Autodesk®
Scaleform®

Scaleform 4.2 AS3 扩展参考

本文介绍 Scaleform 4.2 中可以使用的 ActionScript 3.0 扩展。

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

Autodesk®
GAMEWARE

版权声明

Autodesk® Scaleform® 4.2

Autodesk®

GAMEWARE

如何联系 Autodesk Scaleform：

文档	AS3 扩展参考
地址	Autodesk Scaleform Corporation 6305 Ivy Lane, Suite 310 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™ 是一种轻量级、高性能的富媒体 Adobe® Flash® 矢量图形引擎，它建立在专门针对游戏机和 PC 游戏开发者的洁净室实现基础之上。Autodesk Scaleform 将业经证明的可视化创作工具（如 Adobe Creative Suite®）与领先游戏开发者所需的最新硬件图形加速技术结合在一起。

由于 Flash 主要适用于 Web 而非游戏或应用程序开发，在某些方面（如 IME 输入的焦点处理、鼠标支持、文本字段支持和处理）具有有限的功能性。Autodesk Scaleform 通过将“扩展”(Extension) 添加到 ActionScript 类来改进这些方面的基本功能。为了在 Scaleform 播放器中实现扩展支持，有必要将 scaleform.gfx.Extensions.enabled 属性设置为 `true`（**真**）。

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

或者

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

还有必要添加一个路径 *GFXSDK/Resources/AS3/CLIK* 作为 Flash 中的“源（Source) 路径”，以使扩展类对 ActionScript 3.0 编译器可见（在 Flash Studio 中：*‘Edit’（编辑）* -> *‘Preferences’（首选项）* -> *‘ActionScript’* -> *‘ActionScript 3.0 Settings…’（ActionScript 3.0 设置…）* -> *‘Source path’（Source 路径）*）。在多数情况下，开发者会希望将此语句添加到将用到扩展的 FLA 文件中的第一个帧。如果不设置此属性，Scaleform 播放器就会忽略对扩展的所有引用，试图实现最高水平的 Flash 兼容性。

开发者应明白，本文所述的扩展功能在标准 Flash 播放器中无效，因为仅在 Scaleform 中实现此扩展功能。与每个扩展关联的是一个 Scaleform 版本号，用以标识添加了相应扩展的播放器 SDK 的版本号。本文所述扩展在具有以前的版本号的 Scaleform 播放器中无法工作。

尽管 Autodesk 将竭尽全力使扩展 API 在所有不同版本中得到支持并保持一致，我们还是要保留在未来的 Scaleform 版本中更改、重命名或取消扩展 API 的权利。如果进行重大更改，我们会尽量在重点版本中进行，并会尽早发出通知。

2 扩展

enabled 静态属性

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

Scaleform 版本： 4.0.12

启用/禁用扩展。

noInvisibleAdvance 静态属性

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

Scaleform 版本： 4.0.12

如果设置为 true，此属性会关闭所有看不见的电影剪辑的进展情况。这可能会用来提高包含许多隐藏的电影剪辑的 SWF 的性能。注意，Flash 推进看不见的电影剪辑（它仍然执行时间线动画，调用帧的 ActionScript 代码，等等）。因此，将此属性设置为 true 可能会导致 Scaleform 与 Flash 之间出现行为差异。

getTopMostEntity() 静态方法

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

Scaleform 版本： 4.0.13

返回一个可在 (x, y) 坐标（stage 坐标空间）中找到的最上面的 DisplayObject 实例。此方法在设置和未设置按钮处理程序（例如，CLICK、MOUSE_DOWN、ROLL_OVER 等）的字符之间可能有所不同。此区别对于筛选出没有处理程序来处理鼠标事件的字符来说可能会非常有用。

此方法与 MovieClip.hitTest 相反，因为 hitTest 检查 x 和 y 坐标是否在特定对象的内部，而且 getTopMostEntity 返回指定的 x 和 y 坐标处的实际对象。因此，Extensions.getTopMostEntity(x, y).hitTest(x, y, true) == true。

参数

testAll :Boolean – 指明仅查找带有按钮处理程序的字符 (`false`)，或者查找任意字符 (`true`)。如果不指定此参数，则 getTopMostEntity 假设其为 `true`。

x :Number, y :Number – 用来查找一个字符的 Stage 坐标。

另请参见：

getMouseTopMostEntity

getMouseTopMostEntity() 静态方法

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

Scaleform 版本： 4.0.13

返回可在当前鼠标光标位置找到的一个最上面的 DisplayObject 实例。此函数类似于 getTopMostEntity，但它不使用显式坐标，而是使用鼠标坐标。

参数

testAll :Boolean – 指明仅查找带有按钮处理程序的字符 (`false`)，或者查找任意字符 (`true`)。如果不指定此参数，则 getTopMostEntity 假设其为 `true`。

mouseIndex :Number – 基于零的鼠标索引。

另请参见：

getTopMostEntity

getMouseCursorType() 静态方法

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

Scaleform 版本： 4.0.13

此静态方法返回用于指定鼠标控制器的当前鼠标光标类型。此方法类似于 flash.ui.Mouse.cursor 属性，唯一的区别是此方法允许对多个鼠标更改光标。

要跟踪并随意阻止鼠标光标变化，请使用 MouseCursorEvent.CURSOR_CHANGE 扩展事件。

参数

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

返回

返回光标的类型，参见 flash.ui.MouseCursor 静态常数，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() 静态方法

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

Scaleform 版本： 4.0.13

此静态方法根据参数 cursor 更改鼠标光标。此方法类似于 flash.ui.Mouse.cursor 属性，唯一的区别是此方法允许对多个鼠标更改光标。

要跟踪并随意阻止鼠标光标变化，请使用 MouseCursorEvent.CURSOR_CHANGE 扩展事件。

参数

cursor :uint – 光标的类型，参见 flash.ui.MouseCursor 静态常数，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 静态属性

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

Scaleform 版本： 4.0.12

返回系统中检测到的控制器数量。

setEdgeAAMode() 静态方法

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

Scaleform 版本： 4.0.12

设置某个具体显示对象及其子对象的 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 – 目标显示对象，应用新的 EdgeAA 模式值。
mode :uint – EdgeAA 模式值。

另请参见：

getEdgeAAMode

getEdgeAAMode() 静态方法

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

Scaleform 版本： 4.0.12

从特定显示对象检索 EdgeAA 模式值。

参数

`dispObj :DisplayObject` – 目标显示对象，检索 EdgeAA 模式值。

另请参见：

`setEdgeAAMode`

visibleRect 静态属性

`visibleRect:Rectangle [read]`

Scaleform 版本: 4.0.14

此属性包含当前可见的矩形。当您更改长宽比、缩放模式、对齐和/或视口缩放比例并想知道当时哪个区域可见时，就会更改此矩形。此矩形可能有负的左上角坐标。

isScaleform 静态属性

`isScaleform:Boolean [read]`

Scaleform 版本: 4.0.14

假如 SWF 在 GFx 内而非其他 Flash 播放器内运行时，返回“真”(true)。

3 DisplayObjectEx 扩展

setRendererString 静态方法

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

Scaleform 版本: 4.0.17

本属性允许在任何 MovieClip 实例中通过 ActionScript 发送自定义变量到渲染器。如果本属性置位，字符串值将作为用户数据传递给渲染器。

无默认值。

示例:

```
import scaleform.gfx.*;

// m is a MovieClip on stage.

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

getRendererString()静态方法

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

Scaleform 版本: 4.0.17

返回作为用户数据被发送到渲染器的字符串值。

setRendererFloat 静态方法

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

Scaleform 版本: 4.0.17

本属性允许在任何 MovieClip 实例中通过 ActionScript 发送自定义变量到渲染器。如果本属性置位，浮点值将作为用户数据传递给渲染器。

无默认值。

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

返回作为用户数据被发送到渲染器的浮点值。

disableBatching 静态方法

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

Scaleform 版本: 4.0.17

此属性禁用针对自定义绘图的网格生成批处理。

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

检查是否禁用针对自定义绘制的网格生成批处理。

4 FocusEventEx 扩展

```
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); }
    }
}
```

此事件是标准 flash.events.FocusEvent 的一个扩展。它添加一个 'controllerIdx' 成员，该成员指明导致该事件的控制器的一个基于零的索引。当把 Extensions.enabled 属性设置为 true 时，Scaleform 始终生成 FocusEventEx 事件，而不是标准 FocusEvent。用户可以检查收到的事件是不是 FocusEventEx 的一个实例，如果是，就把该事件对象归到扩展类型中。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 属性

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

Scaleform 版本： 4.0.12

指明哪个键盘/控制器用于该事件（基于零的索引）。

5 FocusManager 扩展

alwaysEnableArrowKeys 静态属性

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

Scaleform 版本： 4.0.12

此静态属性允许箭头键更改焦点 -- 即使把 `_focusrect` 属性设置为 `false`（捕获焦点时应用）。默认情况下，如果通过 `_focusrect = false` 禁用黄色焦点矩形，Flash 就不允许您使用箭头键更改焦点。要更改此行为，请将 `alwaysEnableArrowKeys` 属性设置为 `true`。

disableFocusKeys 静态属性

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

Scaleform 版本： 4.0.12

此静态属性禁用处理所有焦点键（TAB、Shift-TAB 和箭头键），因此，用户可以实施自己的焦点键管理。

moveFocus() 静态方法

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

Scaleform 版本： 4.0.12

此静态方法用来通过对以下焦点键之一进行模拟按键操作来移动焦点矩形：TAB、Shift-TAB 或箭头键。将此方法与 `disableFocusKeys` 和 `modalClip` 属性一起使用，可实施自定义焦点管理。

参数

keyToSimulate :String - 要模拟的键的名称：“向上”、“向下”、“向左”、“向右”、“tab”、“shifttab”。

startFromMovie:InteractiveObject – 可选参数，用来指定一个字符；moveFocus 将它（而非当前聚焦的参数）用作一个起点。此属性可能是空 (Null) 或未定义，这意味着当前聚焦的字符用作起点；这可能会对指定第三个可选参数有用。

includeFocusEnabledChars :Boolean – 可选标志，允许将 moveFocus 放在仅设置 focusEnabled 属性的字符以及设置了 tabEnabled / tabIndex 属性的字符上。如果不指定该标志，或者将该标志设置为 false，那么只有设置了 tabEnabled / tabIndex 属性的字符才能参加焦点移动。

controllerIdx :uint – 用于该操作的控制器的索引。如果不指定，那就是用默认控制器（控制器 0）。

返回

返回下一个要聚焦的字符，或者如果找不到该字符则返回 null。

另请参见：

findFocus
disableFocusKeys
setModalClip
getModalClip

findFocus() 静态方法

```
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

此静态方法用来通过对以下焦点键之一进行模拟按键操作来查找下一个焦点项：TAB、Shift-TAB 或箭头键。将此方法与 disableFocusKeys 和 setModalClip/getModalClip 扩展相配合，可用来实施自定义焦点管理。

参数

keyToSimulate :String - 要模拟的键的名称：“向上”、“向下”、“向左”、“向右”、“tab”、“shifttab”。

parentMovie:DisplayObjectContainer – 电影剪辑，用作模式剪辑。焦点项搜索仅在此剪辑的子项内执行。可以为 null。

loop :Boolean - 要循环焦点的 Boolean 标志。例如，如果当前聚焦的项目位于底部，而键为“向下”键，那么 findFocus 要么返回 “null”（如果此标志为 “false”）或者返回最上面的可聚焦项目（如果此标志为 “true”）。

startFromMovie:InteractiveObject – 可选参数，用来指定一个字符；moveFocus 将它（而非当前聚焦的参数）用作一个起点。此属性可能为 null 或未定义，这意味着当前聚焦的字符用作一个起点。

includeFocusEnabledChars :Boolean – 可选标志，允许将 moveFocus 放在仅设置 focusEnabled 属性的字符以及设置了 tabEnabled / tabIndex 属性的字符上。如果不指定该标志，或者将该标志设置为 false，那么只有设置了 tabEnabled / tabIndex 属性的字符才能参加焦点移动。

controllerIdx :uint – 正在操纵焦点的控制器的一个零基索引。将此与焦点组一起使用，可提供多控制器焦点支持。

返回

返回下一个要聚焦的字符，或者如果找不到该字符则返回 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

此静态方法与指定 stage.focus 属性的操作相同。唯一不同的是，可以指定应与此操作关联的控制器的索引。

参数

obj:InteractiveObject - 新聚焦的交互式对象

controllerIdx :uint - 表明哪个键盘/控制器用于该事件（基于零的索引）。

getFocus() 静态方法

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

Scaleform 版本： 4.0.12

此静态方法返回与 stage.focus 属性相同的值。**唯一不同的是，可以指定应与此操作关联的控制器的索引**。

参数

`controllerIdx :uint -` 指明哪个键盘/控制器用于该事件（基于零的索引）。

返回

当前聚焦的交互式对象。

numFocusGroups 静态属性

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

Scaleform 版本： 4.0.12

返回焦点组数量，通过调用 `setControllerFocusGroup` 函数进行设置。如果焦点组 0 和 3 处于活动状态，numFocusGroups 将返回 2。

setFocusGroupMask() 静态方法

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

Scaleform 版本： 4.0.12

此方法将一个位掩码设置为一个字符以及全部其子项。此位掩码将焦点组所有权指定给该字符，意思是只有该位掩码中表示的控制器才能将焦点移动到该字符内。使用 `setControllerFocusGroup` 扩展方法，可以将焦点组与控制器关联。

例如，我们假定“按钮 1”只能通过控制器 0 聚焦，而“电影剪辑 2”可通过控制器 0 和 1 聚焦。要实现此行为，请将焦点组与控制器进行关联：

```
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” 位掩码可设置为母电影剪辑。这将会把此掩码值传播到所有其子项。

参数

```
obj:InteractiveObject    - 一个交互式对象
mask:uint                - 一个焦点组位掩码
```

另请参见：

setControllerFocusGroup
getFocusGroupMask

getFocusGroupMask() 静态方法

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

Scaleform 版本： 4.0.12

返回当前焦点组位掩码值（有关详细信息，请参见 setFocusGroupMask）。

参数

```
obj:InteractiveObject    - 一个交互式对象
```

返回

一个用于指定的交互式对象的焦点组位掩码。

另请参见：

setControllerFocusGroup
setFocusGroupMask

setControllerFocusGroup() 静态方法

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

Scaleform 版本： 4.0.12

此静态方法将 controllerIndex 表示的控制器与一个焦点组相关联。默认情况下，所有控制器均与焦点组 0 关联，这意味着它们正在使用同一焦点。不过，可以使每个控制器与其自己的焦点一起工作。例如，如果两个控制器应有单独的焦点（在分屏使用情况下），则 setControllerFocusGroup(1,1) 将为其控制器 1 创建一个单独的焦点组。调用 setControllerFocusGroup(1,0) 将使控制器 0 和 1 再次共享同一焦点。

参数

controllerIdx:uint - 控制器的零基索引。
focusGroupIdx:uint – 焦点组的零基索引。

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

此方法返回与指定控制器关联的焦点组索引。

参数

controllerIndex – 物理控制器的零基索引。

返回

焦点组的基于零的索引。

setModalClip() 静态方法

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

Scaleform 版本： 4.0.12

此静态方法将指定的电影剪辑设置为用于焦点管理的“模式”(modal) 剪辑。这意味着 TAB、Shift-TAB 以及箭头键将仅在所有 “tabable” 子项之间的指定电影剪辑内移动焦点。

参数

`controllerIdx:uint -` 控制器的一个零基索引。
`mc:Sprite -` 一个模式剪辑。

getModalClip() 静态方法

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

Scaleform 版本： 4.0.12

此静态方法返回用于指定控制器的模式剪辑。

参数

`controllerIdx` – 控制器的零基索引。

返回

一个模式剪辑，如果未找到，则表明未定义。

6 GamePad 扩展

控件常数

此类为一般游戏手柄控件（如触发器、模拟摇杆和按钮）提供辅助常数。他们是 SF_KeyCodes.h 中定义的常数的镜面反射。Scaleform 在运行时插入正确的值，因此，编译利用这些常数的 SWF 的操作就会按预期进行。

为了方便起见，Scaleform FxPlayer 框架将游戏手柄控件映射到键盘当量，然而，自定义的集成或应用程序可将这些常数与 GFx::Movie::HandleEvent 一起使用，如果对于 AS3 键盘事件有必要的话，在控制器与键盘之间产生某种区别。

Scaleform FxPlayer 框架确实将这些常数与 GamePadAnalogEvents 一起使用，为这样的模拟值提供适当反馈。不过，也可以想到的是，开发者也可以不使用 GamePad 常数，而是使用键盘代码。是否选择将游戏手柄事件与键盘事件结合在一起，留给开发者自行做出判断。

supportsAnalogEvents() 静态方法

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

Scaleform 版本： 4.0.13

假如基本硬件平台的 Scaleform 实现支持游戏手柄模拟事件（如对于触发器和拇指棒），就返回 true。此值可用来确定是否支持 GamePadAnalogEvents。

另请参见：

GamePadAnalogEvent

7 GamePadAnalogEvent 扩展

```
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 支持针对某个特定平台的游戏手柄模拟事件，就触发此事件。此事件只从 Stage 发出，而所有侦听者都会附加到 Stage 对象。Scaleform FxPlayer 框架通过用于 'code' 属性的适当 GamePad 常数触发这些事件，不过，如有必要，开发者可以在自己的 Scaleform 集成中使用其它值（如键盘值）。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 属性

```
code : uint
```

Scaleform 版本： 4.0.13

指明使用哪个键/控件生成此事件。Scaleform FxPlayer 框架将使用游戏手柄常数。参见 GamePad。

controllerIdx 属性

```
controllerIdx : uint
```

Scaleform 版本： 4.0.13

指明哪个键盘/控制器用于该事件（基于零的索引）。

xvalue 属性

```
xvalue : Number
```

Scaleform 版本： 4.0.13

指明 x 轴中的当前值。该值将在 -1 与 1 之间标准化，其中包含 -1 和 1。

yvalue 属性

```
yvalue : Number
```

Scaleform 版本： 4.0.13

指明 y 轴中的当前值。该值将在 -1 与 1 之间标准化，其中包含 -1 和 1。

8 InteractiveObjectEx 扩展

getHitTestDisable() 静态方法

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

Scaleform 版本： 4.0.13

返回 'hitTestDisable' 标志的状态。当设置为 `true` 时，`MovieClip.hitTest` 函数在命中测试检测中忽略此交互式对象。此外，所有其它鼠标事件都传播到该对象。

默认值为 `false`。

参数

o - An interactive object.

返回

一个代表 'hitTestDisable' 标志的状态的 Boolean 值。

另请参见：

`InteractiveObjectEx.setHitTestDisable`

setHitTestDisable() 静态方法

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

Scaleform 版本： 4.0.13

设置 'hitTestDisable' 标志的状态。当它设置为 `true` 时，`MovieClip.hitTest` 函数在命中测试检测中忽略此交互式对象。**此外，所有其它鼠标事件都传播到该对象。**默认值为 `false`。

参数

o - 一个交互式对象。

f - 一个代表 'hitTestDisable' 标志的新状态的 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' 标志的状态。设置为 `true` 时，此字符显示在所有其它字符的上面，而不管其深度如何。

参数

`o - An interactive object.`

返回

一个代表 'topmostLevel' 标志的状态的 Boolean 值。

另请参见：

InteractiveObjectEx.setTopmostLevel

setTopmostLevel () 静态方法

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

Scaleform 版本： 4.0.13

设置 'topmostLevel' 标志的状态。设置为 `true` 时，此字符显示在所有其它字符的上面，而不管其深度如何。当把光标从各个级别拉到对象上面时，这可能对于实现自定义鼠标光标有用。默认值为 `false`。
如果把多个字符标为 "topmostLevel"，则拖曳顺序与没有将字符标为最上面时相同，也就是说，如果对象 A 被拖到对象 B 下面，则在把上述字符标为最上面时，对象 A 仍将会在对象 B 下面，而不管把 "topmostLevel" 属性设置为 `true` 的顺序如何。
注意：一旦某个字符标为 "topmostLevel"，swapDepth ActionScript 函数就不会对此字符产生任何影响。默认值为 `false`。

参数

o - 一个交互式对象。

f - 一个代表 'topmostLevel' 标志的新状态的 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); }
    }
}
```

此事件是标准 flash.events.KeyboardEvent 的一个扩展。它添加一个 'controllerIdx' 成员，该成员指明导致该事件的控制器的一个基于零的索引。当把 Extensions.enabled 属性设置为 true 时，Scaleform 始终生成 KeyboardEventEx 事件，而不是标准 KeyboardEvent。用户可以检查收到的事件是不是 KeyboardEventEx 的一个实例，如果是，就把该事件对象归到扩展类型中。**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 属性

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

Scaleform 版本： 4.0.12

指明哪个键盘/控制器用于该事件（基于零的索引）。

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);
  }
    }
}
```

此事件用来跟踪和/或防止鼠标光标变化。它设置有以下事件的属性：
bubbles – false，它不起泡 (bubble)
cancellable – true，可通过调用 preventDefault() 方法来防止默认操作（光标变化）。

此事件对于实施自定义的动画鼠标光标非常有用。只要 Scaleform 更改鼠标光标的形状（通过翻转文本字段或按钮，或者通过设置 flash.ui.Mouse.cursor 属性），都会对**一个 stage** 触发此事件（假如将 Extensions.enabled 设置为 true 的话）。**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：只有在一个 Stage 上设置了侦听器，才触发此事件。

cursor 属性

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

Scaleform 版本： 4.0.13

此属性指明要更改为的光标的类型。它包含来自 flash.ui.MouseCursor 类的字符串值之一，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”`

mouseIdx 属性

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

Scaleform 版本： 4.0.13

指明为哪个鼠标/控制器生成该事件（基于零的索引）。

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); }
    }
}
```

此事件是标准 flash.events.MouseEvent 的一个扩展。它为支持多控制器和鼠标右/中键添加属性。当把 Extensions.enabled 属性设置为 true 时，Scaleform 始终生成 MouseEventEx 事件，而不是标准 MouseEvent。用户可以检查收到的事件是不是 MouseEventEx 的一个实例，如果是，就把该事件对象归到扩展类型中。**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：一旦启用扩展，就会为鼠标右、中、中心等键触发鼠标事件，而且用户必须检查 buttonIdx 属性，搞清楚哪个按钮生成该事件。

buttonIdx 属性

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

Scaleform 版本： 4.0.13

指明为哪个按钮生成该事件（基于零的索引）。此值可以为下列值之一：

- MouseEventEx.LEFT_BUTTON :uint = 0 - 鼠标左键
- MouseEventEx.RIGHT_BUTTON :uint = 1 - 鼠标右键
- MouseEventEx.MIDDLE_BUTTON :uint = 2 - 鼠标中键
- 假如鼠标有 3 个按键，大于 2 的任何值都是合法的。

mouseIdx 属性

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

Scaleform 版本： 4.0.13

指明为哪个鼠标/控制器生成该事件（基于零的索引）。

nestingIdx 属性

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

Scaleform 版本： 4.0.13

此属性对于 rollOver/Out、mouseOver/Out 和 dragOver/Out 事件来说是可选的。此参数指定同一字符上的嵌套的翻转/拖出 (rollover/dragover) 事件的索引。
当生成嵌套的 rollOver/rollOut、mouseOver/Out 和 dragOver/dragOut 事件（单独针对每个鼠标光标），此参数代表嵌套的基于零的索引：初始事件将会把 0 作为参数；如果第二个光标翻转同一个字符，则把该成员设置为 1，就会触发第二个 rollOver/rollOut 事件。如果有任何光标离开该字符，该成员设置为 1，就会触发 rollOut/mouseOut/dragOut；把该成员设置为 0，就会触发最后一个 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); }
    }
}
```

此事件是标准 flash.events.TextEvent 的一个扩展。它添加一个 'controllerIdx' 成员，该成员表明导致该事件的控制器的一个基于零的索引。当把 Extensions.enabled 属性设置为 true 时，Scaleform 始终生成 TextEventEx 事件，而不是标准 TextEvent。用户可以检查收到的事件是不是 TextEventEx 的一个实例，如果是，就把该事件对象归到扩展类型中。**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

指明哪个键盘/控制器用于该事件（基于零的索引）。

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

开发者现在能够使用 TextFieldEx.LINK_MOUSE_OVER 和 TextFieldEx.LINK_MOUSE_OUT 事件扩展侦听 TextField 链接（由 HTML 标记指定）上的鼠标 over/out 事件。这些事件的行为与其它 AS3 事件相同，而且可以使用 addEventListener 范例进行侦听。

buttonIdx 属性

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

Scaleform 版本: 4.0.14

表明使用哪个鼠标按钮启动 TextEventEx（MouseEventEx.LEFT_BUTTON、MouseEventEx.RIGHT_BUTTON 等；参阅 MouseEventEx.buttonIdx）。

13 System 扩展

actionVerbose 静态属性

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

Scaleform 版本： 4.0.12

启用/禁用操作码跟踪。

getStackTrace() 静态方法

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

Scaleform 版本： 4.0.12

获取格式化为字符串的当前堆栈跟踪。

getCodeFileName() 静态方法

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

Scaleform 版本： 4.0.12

获取当前执行的代码的文件名。

14 TextFieldEx 扩展

appendHtml() 静态方法

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

Scaleform 版本： 4.0.12

将 newHtml 参数指定的 HTML 附加到文本字段的文本末尾。此方法比一个 `htmlText` 属性上的添加指定 (+=)（如 `txt.htmlText += moreHtml`）更加有效。`htmlText` 属性上的常规 += 生成 HTML 字符串，附加新的 HTML 部分，然后从头解析整个 HTML。此函数执行增量 HTML 解析，即它仅解析来自 `newHtml` 字符串参数的 HTML（这就是 `newHtml` 参数中的 HTML 有理由的原因，这对于 += 操作符并非必需的）。它对于包含大量内容的文本字段尤其重要。

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` – 将 HTML 附加到的一个文本字段。
`newHtml:String` – 要将 HTML 附加到现有文本的字符串。

setIMEEnabled() 静态方法

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

Scaleform 版本： 4.0.12

启用/禁用对指定文本字段的 IME。如果将 `isEnabled` 设置为 `false`，则不让在此文本字段中激活 IME。默认情况下，启用了 IME。

参数

`textField :TextField` – 要操作的文本字段。
`isEnabled :Boolean` – 如果是 `true` – 启用 IME；否则禁用。

setVerticalAlign() 静态方法

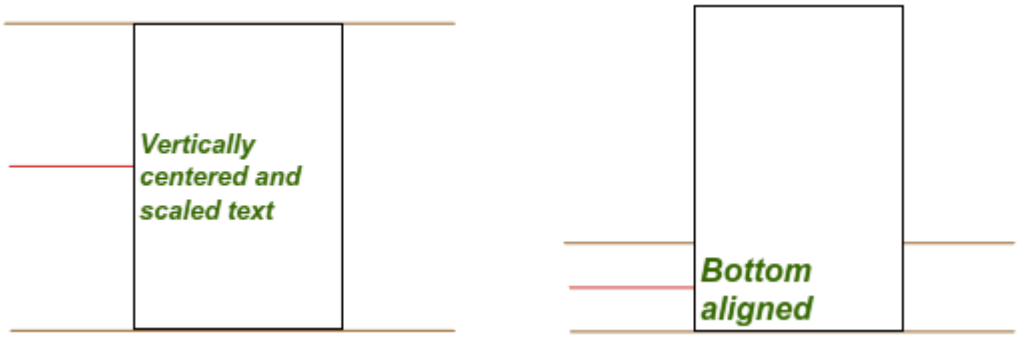
```
public function setVerticalAlign(textField:TextField, valign:String) :void
```

Scaleform 版本： 4.0.12

在文本框内设置文本垂直对齐。该属性的有效值为 TextFieldEx 类中声明的下列常数：

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

如果将此属性设置为 `center`，则文本位于文本框内中心；如果设置为 `bottom`，则文本位于文本框内底部（参见下图）：



垂直居中和缩放文本　　*底部对齐*

默认值为 `top`

参数

`textField:TextField` - 设置垂直对齐的文本字段
`valign:String` - 对齐值 ("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。