,不過,如有必要,開發者可以在自己的 Scaleform 集成中使用其它值(如鍵盤值)。示例:

```
import scaleform.gfx.*;

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

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

```
}
stage.addEventListener(GamePadAnalogEvent.CHANGE, ev);
```

## code 屬性

```
code : uint
```

Scaleform 版本： 4.0.13

指明使用哪個鍵/控制項生成此事件。Scaleform FxPlayer 框架將使用遊戲手柄常數。參見GamePad。

## controllerIdx 屬性

```
controllerIdx : uint
```

Scaleform 版本： 4.0.13

指明哪個鍵盤/控制器用於該事件(基於零的索引)。

## xvalue 屬性

```
xvalue : Number
```

Scaleform 版本： 4.0.13

指明 x 軸中的當前值。該值將在 -1 與 1 之間標準化,其中包含 -1 和 1。

## yvalue 屬性

```
yvalue : Number
```

Scaleform 版本： 4.0.13

指明 y 軸中的當前值。該值將在 -1 與 1 之間標準化,其中包含 -1 和 1。

# 8 InteractiveObjectEx 擴展

## getHitTestDisable() 静态方法

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

Scaleform 版本： 4.0.13

返回 'hitTestDisable' 標誌的狀態。當設置為 true 時,MovieClip.hitTest
函數在命中測試檢測中忽略此互動式物件。此外,所有其它滑鼠事件都傳播到該物件。

預設值為 false。

**参数**

o - An interactive object.

**返回**

一個代表 'hitTestDisable' 標誌的狀態的 Boolean 值。

**另请参见：**

InteractiveObjectEx.setHitTestDisable

## setHitTestDisable() 靜態方法

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

Scaleform 版本： 4.0.13

設置 'hitTestDisable' 標誌的狀態。當它設置為 true 時,MovieClip.hitTest
函數在命中測試檢測中忽略此互動式物件。此外,所有其它滑鼠事件都傳播到該物件。預設值為 false。

**参数**

o - 一个交互式对象。

f -一個代表 'hitTestDisable' 標誌的新狀態的 Boolean 值。

**另请参见：**

InteractiveObjectEx.getHitTestDisable

## getTopmostLevel() 静态方法

public function getTopmostLevel (o:InteractiveObject) : Boolean

Scaleform 版本： 4.0.13

返回 ‘topmostLevel’ 標誌的狀態。設置為 true 時,此字元顯示在所有其它字元的上面,而不管其深度如何。

**参数**

- o - An interactive object.

**返回**

一個代表 ‘topmostLevel’ 標誌的狀態的 Boolean 值。

另请参见：

InteractiveObjectEx.setTopmostLevel

## setTopmostLevel () 靜態方法

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

Scaleform 版本： 4.0.13

設置 ‘topmostLevel’ 標誌的狀態。設置為 true 時,此字元顯示在所有其它字元的上面,而不管其深度如何。當把游標從各個級別拉到物件上面時,這可能對於實現自訂滑鼠游標有用。預設值為 false。 如果把多個字元標為 "topmostLevel",則拖曳順序與沒有將字元標為最上面時相同,也就是說,如果物件A 被拖到物件B 下麵,則在把上述字元標為最上面時,物件A 仍將會在物件B 下麵,而不管把 "topmostLevel" 屬性設置為 true 的順序如何。 注意:一旦某個字元標為 "topmostLevel",swapDepth ActionScript 函數就不會對此字元產生任何影響。 預設值為 false。

**参数**

- o - 一個互動式物件。
- f -一個代表 ‘topmostLevel’ 標誌的新狀態的 Boolean 值。

另请参见：

InteractiveObjectEx.getTopmostLevel

# 9 KeyboardEventEx 扩展

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

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

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

此事件是標準 flash.events.KeyboardEvent 的一個擴展。它添加一個 'controllerIdx' 成員,該成員指明導致該事件的控制器的一個基於零的索引。當把 Extensions.enabled 屬性設置為 true 時,Scaleform 始終生成 KeyboardEventEx 事件,而不是標準 KeyboardEvent。使用者可以檢查收到的事件是不是 KeyboardEventEx 的一個實例,如果是,就把該事件物件歸到擴展類型中。示例：

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

Extensions.enabled = true;

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

## controllerIdx 屬性

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

Scaleform 版本： 4.0.12

指明哪個鍵盤/控制器用於該事件(基於零的索引)。

## 10 MouseCursorEvent 扩展

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

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

        static public const CURSOR_CHANGE : String = "mouseCursorChange";

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

此事件用來跟蹤和/或防止滑鼠游標變化。它設置有以下事件的屬性: bubbles – false,它不起泡 (bubble) cancellable – true,可通過調用 preventDefault() 方法來防止預設操作(游標變化)。
此事件對於實施自訂的動畫滑鼠游標非常有用。只要 Scaleform
更改滑鼠游標的形狀(通過翻轉文本欄位或按鈕,或者通過設置 flash.ui.Mouse.cursor 屬性),都會對一個 stage 觸發此事件(假如將 Extensions.enabled 設置為 true 的話)。示例:

```
Extensions.enabled = true;

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

注意:只有在一個 Stage 上設置了偵聽器,才觸發此事件。

### cursor 属性

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

Scaleform 版本： 4.0.13

注意:只有在一個 Stage 上設置了偵聽器,才觸發此事件。：

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

## mouseIdx 属性

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

Scaleform 版本： 4.0.13

指明為哪個滑鼠/控制器生成該事件(基於零的索引)。

## 11 MouseEventEx 扩展

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

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

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

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

此事件是標準 flash.events.MouseEvent 的一個擴展。它為支援多控制器和滑鼠右/中鍵添加屬性。當把 Extensions.enabled 屬性設置為 true 時,Scaleform 始終生成 MouseEventEx 事件,而不是標準 MouseEvent。使用者可以檢查收到的事件是不是 MouseEventEx 的一個實例,如果是,就把該事件物件歸到擴展類型中。示例:

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

stage.doubleClickEnabled = true;

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

```
      trace(e);
}
stage.addEventListener(MouseEvent.MOUSE_DOWN, ev);
stage.addEventListener(MouseEvent.MOUSE_UP, ev);
stage.addEventListener(MouseEvent.CLICK, ev);
stage.addEventListener(MouseEvent.DOUBLE_CLICK, ev);
```

注意:一旦啟用擴展,就會為滑鼠右、中、中心等鍵觸發滑鼠事件,而且使用者必須檢查 buttonIdx 屬性,搞清楚哪個按鈕生成該事件。

## buttonIdx 屬性

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

Scaleform 版本： 4.0.13

指明為哪個按鈕生成該事件(基於零的索引)。此值可以為下列值之一：

- MouseEventEx.LEFT_BUTTON :uint = 0 - 滑鼠左鍵
- MouseEventEx.RIGHT_BUTTON :uint = 1 - 滑鼠右鍵
- MouseEventEx.MIDDLE_BUTTON :uint = 2 - 滑鼠右鍵
- 假如滑鼠有 3 個按鍵,大於 2 的任何值都是合法的。

## mouseIdx 屬性

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

Scaleform 版本： 4.0.13

指明為哪個滑鼠/控制器生成該事件(基於零的索引)。

## nestingIdx 屬性

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

Scaleform 版本： 4.0.13

此屬性對於 rollOver/Out、mouseOver/Out 和 dragOver/Out
事件來說是可選的。此參數指定同一字元上的嵌套的翻轉/拖出 (rollover/dragover) 事件的索引。

當生成嵌套的 rollOver/rollOut、mouseOver/Out 和 dragOver/dragOut 事件(單獨針對每個滑鼠游標),此參數代表嵌套的基於零的索引:初始事件將會把 0 作為參數;如果第二個游標翻轉同一個字元,則把該成員設置為 1,就會觸發第二個 rollOver/rollOut 事件。如果有任何游標離開該字元,該成員設置為 1,就會觸發 rollOut/mouseOut/dragOut;把該成員設置為 0,就會觸發最後一個 rollOut/mouseOut/dragOut。

## 12 TextEventEx 扩展

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

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

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

此事件是標準 flash.events.TextEvent 的一個擴展。它添加一個 'controllerIdx' 成員,該成員表明導致該事件的控制器的一個基於零的索引。當把 Extensions.enabled 屬性設置為 true 時,Scaleform 始終生成 TextEventEx 事件,而不是標準 TextEvent。使用者可以檢查收到的事件是不是 TextEventEx 的一個實例,如果是,就把該事件物件歸到擴展類型中。示例:

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

Extensions.enabled = true;

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

### controllerIdx 属性

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

Scaleform 版本： 4.0.12

指明哪個鍵盤/控制器用於該事件(基於零的索引)。

## LINK_MOUSE_OVER/LINK_MOUSE_OUT 事件

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

Scaleform 版本： 4.0.14

開發者現在能夠使用 TextFieldEx.LINK_MOUSE_OVER 和 TextFieldEx.LINK_MOUSE_OUT 事件擴展偵聽 TextField 連結(由 HTML 標記指定)上的滑鼠 over/out 事件。這些事件的行為與其它 AS3 事件相同,而且可以使用 addEventListener 範例進行偵聽。

## buttonIdx 屬性

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

Scaleform版本: 4.0.14

表明使用哪個滑鼠按鍵啟動 TextEventEx(MouseEventEx.LEFT_BUTTON、MouseEventEx.RIGHT_BUTTON 等;參閱 MouseEventEx.buttonIdx)。

# 13 System 扩展

## actionVerbose 静态属性

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

Scaleform 版本： 4.0.12

啟用/禁用操作碼跟蹤。

## getStackTrace() 静态方法

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

Scaleform 版本： 4.0.12

獲取格式化為字串的當前堆疊跟蹤。

## getCodeFileName() 静态方法

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

Scaleform 版本： 4.0.12

獲取當前執行的代碼的檔案名。

# 14 TextFieldEx 扩展

## appendHtml() 静态方法

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

Scaleform 版本： 4.0.12

將 newHtml 參數指定的 HTML 附加到文本欄位的文本末尾。此方法比一個 htmlText 屬性上的添加指定(+=)(如 txt.htmlText += moreHtml)更加有效。htmlText 屬性上的常規 += 生成 HTML 字串,附加新的 HTML 部分,然後從頭解析整個 HTML。此函數執行增量 HTML 解析,即它僅解析來自 newHtml 字串參數的 HTML(這就是 newHtml 參數中的 HTML 有理由的原因,這對於 +=
操作符並非必需的)。它對於包含大量內容的文本欄位尤其重要。

注意：如果将一个样式表应用到该文本字段，此方法就无效。

**参数**
`textField:TextField` –將 HTML 附加到的一個文本欄位。
`newHtml:String` –要將 HTML 附加到現有文本的字串。

## setIMEEnabled() 静态方法

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

Scaleform 版本： 4.0.12

啟用/禁用對指定文本欄位的 IME。如果將 isEnabled 設置為 false,則不讓在此文本欄位中啟動 IME。預設情況下,啟用了 IME。
**参数**
`textField :TextField` –要操作的文本欄位。
`isEnabled :Boolean` –如果是 true – 啟用 IME;否則禁用。

## setVerticalAlign() 静态方法

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

Scaleform 版本： 4.0.12

在文字方塊內設置文本垂直對齊。該屬性的有效值為 TextFieldEx 類中聲明的下列常數：

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

如果將此屬性設置為 center,則文本位於文字方塊內中心;如果設置為 bottom,則文本位於文字方塊內底部(参見下圖)：


*垂直居中和缩放文本*　　*底部对齐*

默认值为 top

**参数**

`textField:TextField` - 設置垂直對齊的文本欄位

`valign:String` - 對齊值("top", "center", "bottom")。

## getVerticalAlign()静态方法

```
static public function getVerticalAlign(textField:TextField) : String
```

Scaleform版本: 4.0.14

返回在文本框内文本垂直对齐。

## setImageSubstitutions ()静态方法

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

Scaleform 版本： 2.0.38

设置文本域图像替换链接。
只有在图像插入到SWF时字符串链接替换才有效；这些图像还需要匹配链接以输出名字。插入图像到SWF时需要用到：

1. 导入位图图像到库。
2. 右键点击（Windows系统）或者控制键点击（Macintosh系统）位于库中的图像，从菜单目录中选择链接项。
3. 选择ActionScript输出，在第一帧中输出并在标识文本框中输入其名字（例如，myImage）。
4. 点击OK确定链接符。

在图像导入链接符分配完成后，需要创建一个BitmapData实例。下面为该实例的ActionScript代码示例：

```
import flash.display.BitmapData;
var imageBmp:BitmapData = BitmapData.loadBitmap("myImage");
```

注意：不要遗忘导入声明(导入`import flash.display.BitmapData`;或使用完整名称 - `flash.display.BitmapData`)，不然导致结果“未定义”！

如果需要使用多个图像做替换，需要为每个图像重复这些步骤，给出不同的链接ID。

单个替换对象设置如下：

`subString:String`
定义替换图像的子链接；这个为必要项。最大子链长度为15个字符。

`image : BitmapData`
指定图像；必要项。

`width : Number`
定义屏幕中的图像显示宽度，以像素为单位。可选项。

`height : Number`
定义屏幕中的图像显示高度，以像素为单位。可选项。

baseLineY : Number

定义图像基准线的Y坐标偏移量，以原始图像的像素为单位（w/o转换）。可选项。默认情况下，该值与图像高度一致，因此，图像底部正好位于基准线上。

id : String

指定替换ID作为“updateImageSubstitution ”调用的的第一个参数。可选项。

“substInfoArr”应该为这些对象的序列，如同“substInfo”应该为对象的实例一样。版本setImageSubstitutions(substInfo:Object)只支持单个替换，而版本setImageSubstitutions(substInfoArr:Array)可设置多个。

注意，每个setImageSubstitutions的调用增加内部列表的替换。清楚这些元素需要调用setImageSubstitutions(null)。

setImageSubstitutions函数调用后，在ActionScript中必须保持一个到替换序列或者单个描述对象的索引；无论如何，需要保持这个索引，由于无法将替换序列恢复到文本域中，在ActionScript的其他地方需要引用则需要这个索引。

**参数**

| | |
|---|---|
| substInfoArr:Array – | 描述符对象替换序列（见上）。 |
| substInfo:Object – | 单个描述符对象替换（见上）。 |
| null – | 清除所有替换。 |

可参考：

updateImageSubstitution()

示例：

```
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);
```

只要文本域中包含子链“=）”，当并不应用，该子链将被名为“smile1”图像链接符所替换。

## updateImageSubstitution ()静态方法

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

Scaleform 版本： 2.0.38

替换或删除原先由setImageSubstitutions 函数为文本替换创建的图像。

**参数**

id:String – 替换ID号，与setImageSubstitutions函数调用使用的描述符对象ID成员相同。

image:BitmapData – 指定新图像；若为null则完全删除替换。

可参考：
setImageSubstitutions()

示例：

```
t.updateImageSubstitution("sm=)", b3);
```

以下为插入图像的动画示例。在onEnterFrame句柄或使用setInterval可作更新操作。注意，updateImageSubstitution调用时不会发生文本重新格式化；因此，新图像的大小应该与原先图像保持一致。

```
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;
    }
```

## setTextAutoSize() 静态方法

```
static public function setTextAutoSize(textField:TextField, autoSz:String) : void
```

Scaleform版本: 4.0.14

启用自动调整文本字体大小，以收缩或适应文本字段。该属性的有效值为在 TextFieldEx 类中声明的下列常数：

```
            public static const TEXTAUTOSZ_NONE:String      = "none";
            public static const TEXTAUTOSZ_SHRINK:String    = "shrink";
            public static const TEXTAUTOSZ_FIT:String       = "fit";
```

如果自动调整文本大小值为 `shrink`（收缩）或 `fit`（适应），而且文本不适应某个文本字段，就会按比例缩小该文本的大小，以使整个文本适应文本字段，因而没有必要进行滚动。假如文本大小变得太小（字体大小约 5 号），那么仍然使用默认滚动逻辑，并且不会进一步减小字体大小。

设置textAutoSize为“`fit`”模式时，字体大小增大到填充满整个文本空间位置。设置为“`shrink`”模式时当字体大小超出文本空间范围时将减小字体的大小。


**参数**

`textField:TextField` – 调整文本字体大小的一个文本字段。

`autoSz:String` - 自动文本调整大小值 (`"none"`, `"shrink"`, `"fit"`)。

## getTextAutoSize() 静态方法

```
static public function getTextAutoSize(textField:TextField) : String
```

Scaleform版本: 4.0.14

返回为自动调整文本字体大小而设置的值，以收缩或适应文本字段。有效值为 "none"（无）、"shrink"（收缩）和 "fit"（适应）。

## setNoTranslate()静态方法

```
static public function setNoTranslate(textField:TextField, noTranslate:Boolean) :
                                                                        void
```

Scaleform版本: 4.1.20

设置一个布尔值，以禁用对目标文本字段的自动翻译支持（参阅 GFx::Translator）。假如设置为真 (true)，防止对文本字段调用 `Scaleform::GFx::Translator::Translate` 回调。

**参数**

`textField:TextField` – 调整文本字体大小的一个文本字段。
`noTranslate:Boolean` - 用来禁用/启用自动翻译的一个布尔值。

## gettNoTranslate() 静态方法

```
static public function getNoTranslate(textField:TextField) : Boolean
```

Scaleform版本: 4.1.20

返回用来确定是否对目标文本字段禁用自动翻译支持的布尔值。