

Autodesk® Scaleform®

Scaleform CLIK AS3 用户指南

本文包括了 Scaleform CLIK 框架及所带内置组件执行方法的详细使用说明

作者: Prasad Silva, Matthew Doyle
版本: 2.0
最后修订: 2010 年 8 月 19 号

Copyright Notice

Autodesk® Scaleform® 4.2

© 2012 Autodesk, Inc. All rights reserved. Except as otherwise permitted by Autodesk, Inc., this publication, or parts thereof, may not be reproduced in any form, by any method, for any purpose.

Certain materials included in this publication are reprinted with the permission of the copyright holder.

The following are registered trademarks or trademarks of Autodesk, Inc., and/or its subsidiaries and/or affiliates in the USA and other countries: 123D, 3ds Max, Algor, Alias, AliasStudio, ATC, AUGI, AutoCAD, AutoCAD Learning Assistance, AutoCAD LT, AutoCAD Simulator, AutoCAD SQL Extension, AutoCAD SQL Interface, Autodesk, Autodesk Homestyler, Autodesk Intent, Autodesk Inventor, Autodesk MapGuide, Autodesk Streamline, AutoLISP, AutoSketch, AutoSnap, AutoTrack, Backburner, Backdraft, Beast, Beast (design/logo) Built with ObjectARX (design/logo), Burn, Buzzsaw, CAiCE, CFdesign, Civil 3D, Cleaner, Cleaner Central, ClearScale, Colour Warper, Combustion, Communication Specification, Constructware, Content Explorer, Creative Bridge, Dancing Baby (image), DesignCenter, Design Doctor, Designer's Toolkit, DesignKids, DesignProf, DesignServer, DesignStudio, Design Web Format, Discreet, DWF, DWG, DWG (design/logo), DWG Extreme, DWG TrueConvert, DWG TrueView, DWFX, DXF, Ecotect, Evolver, Exposure, Extending the Design Team, Face Robot, FBX, Fempro, Fire, Flame, Flare, Flint, FMDesktop, Freewheel, GDX Driver, Green Building Studio, Heads-up Design, Heidi, Homestyler, HumanIK, i-drop, ImageModeler, iMOUT, Incinerator, Inferno, Instructables, Instructables (stylized robot design/logo), Inventor, Inventor LT, Kynapse, Kynogon, LandXplorer, Lustre, MatchMover, Maya, Mechanical Desktop, MIMI, Moldflow, Moldflow Plastics Advisers, Moldflow Plastics Insight, Moondust, MotionBuilder, Movimento, MPA, MPA (design/logo), MPI (design/logo), MPX, MPX (design/logo), Mudbox, Multi-Master Editing, Navisworks, ObjectARX, ObjectDBX, Opticore, Pipeplus, Pixlr, Pixlr-o-matic, PolarSnap, Powered with Autodesk Technology, Productstream, ProMaterials, RasterDWG, RealDWG, Real-time Roto, Recognize, Render Queue, Retimer, Reveal, Revit, RiverCAD, Robot, Scaleform, Scaleform GFx, Showcase, Show Me, ShowMotion, SketchBook, Smoke, Softimage, Sparks, SteeringWheels, Stitcher, Stone, StormNET, Tinkerbox, ToolClip, Topobase, Toxik, TrustedDWG, T-Splines, U-Vis, ViewCube, Visual, Visual LISP, Vtour, WaterNetworks, Wire, Wiretap, WiretapCentral, XSI.

All other brand names, product names or trademarks belong to their respective holders.

Disclaimer

THIS PUBLICATION AND THE INFORMATION CONTAINED HEREIN IS MADE AVAILABLE BY AUTODESK, INC. "AS IS." AUTODESK, INC. DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE REGARDING THESE MATERIALS.

How to Contact Autodesk Scaleform:

Document	CLIK AS3 User Guide
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 简介	1
1.1 概要	1
1.1.1 在 Scaleform CLIK 包含的内容	1
1.2 UI 关注	2
1.3 组件理解	3
1.3.1 Inspectable Properties	3
1.4 框架基础知识	4
1.4.1 事件	4
1.4.2 焦点	5
2 内置组件	6
2.1 基本按钮和文本类型	6
2.1.1 Button	8
2.1.2 CheckBox	12
2.1.3 Label	16
2.1.4 TextInput	18
2.1.5 TextArea	22
2.2 分组类型	26
2.2.1 RadioButton	26
2.2.2 ButtonGroup	31
2.2.3 ButtonBar	32
2.3 滚动类型	35
2.3.1 ScrollIndicator	35
2.3.2 ScrollBar	37
2.3.3 Slider	40
2.4 列表类型	42
2.4.1 NumericStepper	44
2.4.2 OptionStepper	46
2.4.3 ListItemRenderer	49
2.4.4 ScrollingList	54
2.4.5 TileList	59
2.4.6 DropdownMenu	64
2.5 进度类型	69
2.5.1 StatusIndicator	69

2.6	其他类型.....	71
2.6.1	Window	72
3	艺术细节	75
3.1	最优方法.....	75
3.1.1	像素图像	75
3.1.2	蒙版	76
3.1.3	动画	76
3.1.4	图层和绘制元素	76
3.1.5	复杂皮肤界面.....	77
3.1.6	2 的指数倍	77
3.2	已知问题和推荐工作流程	77
3.2.1	复制组件	77
3.3	皮肤绘制实例.....	79
3.3.1	StatusIndicator 皮肤绘制	80
3.4	字体和本地化.....	82
3.4.1	概要	82
3.4.2	内置字体	82
3.4.3	在 textField 嵌入字体	82
3.4.4	本地化系统.....	83
4	编程详述	84
4.1	UI 组件 UIComponent	84
4.1.1	初始化.....	85
4.2	组件状态.....	85
4.2.1	按钮组件	85
4.2.2	非按钮交互组件	87
4.2.3	非交互组件.....	87
4.2.4	特殊案例	87
4.3	事件模型.....	88
4.3.1	最佳使用方法.....	88
4.4	运行时创建组件	89
4.5	焦点处理.....	90
4.5.1	最佳使用方法	90
4.5.2	在复合附件捕获焦点	91
4.6	输入处理.....	91
4.6.1	最佳使用方法	91
4.6.2	多鼠标光标.....	93

4.7	失效.....	94
4.7.1	最佳使用方法.....	94
4.8	组件缩放.....	95
4.8.1	Scale9Grid.....	95
4.8.2	强制.....	95
4.9	组件和数据设置.....	96
4.9.1	最佳使用方法.....	96
4.10	动态动画.....	97
4.10.1	最佳使用方法.....	98
4.11	布局框架.....	98
4.11.1	布局.....	99
4.11.2	LayoutData.....	100
4.12	弹出式支持.....	101
4.12.1	最佳使用方法.....	101
4.13	拖放.....	102
4.13.1	最佳使用方法.....	102
5	实例.....	104
5.1	基础.....	104
5.1.1	包含两个 textField 的 ListItem Renderer	104
5.1.2	像素滚动视图	106
6	常见问题解答	108
7	CLIK AS3 vs. CLIK AS2	109

1 简介

本文档提供了 Scaleform® 通用精简接口工具（Common Lightweight Interface Kit，CLIK™）框架和组件的详细使用说明。在深入认识 CLIK 用户指南之前，希望开发者能够先阅读 [CLIK 入门](#)。这两个使用指南介绍了创建和运行 Scaleform CLIK 所需要的步骤，介绍了基本的概念并提供了创建和美化 CLIK 组件的详细指南。然而，专业级用户更喜欢在学习入门文档时直接查阅本文档作为参考。

1.1 概要

Scaleform CLIK 为一组 ActionScript™ 2.0 (AS2) 用户接口元素、库和工作流增强工具集，Scaleform 4.0 用户为游戏控制器、PC 和掌上游戏机应用快速实现丰富和高效的接口。框架的主要目标为构建一个精简（依据内存和 CPU 使用量）、易于绘制皮肤并高度个性化的架构。另外，对于一个基本的 UI 内核类和系统基本框架，CLIK 包含了 15 个以上内置通用接口元素（例如，按钮、滚动条、文本输入框），将帮助开发者快速创建和重构用户接口界面。

1.1.1. 在 Scaleform CLIK 包含的内容

组件：简单扩展内置 UI 控制组件

Button	Slider
ButtonBar	StatusIndicator
CheckBox	TileList
RadioButton	Label
TextInput	ScrollingList
TextArea	DropdownMenu
ScrollIndicator	NumericStepper
ScrollBar	

类：系统核心 API 函数

InputManager	UIComponent
FocusManager	Tween
DragManager	DataProvider
PopUpManager	IDataProvider
IUIComponent	IList
Constraints	IListItemRenderer

文档和实例文件:

[Getting Started with CLIK](#)

[Getting Started with CLIK Buttons](#)

[CLIK AS3 API Reference](#)

[CLIK AS3 User Guide](#)

[CLIK Flash Samples](#)

[CLIK Video Tutorials](#)

1.2 UI 关注

创建一个美观的 UI 界面，第一步为在纸面或画图编辑器上如 Microsoft Visio® 上绘制草图。第二步为在 Flash 中开始原型化 UI 界面，其目的为在专门图形设计之前绘制出所有的界面项和流程图。Scaleform CLIK 就是专门设计用于使用户快速原型化和重构以完成整个过程。

构建一个完整的 UI 界面有很多不同的方法。在 Flash 中，不存在页的概念，而在 Visio 或其他流程图绘制软件中有此概念，这里使用了关键帧和动画剪辑代替。如果所有的页面都在同一个 Flash 文件中，文件将占据更多内存，但是在页间切换更加快速且容易。如果每个页分别在单独的文件内存放，整体的内存使用量可以降低，但是导入时间更长。将每个页作为单独的文件也具有一些优势，可以使多个设计师和美工人员同时处理相同界面。而且作为技术和设计的需求，在决定 UI 项目结构时可以确保考虑到工作流程。

与传统的桌面或 web 应用不同，这里特别需要了解可以有很多种不同的方法来创建丰富的多媒体游戏界面。每个引擎，甚至不同的平台，实现方法也有所不同，有的方法使用起来更加高效，而有的则更加低效。例如，放置一个多页界面到单个 Flash 文件。这样管理起来更加方便，转换操作也更加容易，但是增加了内存的消耗，对于老的游戏控制器或移动终端则非常不利。如果一切都在单个 Flash 文件中进行，则不能使多个设计师同时工作在同一个界面的不同部分。如果项目为一个复杂的界面设计，需要一个庞大的设计团队，或者具有其他特殊的技术和设计需求，则最好将每个 UI 页面放置到不同的 Flash 文件。在很多情况下，这两种方案都是可行的，但是，在特定的项目中，其中一种可能比另外一种方案更具有优势。例如，屏幕可能需要根据需求导入和导出，或者在网络上动态更新和传输。衡量的底线为应该仔细评估 UI 界面资源的管理，从 UI 的逻辑规划到工作流实现以及性能都需要考虑完善。

虽然 Scaleform 强烈建议开发者使用 Flash 和 ActionScript 作为 UI 界面主要开发手段，但没有完美的答案，某些团队也许更喜欢用应用引擎脚本语言（如 Lua 或 UnrealScript）来完成大多数繁重工作。在这些情况下，Flash 应该被主要用作动画实现工具，只包含极少量的动态脚本 ActionScript 和 Flash 文件内部的交互内容。

在早期了解技术、设计和工作流程的需求非常重要，然后继续评估整个过程的整体方法，特别是在深入项目之前，确保顺利和最终的成功。

1.3 组件理解

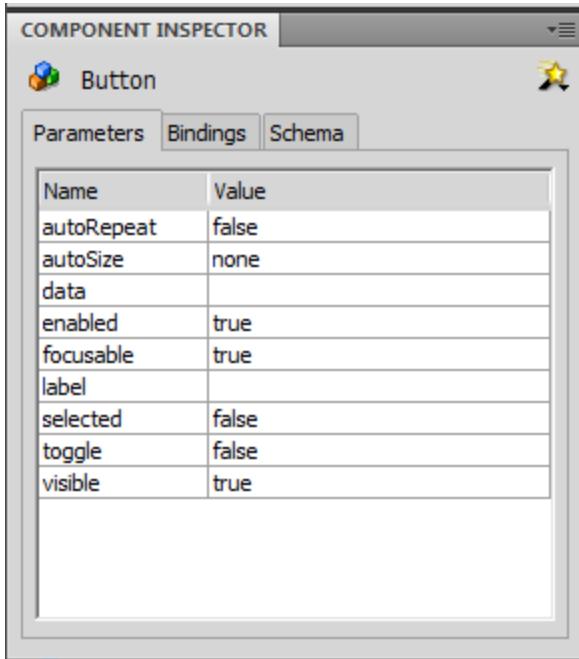
在开始之前，开发者需要理解 Flash 组件的确切技术细节。Flash 的一系列默认接口创建工具和编译块我们称之为组件。但是，在本文档中“组件”指使用 Scaleform CLIK 框架创建的内置组件，这些组件由 Scaleform 与世界知名的 gskinner.com 团队联合开发。

gskinner.com 由 Grant Skinner 领导，Grant Skinner 由 Adobe 任命负责为 Flash Creative Suite® 4 (CS4) 创建组件，为世界知名的 Flash 领先开发团队之一。关于 gskinner.com 的更多信息，请访问 <http://gskinner.com/blog>。

为更深入理解内置 CLIK 组件，在 Flash studio 中开打默认的 CLIK 文件：
Scaleform SDK\Resources\AS3\CLIK\components\CLIK_Components_AS3.fla

1.3.1 Inspectable Properties

组件的重要属性可以通过 Flash IDE 的 Component Inspector 面板或者 Parameters 标签进行设置。需要在 CS4 中打开该面板，在上方工具条中选择 Window 下拉菜单，点击启动 Component Inspector 窗口，或者按下(**Shift+F7**)键。这将打开 Component Inspector 面板。这些被称之为“检查属性”。这为不熟悉 AS 编程美工和设计师配置组件行为和功能提供了一种便利的方法。



改变属性只能在发布包含该组件的 SWF 文件后才有效。Flash IDE 在设计阶段场景中不显示任何改变，因为 CLIK 组件不属于编译剪辑。这是用来确保组件易于使用和绘制皮肤。

1.4 框架基础知识

1.4.1 事件

多数组件产生进行用户互动、状态更改和焦点管理的事件。这些事件对于使用 CLIK 组件创建功能性用户界面来说非常重要。

所有事件回调均会收到单个 Event (事件) 或 Event 子类参数，该参数包含有相应事件的相关信息。下列属性是所有事件所共有的。

- **type:** 事件类型。
- **target:** 事件目标。
- **currentTarget:** 通过一个事件侦听器积极处理 Event 对象的对象。
- **eventPhase:** 事件流中的当前阶段。（EventPhase.CAPTURING_PHASE、EventPhase.AT_TARGET、EventPhase.BUBBLING_PHASE）。
- **bubbles:** 指明事件是不是起泡事件。
- **cancelable:** 指明与事件关联的行为是否可以避免

有关 ActionScript 3 Event 类的更多信息，请参阅 Adobe 的 ActionScript 3 参考文档：

http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/events/Event.html

有关每个组件生成的所有事件的列表，请参阅本用户指南中的“预建组件”一节，其中还包含有相应事件的任何独特属性。

在 Scaleform 内，某些本机事件类型将会被 Scaleform 的扩展所取代。例如，当把 `Extensions.enabled` 属性设置为 `true`（真）时，Scaleform 始终生成 `MouseEventEx`，一个扩展 `MouseEvent` 事件（而不是标准 `MouseEvent`）的类。`MouseEventEx` 添加了支持多控制器和鼠标右/中按钮的属性。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。

1.4.2 焦点

在 Flash 内，有一个焦点的概念。默认情况下，`stage` 焦点设置为通过鼠标或键盘选定的最后一个 `InteractiveObject`。

通常，只有一个获得焦点的组件接收键盘事件。设置一个组件的焦点有多种方法。其中一些方法在本文的 [Programming 设计细节](#) 中有所描述。大多数 CLIK 组件当交互时，特别是按下鼠标左键或类似控制器在上面按下（点击）时也可以接收焦点。`(Tab)` 和`(Shift+Tab)`键（或对应导航控制）可以在显示的焦点组件上移动焦点。这种特性也是大多数桌面应用软件和网页上所提供的。注意非 CLIK 元素使用的焦点也可以被 CLIK 组件使用。这意味着一个 Flash 开发者能够将 CLIK 元素和非 CLIK 元素在场景相互混合和匹配并使焦点行为按意图进行，特使在使用`(Tab)` 和`(Shift+Tab)`键时可以在场景中移动焦点。

默认情况下按钮响应 `(Enter)` 键和空格键。将鼠标箭头移动到按钮上面然后移开也会使组件产生动作，拖动鼠标光标移入和移出也一样。

在游戏控制器或掌上游戏机，开发者能简单用对应游戏杆来控制键盘和鼠标控制事件。例如，`(Enter)` 键在 Xbox360 或 PS3 控制器上通常映射为 `(A)` 或 `(X)` 按钮。此映射使 UI 界面中使用 CLIK 应用到多种类型的平台之上。

2 内置组件

初次观察，Scaleform CLIK 为一个基本的内置 UI 组件集，但是 CLIK 的真实意图是为创建丰富组件和界面提供一个框架。开发者可以自由—并可以按设想进行扩展—创建自定义组件适应自身需求并在 CLIK 框架上进行构建。

内置 CLIK 组件提供标准的 UI 功能，从基本的按钮和复选框到列表框、下拉菜单和模式对话框等复合组件。开发者可以方便得将这些标准组件进行扩展，增加更多特性或在从头创建自定义组件时作为简单的参考组件。

以下选项详细得描述了每个内置组件。他们按照复杂性和功能性进行分组。每个组件使用子章节列表进行描述。

- **User interaction:** 用户如何与组件进行交互。
- **Component setup:** 当在 Flash 授权环境中构造组件时需要的元素。
- **States:** 组件到函数所需要的的不同可视状态（关键帧）。
- **Inspectable properties:** 在 Flash 授权环境中公布的属性，便于不使用代码配置特定组件特性。
- **Events:** 在 UI 界面中其他对象可以监听的组件所触发的事件列表。
- **Tips and tricks:** 完成各种与讨论组件相关的任务的实例代码。

2.1 基本按钮和文本类型

基本类型包括 Button、CheckBox、Label、TextInput 和 TextArea 组件。按钮 Button 构成了多数用户界面的主干，CheckBox 继承了 Button 的功能。Label 为一个静态标签类型，而 TextInput 和 TextArea 分别可以表示单行文本和多行文本。



图 1：来自 **Free Realms** 的主菜单按钮实例

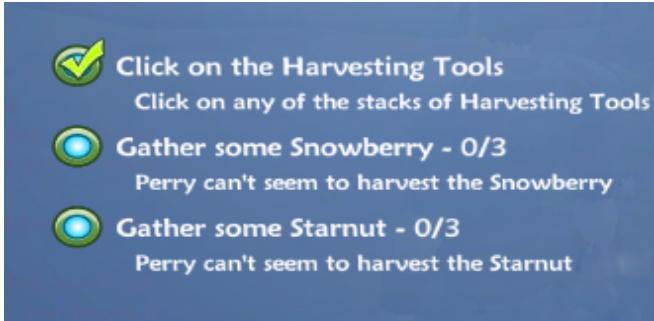


图 2: 来自 *Free Realms* 复选框 (Check box) 实例



图 3: 来自 *Free Realms* 的标签 (Label) 实例



图 4: 来自 *Crysis Warhead* 的文本输入框 (Text input) 实例

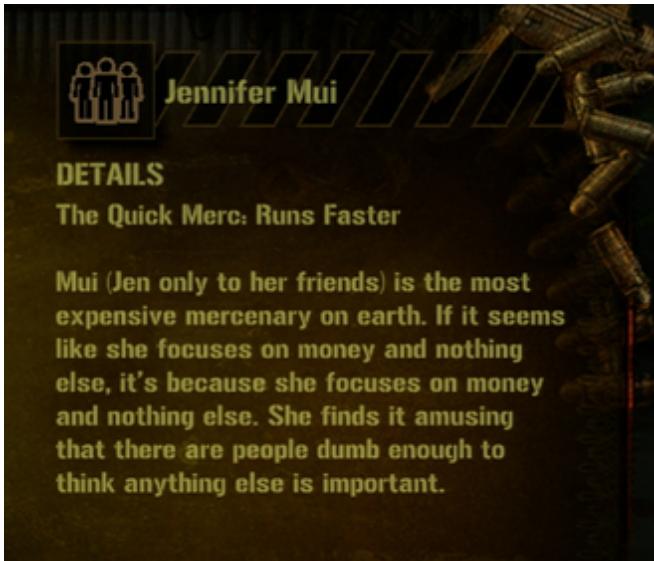


图 5: 来自 *Mercenaries 2* 的文本区域 (Text area) 实例

2.1.1 Button



图 6：无皮肤按钮

按钮 **Button** 为 CLIK 框剪的基本组件，可以在任何地方使用需要一个能发出滴答声的界面控制。默认的 **Button** 类 (`scaleform.clik.controls.Button`) 支持一个 **textField** 来显示一个标签，并制定可视化用户互动。按钮可以单独使用，也可以作为合成组件的一部分，如作为 **ScrollBar** 的方向按钮或者 **Slider** 翻页按钮。大多数交互组件可以响应点击动作或为扩展按钮。

2.1.1.1 用户交互

按钮组件可以用鼠标或任何类似控制器点击。当获得焦点时也可以通过键盘按钮控制。

2.1.1.2 组件设置

一个使用 CLIK **Button** 类的 **MovieClip** 必须具备下面所列的子单元。也列出了可选元素。

- **textField:** (可选) **TextField** 类型。按钮标签
- **focusIndicator:** (可选) **MovieClip** 类型。一个单独的 **MovieClip** 用来显示焦点状态。如果被使用，必须有两个命名帧：“show”和“hide”。默认情况下，**over** 状态用来表示一个获得焦点的 **Button** 组件。但是在有些类中，这个行为限制了使用，设计师可能要将焦点状态和鼠标 **over** 状态分开使用。

2.1.1.3 状态

CLIK 按钮组件支持基于用户交互的不同的状态。这些状态包括：

- **up** 或默认状态；
- 当鼠标箭头在组件上方或者获得焦点时 **over** 状态；
- 当按钮按下时 **down** 状态；
- **disabled** 状态；

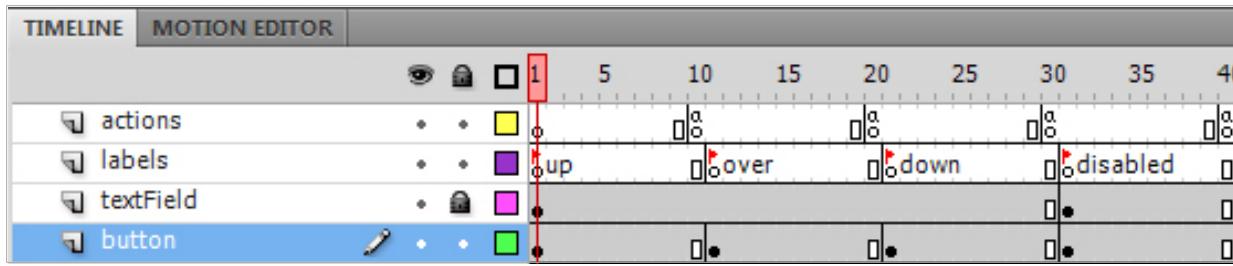


图 8: Button 时间轴

这些状态在 Flash 时间轴中用关键帧来表示，为按钮组件所需要的最少关键帧设置的正确操作。同时还有其余状态扩展组件功能以支持复杂用户交互和动画转换，这些信息在文档 [CLIK 按钮入门](#) 中有所描述。

2.1.1.4 属性检查

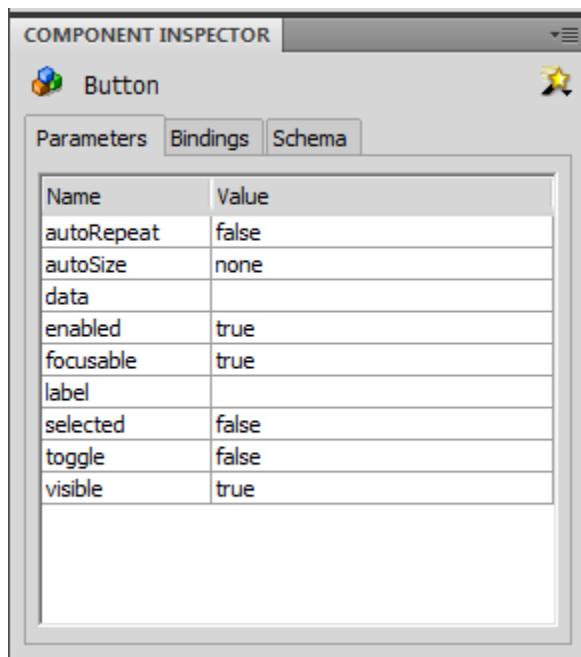


图 9: CS4 组件观察中的按钮组件属性检查窗口

按钮组件的检查属性为：

autoRepeat	确定在按下并保持住按钮时该按钮是否发出 "click" (点击) 事件。
autoSize	确定按钮是否进行缩放以适应其所包含的文本以及已调整大小的按钮将向哪个方向对齐。将 <code>autoSize</code> 属性设置为 <code>TextFieldAutoSize.NONE</code> 将会使大小保持不变。
Data	与该按钮相关的数据。在 <code>ButtonGroup</code> 中使用按钮时此属性尤其有用。
enabled	如果设置为 <code>false</code> (假)，就禁用该按钮。已禁用的组件将不再收到用户输入。
focusable	启用/禁用对于组件的焦点管理。将 <code>focusable</code> (可聚焦) 属性设置为 <code>false</code> 将

	会取消对 tab 键、方向键以及基于鼠标的焦点更改的支持。
Label	设置按钮中的文本显示。
selected	设置该按钮的选定状态。按钮可以有两组鼠标状态：已选定状态和未选定状态。当一个按钮的 <code>toggle</code> (切换) 属性为 <code>true</code> 时，点击该按钮时就会更改已选定的状态，不过，即使 <code>toggle</code> 属性为 <code>false</code> ，也可以使用 ActionScript 设置已选定状态。
toggle	设置按钮套索模式。如果设置为 <code>true</code> ，按钮将作为一个套索按钮使用。
visible	如果设置为 <code>false</code> 则隐藏按钮。

2.1.1.5 事件

所有事件回调均会收到单个 `Event` (事件) 或 `Event` 子类参数，该参数包含有相应事件的相关信息。下列属性是所有事件所共有的。

- **type:** 事件类型。
- **target:** 事件目标。
- **currentTarget:** 通过一个事件侦听器积极处理 `Event` 对象的对象。
- **eventPhase:** 事件流中的当前阶段。（`EventPhase.CAPTURING_PHASE`、`EventPhase.AT_TARGET`、`EventPhase.BUBBLING_PHASE`）。
- **bubbles:** 指明事件是不是起泡事件。
- **cancelable:** 指明与事件关联的行为是否可以避免

按钮组件产生事件列表如下。事件旁边的属性列表为通用属性的补充内容。

ComponentEvent.SHOW	运行时可视属性已设置为 <code>true</code>
ComponentEvent.HIDE	运行时可视属性已设置为 <code>false</code>
ComponentEvent.STATE_CHANGE	组件的状态已更改。
E	
FocusHandlerEvent.FOCUS_IN	组件已收到焦点。
FocusHandlerEvent.FOCUS_OUT	组件已失去焦点。
UT	
Event.SELECT	选定的属性已发生变化。
MouseEvent.ROLL_OVER	鼠标光标滑过该按钮。 <code>mouseIdx</code> : 用来生成该事件的鼠标光标的索引（仅适用于多鼠标光标环境）。 <code>uint</code> 类型。仅限于 <code>Scaleform</code> ，需要将该事件归于 <code>MouseEventEx</code> 。 <code>buttonIdx</code> : 指明为哪个按钮生成该事件（基于零的索引）。仅限于 <code>Scaleform</code> ，需要将该事件归于 <code>MouseEventEx</code> 。

MouseEvent.ROLL_OUT	鼠标光标已滑离该按钮。 <i>mouseldx</i> : 用来生成该事件的鼠标光标的索引（仅适用于多鼠标光标环境）。uint 类型。仅限于 Scaleform，需要将该事件归于 MouseEventEx。 <i>buttonIdx</i> : 指明为哪个按钮生成该事件（基于零的索引）。仅限于 Scaleform，需要将该事件归于 MouseEventEx。
ButtonEvent.PRESS	已按该按钮。 <i>controllerIdx</i> : 用来生成该事件的控制器的索引（仅适用于多鼠标光标环境）。uint 类型。 <i>isKeyboard</i> : 为 true - 假如事件是由计算机键盘 (Keyboard)/游戏键盘 (Gamepad) 生成的；为 false – 假如事件是由鼠标生成的。 <i>isRepeat</i> : 为 true – 假如事件是由一个被按住的 autoRepeating 按钮生成的；为 false – 该按钮当前不重复。
MouseEvent.DOUBLE_CLICK	已双击该按钮。仅在 <i>doubleClickEnabled</i> 属性为 true 时激发。 <i>mouseldx</i> : 用来生成该事件的鼠标光标的索引（仅适用于多鼠标光标环境）。uint 类型。仅限于 Scaleform，需要将该事件归于 MouseEventEx。 <i>buttonIdx</i> : 指明为哪个按钮生成该事件（基于零的索引）。仅限于 Scaleform，需要将该事件归于 MouseEventEx。
ButtonEvent.CLICK	单击了该按钮。 <i>controllerIdx</i> : 用来生成该事件的控制器的索引（仅适用于多鼠标光标环境）。uint 类型。 <i>isKeyboard</i> : 为 true - 假如事件是由计算机键盘 (Keyboard)/游戏键盘 (Gamepad) 生成的；为 false – 假如事件是由鼠标生成的。 <i>isRepeat</i> : 为 true – 假如事件是由一个被按住的 autoRepeating 按钮生成的；为 false – 该按钮当前不重复。
ButtonEvent.DRAG_OVER	鼠标光标拖动到按钮上方（鼠标左键被按下） <i>controllerIdx</i> : 用来生成该事件的控制器的索引（仅适用于多鼠标光标环境）。uint 类型。
ButtonEvent.DRAG_OUT	鼠标箭头从按钮拖开（鼠标左键被按下）。 <i>controllerIdx</i> : 用来生成该事件的控制器的索引（仅适用于多鼠标光标环境）。uint 类型。
ButtonEvent.RELEASE_OUTSIDE	鼠标箭头从按钮拖开鼠标左键松开。 <i>controllerIdx</i> : 用来生成该事件的控制器的索引（仅适用于多鼠标光标环境）。uint 类型。

ActionScript 代码片段用来捕获或处理这些事件。下例中展示了如何处理按钮点击事件。

```
myButton.addEventListener( ButtonEvent.PRESS, onButtonPress );
function onButtonPress( e:ButtonEvent ):void {
```

```
// Do something  
}
```

第一行代码为“`ButtonEvent.PRESS x`”事件安装事件监听器，按钮名称为‘`myButton`’，当事件触发时使其调用 `onButtonPress` 函数。相同的代码类型也可以用来处理其他的事件。向事件处理程序（示例中名为‘`e`’）提供的 `ButtonEvent` 参数包含该事件的相关信息。该参数中的事件的类型必须是同一个类，或者是在添加侦听器时使用的该类型的一个祖先。

2.1.2 CheckBox



图 10：无皮肤复选框 CheckBox

复选框 `CheckBox` (`scaleform.clik.controls.CheckBox`) 为一个按钮组件当被点击时设置为选中状态。复选框用来显示 `true/false` (布尔值) 的变化。与 `ToggleButton` 功能类似，但是隐藏设置 `Toggle` 属性。

2.1.2.1 用户交互

使用鼠标或者任何相关键盘控制器点击 `CheckBox` 组件可以使其为选中或未选中。在其他方面，复选框行为与按钮相同。

2.1.2.2 组件设置

使用 CLIK `CheckBox` 类的动画剪辑 `MovieClip` 必须具备下面所列的子单元。也列出了可选元素：

- **`textField`**: (可选) `TextField` 类型，按钮标签。
- **`focusIndicator`**: (可选) `MovieClip` 类型，一个独立的 `MovieClip` 用来显示焦点状态。如果被使用，该 `MovieClip` 必须有两个名字为“`show`”和“`hide`”的帧。

2.1.2.3 状态

根据 `Toggle` 属性，复选框 `CheckBox` 需要另外的关键帧集来表示选择状态。这些状态包括：

- **`up`** 或默认状态：

- 当鼠标箭头在组件上方或者获得焦点时为 **over** 状态;
- 当按钮被点击时候为 **down** 状态;
- disabled** 状态;
- selected_up** 或者默认状态;
- 当鼠标箭头位于组件上方或获得焦点时为 **selected_over** 状态;
- 当按钮被按下时为 **selected_down** 状态;
- selected_disabled** 状态;

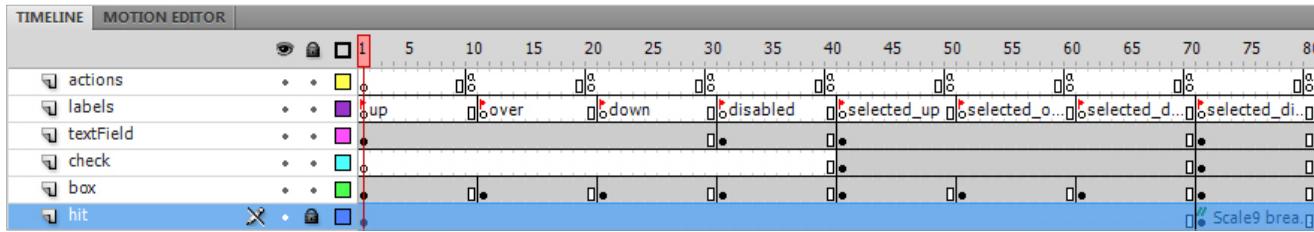
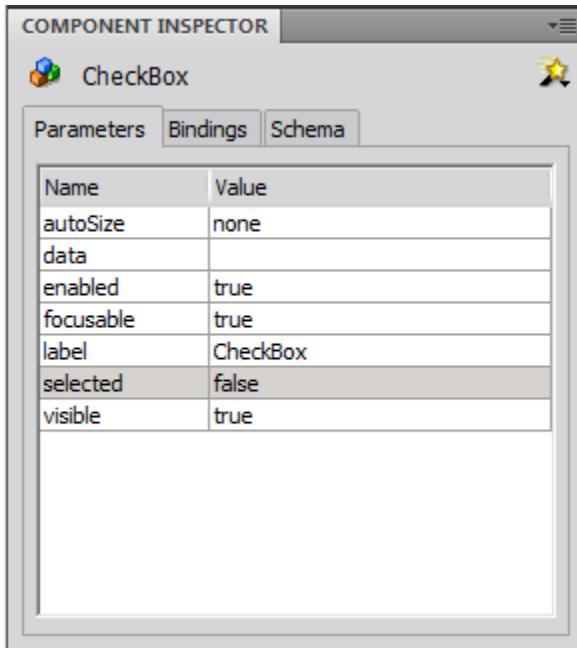


图 11: CheckBox 时间轴

这里为复选框 CheckBox 所需要的最少关键帧设置。按钮 Button 组件支持状态和关键帧的扩展设置，与复选框组件相同，在[CLIK 按钮入门](#)文档中有详细描述。

2.1.2.4 属性检查



由于从控制按钮继承而来，复选框 CheckBox 包含了与按钮相同的检查属性，具有 autoRepeat 属性和 Toggle 属性。

autoSize	确定按钮是否进行缩放以适应其所包含的文本以及已调整大小的按钮将向哪个方向对齐。将 <code>autoSize</code> 属性设置为 <code>autoSize=TextFieldAutoSize.NONE</code> 将会使当前大小保持不变。
Data	与该按钮相关的数据。在 <code>ButtonGroup</code> 中使用按钮时此属性尤其有用。
Enabled	如果设置为 <code>false</code> , 就禁用该按钮。已禁用的组件将不再收到用户输入。
focusable	启用/禁用对于组件的焦点管理。将 <code>focusable</code> (可聚焦) 属性设置为 <code>false</code> 将会取消对 <code>tab</code> 键、方向键以及基于鼠标的焦点更改的支持。
Label	设置按钮标签
selected	设置该按钮的选定状态。按钮可以有两组鼠标状态: 已选定状态和未选定状态。当一个按钮的 <code>toggle</code> (切换) 属性为 <code>true</code> 时, 点击该按钮时就会更改已选定的状态, 不过, 即使 <code>toggle</code> 属性为 <code>false</code> , 也可以使用 ActionScript 设置已选定状态。
Visible	如果设置为 <code>false</code> 隐藏按钮。

2.1.2.5 事件

所有事件回调均会收到单个 `Event` (事件) 或 `Event` 子类参数, 该参数包含有相应事件的相关信息。下列属性是所有事件所共有的。

- **type:** 事件类型。
- **target:** 事件目标。
- **currentTarget:** 通过一个事件侦听器积极处理 `Event` 对象的对象。
- **eventPhase:** 事件流中的当前阶段。 (`EventPhase.CAPTURING_PHASE`、
`EventPhase.AT_TARGET`、`EventPhase.BUBBLING_PHASE`)。
- **bubbles:** 指明事件是不是起泡事件。
- **cancelable:** 指明与事件关联的行为是否可以避免

复选框 `CheckBox` 组件产生的事件列表如下。事件旁边的属性列表为通用属性的补充内容。

ComponentEvent.SHOW	运行时可视属性已设置为 <code>true</code>
ComponentEvent.HIDE	运行时可视属性已设置为 <code>false</code>
ComponentEvent.STATE_CHANGE	组件的状态已更改。
FocusHandlerEvent.FOCUS_IN	组件已收到焦点。
FocusHandlerEvent.FOCUS_OUT	组件已失去焦点。
Event.SELECT	选定的属性已发生变化。
MouseEvent.ROLL_OVER	鼠标光标滑过该按钮。 <i>mousidx</i> : 用来生成该事件的鼠标光标的索引 (仅适用于多鼠标光标环境)。 <code>uint</code> 类型。仅限于 <code>Scaleform</code> , 需要将该事件归于 <code>MouseEventEx</code> 。

	<i>buttonIdx</i> : 指明为哪个按钮生成该事件（基于零的索引）。仅限于 Scaleform，需要将该事件归于 MouseEventEx。
MouseEvent.ROLL_OUT	鼠标光标已滑离该按钮。 <i>mouselidx</i> : 用来生成该事件的鼠标光标的索引（仅适用于多鼠标光标环境）。uint 类型。仅限于 Scaleform，需要将该事件归于 MouseEventEx。
	<i>buttonIdx</i> : 指明为哪个按钮生成该事件（基于零的索引）。仅限于 Scaleform，需要将该事件归于 MouseEventEx。
ButtonEvent.PRESS	已按该按钮。 <i>controllerIdx</i> : 用来生成该事件的控制器的索引（仅适用于多鼠标光标环境）。uint 类型。 <i>isKeyboard</i> : 为 true - 假如事件是由计算机键盘 (Keyboard)/游戏键盘 (Gamepad) 生成的；为 false – 假如事件是由鼠标生成的。 <i>isRepeat</i> : 为 true – 假如事件是由一个被按住的 autoRepeating 按钮生成的；为 false – 该按钮当前不重复。
MouseEvent.DOUBLE_CLICK	已双击该按钮。仅在 <i>doubleClickEnabled</i> 属性为 true 时激发。 <i>mouselidx</i> : 用来生成该事件的鼠标光标的索引（仅适用于多鼠标光标环境）。uint 类型。仅限于 Scaleform，需要将该事件归于 MouseEventEx。
	<i>buttonIdx</i> : 指明为哪个按钮生成该事件（基于零的索引）。仅限于 Scaleform，需要将该事件归于 MouseEventEx。
ButtonEvent.CLICK	单击了该按钮。 <i>controllerIdx</i> : 用来生成该事件的控制器的索引（仅适用于多鼠标光标环境）。uint 类型。 <i>isKeyboard</i> : 为 true - 假如事件是由计算机键盘 (Keyboard)/游戏键盘 (Gamepad) 生成的；为 false – 假如事件是由鼠标生成的。 <i>isRepeat</i> : 为 true – 假如事件是由一个被按住的 autoRepeating 按钮生成的；为 false – 该按钮当前不重复。
ButtonEvent.DRAG_OVER	鼠标光标拖动到按钮上方（鼠标左键被按下） <i>controllerIdx</i> : 用来生成该事件的控制器的索引（仅适用于多鼠标光标环境）。uint 类型。
ButtonEvent.DRAG_OUT	鼠标箭头从按钮拖开（鼠标左键被按下）。 <i>controllerIdx</i> : 用来生成该事件的控制器的索引（仅适用于多鼠标光标环境）。uint 类型。
ButtonEvent.RELEASE_OUTSIDE	鼠标箭头从按钮拖开鼠标左键松开。 <i>controllerIdx</i> : 用来生成该事件的控制器的索引（仅适用于多鼠标光标环境）。uint 类型。

下例中代码展示了如何处理复选框 CheckBox 的 Toggle 动作：

```
myCheckBox.addEventListener(Event.SELECT, onCheckBoxToggle);
function onCheckBoxToggle(e:Event):void {
    // Do something
}
```

2.1.3 Label



图 12: 未绘制皮肤的标签

CLIK 标签 Label 组件(scaleform.clik.controls.Label)为一个不可编辑的标准 textField 组件，由 MovieClip 符号进行围绕，具有一些附加的便利特性。在本质上，标签 Label 支持与标准 textField 相同的属性和行为，但是，有一些常用的特性为该组件本身所特有。如果用户需要直接改变其属性，支持访问标签实际上的 textField 区域。在某些情况下，如一些下面内容将会描述的内容，开发者可能会使用 textField 替代标签组件。

由于标签 Label 为一个 MovieClip 符号，可以用图形元素进行修饰，而这在标准的 textField 无法做到。作为一个符号，不需要如 textField 实例一样对每一个进行配置。标签 Label 还提供了 disabled 状态在时间轴可以被定义。然而，用标准的 textField 来模拟这些功能需要很多复杂的 AS2 代码。

标签 Label 组件默认强制使用，这意味着在运行时在场景中改变一个标签 Label 实例大小不会显示出效果，开发者应该在大多数情况下使用 textField 来提到 Label 标签。通常，文本元素不需要经常被重用，则 textField 比起标签 Label 使用更加简单。

2.1.3.1 用户交互

在标签 Label 内无用户交互。

2.1.3.2 组件设置

使用 CLIK Label 类的 MovieClip 必须拥有下列命名的子元素。标注了对应的可选元素：

- **textField:** TextField 类型， Label 标签文本

2.1.3.3 状态

CLIK Label 组件支持基于标准属性的两种状态:

- **default** 或者 enabled 状态;
- **disabled** 状态

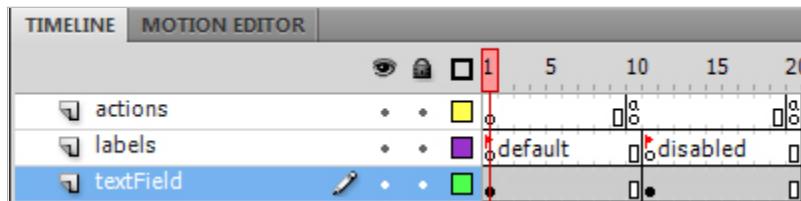
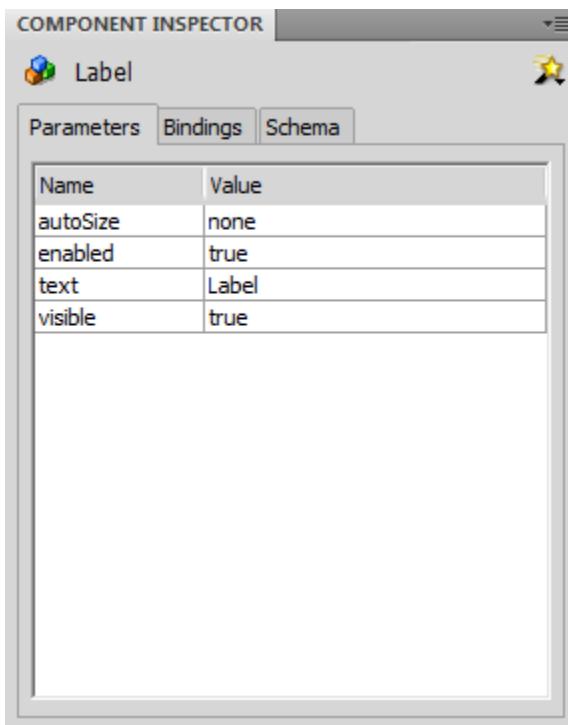


图 13: Label 时间轴

2.1.3.4 检查属性



标签 Label 的检查属性如下:

text	设置标签文本
visible	如果设置为 false, 就隐藏组件。
enabled	如果设置为 false, 就禁用该按钮。
autoSize	确定按钮是否进行缩放以适应其所包含的文本以及已调整大小的按钮将向哪个方向对齐。将 autoSize 属性设置为 autoSize=TextFieldAutoSize.NONE 将会使其当前大小

保持不变。

2.1.3.5 事件

所有事件回调均会收到单个 Event (事件) 或 Event 子类参数，该参数包含有相应事件的相关信息。下列属性是所有事件所共有的。

- **type:** 事件类型。
- **target:** 事件目标。
- **currentTarget:** 通过一个事件侦听器积极处理 Event 对象的对象。
- **eventPhase:** 事件流中的当前阶段。（EventPhase.CAPTURING_PHASE、EventPhase.AT_TARGET、EventPhase.BUBBLING_PHASE）。
- **bubbles:** 指明事件是不是起泡事件。
- **cancelable:** 指明与事件关联的行为是否可以避免

由标签 Label 组件产生的事件列表如下。事件旁边的属性列表为通用属性的补充内容。

ComponentEvent.SHOW	运行时组件可视属性已设置为 true
ComponentEvent.HIDE	运行时组件可视属性已设置为 false
ComponentEvent.STATE_CHANGE	组件的状态已更改。

2.1.4 TextInput

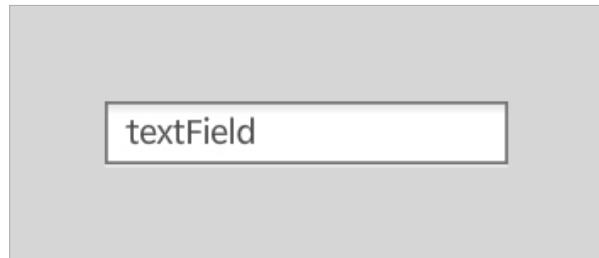


图 14: 无皮肤 TextInput.

文本输入 TextInput (scaleform.clik.controls.TextInput) 为一个可编辑的 textField 组件，用来捕获用户文本输入。与标签 Label 类似，该组件仅仅为一个标准 textField 的外框，因此支持 textField 的功能，如密码模式、最大字符数和 HTML 文本。只有一部分属性为该组件本身所有，其他的属性可以直接进入 TextInput 的 textField 实例进行更改。

`textField` 组件应该用于输入，因为无需编辑文本可以使用 `Label` 标签来显示。与 `Label` 标签类似，开发者可能会根据自身需求用标准的 `textFields` 代替 `TextInput` 组件使用。但是，当开发复杂 UI 界面时，特别是在 PC 应用中，`TextInput` 组件提供了基于标准 `textField` 的有价值的功能扩展。

作为特殊属性，`TextInput` 支持焦点和 `disabled` 状态，这在标准 `textField` 中很难实现。根据独立的焦点状态，`TextInput` 支持自定义焦点指示器，这在标准 `TextInput` 也不包含。复杂的 AS2 代码需要改变标准 `TextInput` 的外观类型，而 `TextInput` 的外观类型可以在时间轴上简单得进行配置。`TextInput` 检查属性为不熟悉 Flash Studio 的设计师和编程人员提供了一种简单的工作流程。开发者可以简便得监听 `TextInput` 触发的事件以创建自定义行为。

`TextInput` 同时也支持 `textField` 提供的标准选择和剪切、拷贝和粘贴功能，包括了多行 HTML 格式文本。默认情况下，快捷键命令为选择 (Shift+Arrows)、剪切 (Shift+Delete)、拷贝 (Ctrl+Insert)、和粘贴 (Shift+Insert)。

“文本输入”可支持鼠标滚动和滑出事件。可通过对这一特殊的 `actAsButton` 属性进行设置，来提供能够执行两种鼠标事件的两个额外关键帧。这些帧被命名为“over”和“out”，分别代表滚动和滑出状态。如果设置了 `actAsButton` 模式，并且“over”/“out”帧不存在，那么“文本输入”将会按照“默认值”关键帧执行这两种事件。请注意，这些帧不会通过预设的“文本输入”组件而出现。开发商会根据具体要求对他们进行添加。

当用户未设定或输入值时，该组件还支持默认显示的文本。可将默认文本属性设置为任何字符串。默认文本的主题（颜色和样式）为浅灰(0xAAAAAA)，斜体。可通过向“默认文本格式”属性分配一个新的“文本格式”对象的方法对样式进行自定义。

2.1.4.1 用户交互

点击 `TextInput` 使其获得焦点，则在 `textField` 中出现一个箭头。当箭头显现时，用户能够通过键盘或类似控制设备输入字符。按下坐右方向键可以移动箭头。当箭头已经位于 `textField` 的左边缘，使用左方向键则焦点将转移到左边的控制部件。使用右方向键也如此。

2.1.4.2 组件设置

使用 CLIK `TextInput` 类的 MovieClip 必须具备下面所列的子单元。也列出了可选元素：

- **`textField`:** `textField` 类型

2.1.4.3 States 状态

CLIK TextInput 组件支持三种状态，均基于焦点和 disabled 属性：

- **default** 或 enabled 状态；
- **focused** 状态，通常为 textField 周围突出显示的边框；
- **disabled** 状态。

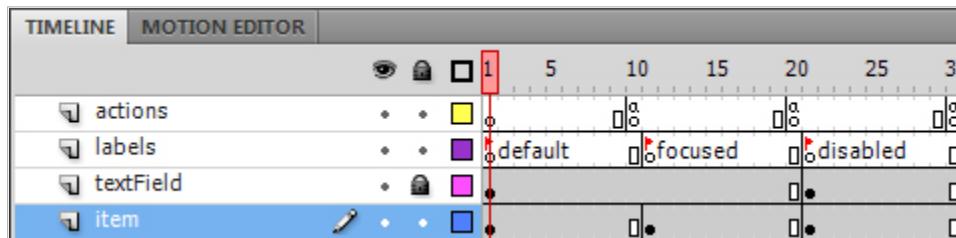
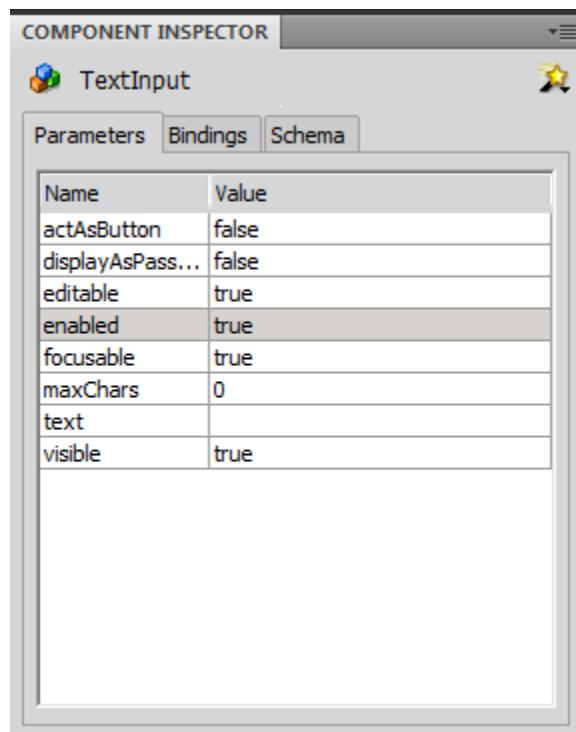


图 15: TextInput 时间轴

2.1.4.4 检查属性



文本输入 TextInput 组件的检查属性如下所示：

actAsButton	如果为“真”，则“文本输入”的行为在未选中状态下类似一个按钮，并支持滚动和滑出状态。一旦按下鼠标或 tab 键，“文本输入”将会转为正常模式，直至退出选中状态。
--------------------	--

displayAsPassword	如果是 true，就把 textField 设置为显示 '*' 字符，而不是真正字符。textField 的值将会是用户输入的真正的字符，由文本属性返回。
editable	如果设置为 false，就使 TextInput 变为不可编辑。
enabled	如果设置为 false，就禁用组件。
focusable	启用/禁用对于组件的焦点管理。将 focusable (可聚焦) 属性设置为 false 将会取消对 tab 键、方向键以及基于鼠标的焦点更改的支持。
maxChars	一个大于零的数字，作为 textField 中可以输入的最多字符数。
text	设置 textField 的初始文本。
visible	如果设置为 false 则隐藏组件

2.1.4.5 事件

所有事件回调均会收到单个 Event (事件) 或 Event 子类参数，该参数包含有相应事件的相关信息。下列属性是所有事件所共有的。

- **type:** 事件类型。
- **target:** 事件目标。
- **currentTarget:** 通过一个事件侦听器积极处理 Event 对象的对象。
- **eventPhase:** 事件流中的当前阶段。（EventPhase.CAPTURING_PHASE、EventPhase.AT_TARGET、EventPhase.BUBBLING_PHASE）。
- **bubbles:** 指明事件是不是起泡事件。
- **cancelable:** 指明与事件关联的行为是否可以避免

文本输入 TextInput 组件产生的事件列表如下。事件旁边的属性列表为通用属性的补充内容。

ComponentEvent.SHOW	visible (可见) 属性已在运行时设置为 true。
ComponentEvent.HIDE	visible 属性已在运行时设置为 false。
ComponentEvent.STATE_CHANGE	组件的状态已更改。
FocusHandlerEvent.FOCUS_IN	组件已收到焦点。
FocusHandlerEvent.FOCUS_OUT	组件已失去焦点。
Event.SELECT	选定的属性已发生变化。
MouseEvent.ROLL_OVER	鼠标光标滑过该按钮。 <i>mouseldx</i> : 用来生成该事件的鼠标光标的索引（仅适用于多鼠标光标环境）。uint 类型。仅限于 Scaleform，需要将该事件归于 MouseEventEx。 <i>buttonIdx</i> : 指明为哪个按钮生成该事件（基于零的索引）。仅限于 Scaleform，需要将该事件归于 MouseEventEx。
MouseEvent.ROLL_OUT	鼠标光标已滑离该按钮。 <i>mouseldx</i> : 用来生成该事件的鼠标光标的索引（仅适用于多鼠标光标环境）。uint 类型。仅限于 Scaleform，需要将该事件归于

MouseEventEx。

buttonIdx: 指明为哪个按钮生成该事件（基于零的索引）。仅限于 Scaleform，需要将该事件归于 MouseEventEx。

以下代码为揭示如何监听 textField 内容变化的实例：

```
myTextInput.addEventListener(Event.CHANGE, onTextChange);
function onTextChange(e:Event):void {
    trace("Latest text: " + e.target.text);
}
```

2.1.5 TextArea

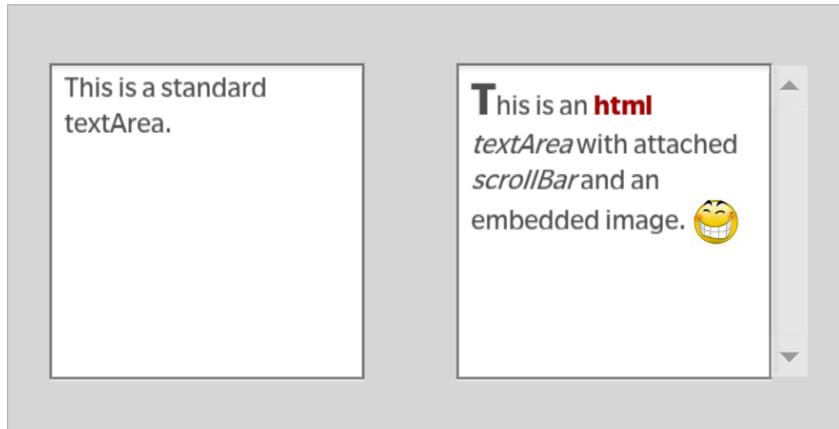


图 16: 无皮肤 TextArea.

文本区域 TextArea (scaleform.clik.controls.TextArea) 从 CLIK TextInput 继承而来，共享相同功能、属性和状态，但是具有一个可选滚动条 ScrollBar 用于多行可编辑滚动文本输入。请参考“文本输入”描述，学习更多的有关“文本输入”和“文本区域”共享的特殊功能。

类似于 Label 和 TextInput，TextArea 也为一个标准的多行 textField 的外框，因此支持 textField 的属性和行为，如 HTML 文本、文字边框、选择、剪切、拷贝、粘贴。开发者可以简单得用一个标准的 textField 替代 TextArea，但是，强烈建议使用该组件，因为其具有扩展功能、状态、检查属性和事件。

尽管标准 textField 能够用于 ScrollIndicator 或者 ScrollBar，TextArea 提供了与这些组件紧凑的组合功能。与标准的 textField 不同，TextArea 能够在使用键盘或类似控制器使其获得焦点时进行滚动，甚至在不可编辑时也如此。由于滚动组件不能获得焦点，TextArea 能够展现更多有没的焦点图型状态，能够在获得焦点的时候装饰自身和滚动组件。

2.1.5.1 用户交互

点击 `TextArea` 使其获得焦点，则在 `textField` 中出现一个箭头。当箭头显现时，用户能够通过键盘或类似控制设备输入字符。按下左右方向键可以移动箭头。当箭头已经位于 `textField` 的边缘，使用方向键则焦点将转移到相邻的控制部件。

2.1.5.2 组件设置

一个使用 CLIK `TextArea` 类的 MovieClip 必须具备下面所列的子单元。也列出了可选元素：

- **`textField`**: `TextField` 类型

2.1.5.3 状态

与上级组件 `TextInput` 类似，`TextArea` 组件支持三种状态，以焦点和 `disabled` 属性为基础。

- **`default`** 或 `enabled` 状态；
- **`focused`** 状态，通常为 `textField` 周围突出显示的边框；
- **`disabled`** 状态；

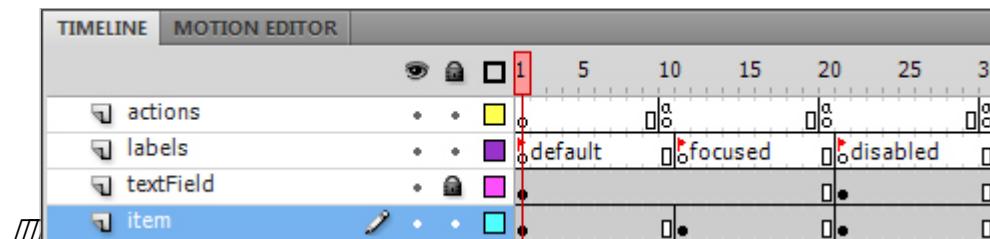
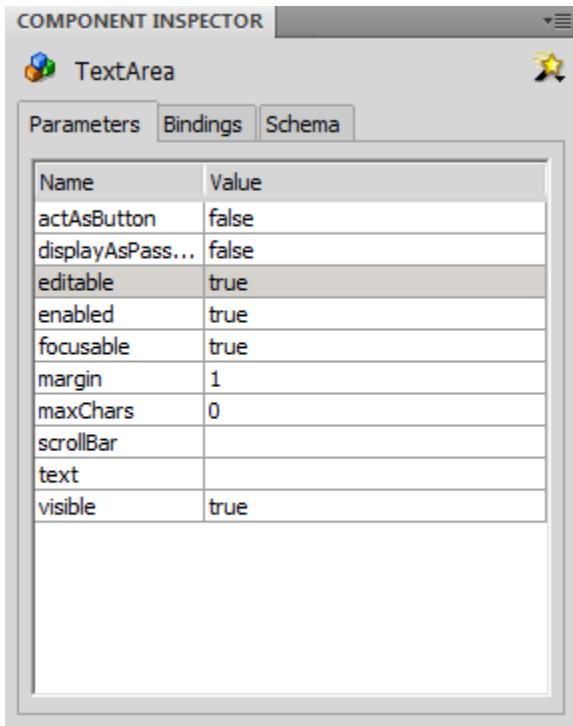


图 17: `TextArea` 时间轴

2.1.5.4 检查属性



TextArea 组件检查属性与 TextInput 类似，具有一对辅助和冗长的密码特性。辅助特性与 CLIK ScrollBar 组件有关，将在 2.4 节描述：

actAsButton	如果为 true，则 TextInput 的行为将会与一个未聚焦的按钮相似，并支持 rollOver 和 rollOut 状态。一旦通过按鼠标或 tab 键进行了聚焦，TextInput 就恢复到其正常模式，直到失去焦点。
displayAsPassword	如果是 true，就把 textField 设置为显示 '*' 字符，而不是真正字符。textField 的值将会是用户输入的真正的字符，由文本属性返回。
editable	如果设置为 false，就使 TextInput 变为不可编辑。
enabled	如果设置为 false，就禁用组件。
focusable	启用/禁用对于组件的焦点管理。将 focusable（可聚焦）属性设置为 false 将会取消对 tab 键、方向键以及基于鼠标的焦点更改的支持。
maxChars	一个大于零的数字限制可以输入 textField 的字符的数量。
scrollBar	CLIK ScrollBar 组件使用的实例名，或者一个到 ScrollBar 符号的链接 ID（本例中由 TextArea 创建一个实例）。
scrollPolicy	当设置为“auto”时，滚动条将只显示是否有足够的文本可以需要滚动。如果设置为“on”则 ScrollBar 将长期显示，如果设置为“off”则不显示，该属性只影响分配一个 ScrollBar 的组件（参考 ScrollBar 特性）
text	设置 textField 的初始文本。
visible	如果设置为 false 则隐藏组件

2.1.5.5 事件

所有事件回调均会收到单个 Event (事件) 或 Event 子类参数，该参数包含有相应事件的相关信息。下列属性是所有事件所共有的。

- **type**: 事件类型。
- **target**: 事件目标。
- **currentTarget**: 通过一个事件监听器积极处理 Event 对象的对象。
- **eventPhase**: 事件流中的当前阶段。（EventPhase.CAPTURING_PHASE、EventPhase.AT_TARGET、EventPhase.BUBBLING_PHASE）。
- **bubbles**: 指明事件是不是起泡事件。
- **cancelable**: 指明与事件关联的行为是否可以避免

文本区域 TextArea 组件产生的事件列表如下所示。事件旁列出的属性为通用属性的补充。

ComponentEvent.SHOW	visible (可见) 属性已在运行时设置为 true。
ComponentEvent.HIDE	visible 属性已在运行时设置为 false。
FocusHandlerEvent.FOCUS_IN	组件已收到焦点。
Event.CHANGE	textField 内容已改变。
Event.SCROLL	textField 内容已改变
ComponentEvent.STATE_CHANGE	在文本区域滚动
MouseEvent.ROLL_OVER	鼠标光标滑过该按钮。 <i>mouseldx</i> : 用来生成该事件的鼠标光标的索引（仅适用于多鼠标光标环境）。uint 类型。仅限于 Scaleform，需要将该事件归于 MouseEventEx。 <i>buttonIdx</i> : 指明为哪个按钮生成该事件（基于零的索引）。仅限于 Scaleform，需要将该事件归于 MouseEventEx。
MouseEvent.ROLL_OUT	鼠标光标已滑离该按钮。 <i>mouseldx</i> : 用来生成该事件的鼠标光标的索引（仅适用于多鼠标光标环境）。uint 类型。仅限于 Scaleform，需要将该事件归于 MouseEventEx。 <i>buttonIdx</i> : 指明为哪个按钮生成该事件（基于零的索引）。仅限于 Scaleform，需要将该事件归于 MouseEventEx。

以下例子展示了如何监听 TextArea 滚动事件：

```
myTextArea.addEventListener(Event.SCROLL, onTextScroll);
function onTextScroll(e:Event):void {
    // Do something
}
```

2.2 分组类型

分组类型包括 `RadioButton`、`ButtonGroup` 和 `ButtonBar` 组件。`ButtonGroup` 为一个管理类型具有特别的逻辑来维护按钮 `Button` 的分组。不具有可视外观不存在于场景中。但是，`ButtonBar` 存在于场景中也用来维护按钮分组。`RadioButton` 为一个特殊的按钮可以自动与同类组件分组到 `ButtonGroup` 中。

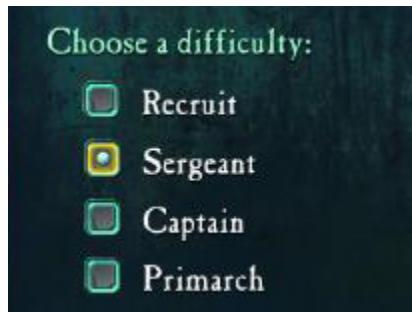


图 18:来自 *Dawn of War II* 的选择按钮组

2.2.1 RadioButton



图 19:无皮肤的选择按钮 RadioButton

选择按钮 `RadioButton` (`scaleform.clik.controls.RadioButton`) 为一个按钮组件，通常属于一个集用来显示和改变一个值。在这个集中只能选择一个选择按钮，点击集中另外一个选择按钮将选中一个新的组件，之前被选中的组件将失去被选择状态。

CLIK 选择按钮与复选框 `CheckBox` 组件非常类似，共享功能、状态和行为。主要的区别为选择按钮支持分组属性，可以指派一个自定义按钮组 `ButtonGroup`（见下节）。选择按钮不需要固定设置为选中属性，因为选中属性由按钮组实例进行管理。

2.2.1.1 用户交互

使用鼠标或类似控制器点击未选中的选择按钮组件将其选中。如果选择按钮为被选中状态，另外一同属一个按钮组的选择按钮被点击，则先前选中的选择按钮将失去选中状态。其他方面，选择按钮行为与按钮相同。

2.2.1.2 组件设置

使用 CLIK RadioButton 类的 MovieClip 必须具备下面所列的子单元。也列出了可选元素：

- **textField:** (可选) TextField 类型，按钮标签。
- **focusIndicator:** (可选 MovieClip 类型，一个独立的 MovieClip 用来显示焦点状态。如果被使用，该 MovieClip 必须具有两个帧名为：“show”和“hide”。

2.2.1.3 状态

由于选择按钮可以在选中和未选中状态间转换，与复选框 CheckBox 类似，需要至少以下几种状态：

- **up** 或默认状态；
- 当鼠标箭头在组件上方或者获得焦点时为 **over** 状态；
- 当按钮被点击时候为 **down** 状态；
- **disabled** 状态；
- **selected_up** 或者默认状态；
- 当鼠标箭头位于组件上方或获得焦点时为 **selected_over** 状态；
- 当按钮被按下时为 **selected_down** 状态；
- **selected_disabled** 状态；

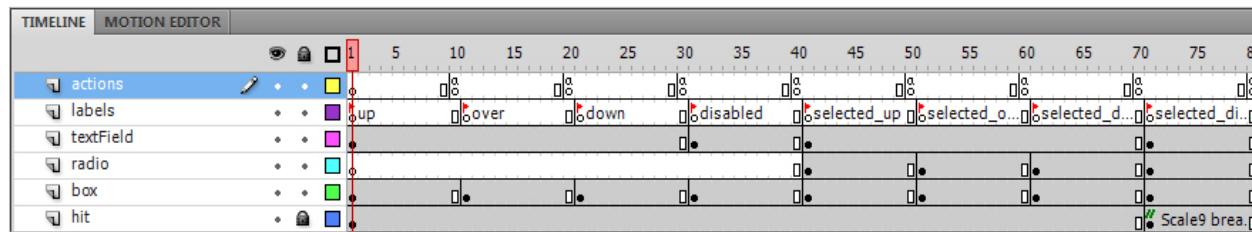
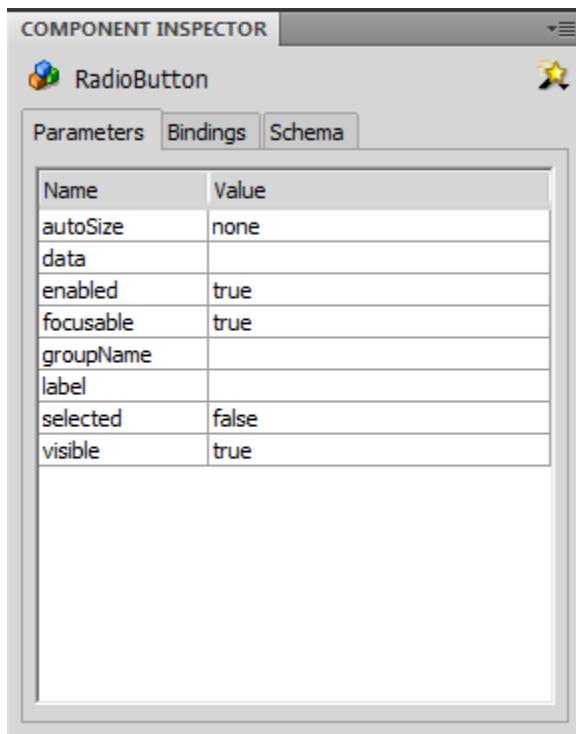


图 20:选择按钮 RadioButton 时间轴

这里为选择按钮 RadioButton 所需要的最少关键帧设置。按钮 Button 组件支持状态和关键帧的扩展设置，与选择按钮 RadioButton 组件相同，在 [CLIK 按钮入门](#) 文档中有详细描述。

2.2.1.4 检查属性



由于从按钮 Button 控制按钮继承而来，选择按钮 RadioButton 包含了与按钮相同的检查属性，具有 disableFocus 属性和 Toggle 属性。

autoSize	确定按钮是否进行缩放以适应其所包含的文本以及已调整大小的按钮将向哪个方向对齐。将 autoSize 属性设置为 autoSize=TextFieldAutoSize.NONE 将会使其当前大小保持不变。
data	与该按钮相关的数据。在 ButtonGroup 中使用按钮时此属性尤其有用。
enabled	如果设置为 false，就禁用该按钮。禁用的组件将不再收到用户输入。
focusable	启用/禁用对于组件的焦点管理。将 focusable (可聚焦) 属性设置为 false 将会取消对 tab 键、方向键以及基于鼠标的焦点更改的支持。
groupName	ButtonGroup 实例的一个名称，该实例本身存在，或者应由 RadioButton 自动创建。如果是由 RadioButton 创建，新的 ButtonGroup 将会存在于 RadioButton 的容器内部。例如，如果 RadioButton 存在于 _root 之内，则会在 _root 中创建其 ButtonGroup 对象。使用同一个组的所有 RadioButton 都将属于一个集。
label	设置该按钮的标签。
selected	设置该按钮的选定状态。按钮可以有两组鼠标状态：已选定状态和未选定状态。当一个按钮的 toggle (切换) 属性为 true 时，点击该按钮时就会更改已选定的状态，不过，即使 toggle 属性为 false，也可以使用 ActionScript 设置已选定状态。

2.2.1.5 事件

所有事件回调均会收到单个 Event (事件) 或 Event 子类参数，该参数包含有相应事件的相关信息。下列属性是所有事件所共有的。

- **type**: 事件类型。
- **target**: 事件目标。
- **currentTarget**: 通过一个事件监听器积极处理 Event 对象的对象。
- **eventPhase**: 事件流中的当前阶段。（EventPhase.CAPTURING_PHASE、EventPhase.AT_TARGET、EventPhase.BUBBLING_PHASE）。
- **bubbles**: 指明事件是不是起泡事件。
- **cancelable**: 指明与事件关联的行为是否可以避免

选择按钮 RadioButton 组件产生的事件列表如下所示。事件旁列出的属性为通用属性的补充。

ComponentEvent.SHOW	运行时可视属性已设置为 true
ComponentEvent.HIDE	运行时可视属性已设置为 false
ComponentEvent.STATE_CHANGE	组件的状态已更改。
FocusHandlerEvent.FOCUS_IN	组件已收到焦点。
FocusHandlerEvent.FOCUS_OUT	组件已失去焦点。
Event.SELECT	选定的属性已发生变化。
MouseEvent.ROLL_OVER	鼠标光标滑过该按钮。 <i>mouselidx</i> : 用来生成该事件的鼠标光标的索引（仅适用于多鼠标光标环境）。uint 类型。仅限于 Scaleform，需要将该事件归于 MouseEventEx。 <i>buttonIdx</i> : 指明为哪个按钮生成该事件（基于零的索引）。仅限于 Scaleform，需要将该事件归于 MouseEventEx。
MouseEvent.ROLL_OUT	鼠标光标已滑离该按钮。 <i>mouselidx</i> : 用来生成该事件的鼠标光标的索引（仅适用于多鼠标光标环境）。uint 类型。仅限于 Scaleform，需要将该事件归于 MouseEventEx。 <i>buttonIdx</i> : 指明为哪个按钮生成该事件（基于零的索引）。仅限于 Scaleform，需要将该事件归于 MouseEventEx。
ButtonEvent.PRESS	已按该按钮。 <i>controllerIdx</i> : 用来生成该事件的控制器的索引（仅适用于多鼠标光标环境）。uint 类型。 <i>isKeyboard</i> : 为 true - 假如事件是由计算机键盘 (Keyboard)/游戏键盘 (Gamepad) 生成的；为 false – 假如事件是由鼠标生成的。 <i>isRepeat</i> : 为 true – 假如事件是由一个被按住的 autoRepeating 按钮生成的；为 false – 该按钮当前不重复。
MouseEvent.DOUBLE_CLICK	已双击该按钮。仅在 <i>doubleClickEnabled</i> 属性为 true 时激发。

	<p><i>mouselidx</i>: 用来生成该事件的鼠标光标的索引（仅适用于多鼠标光标环境）。uint 类型。仅限于 Scaleform，需要将该事件归于 MouseEventEx。</p> <p><i>buttonIdx</i>: 指明为哪个按钮生成该事件（基于零的索引）。仅限于 Scaleform，需要将该事件归于 MouseEventEx。</p>
ButtonEvent.CLICK	<p>单击了该按钮。</p> <p><i>controllerIdx</i>: 用来生成该事件的控制器的索引（仅适用于多鼠标光标环境）。uint 类型。</p> <p><i>isKeyboard</i>: 为 true - 假如事件是由计算机键盘 (Keyboard)/游戏键盘 (Gamepad) 生成的；为 false – 假如事件是由鼠标生成的。</p> <p><i>isRepeat</i>: 为 true – 假如事件是由一个被按住的 autoRepeating 按钮生成的；为 false – 该按钮当前不重复。</p>
ButtonEvent.DRAG_OVER	<p>鼠标光标拖动到按钮上方（鼠标左键被按下）</p> <p><i>controllerIdx</i>: 用来生成该事件的控制器的索引（仅适用于多鼠标光标环境）。uint 类型。</p>
ButtonEvent.DRAG_OUT	<p>鼠标箭头从按钮拖开（鼠标左键被按下）。</p> <p><i>controllerIdx</i>: 用来生成该事件的控制器的索引（仅适用于多鼠标光标环境）。uint 类型。</p>
ButtonEvent.RELEASE_OUTSIDE	<p>鼠标箭头从按钮拖开鼠标左键松开。</p> <p><i>controllerIdx</i>: 用来生成该事件的控制器的索引（仅适用于多鼠标光标环境）。uint 类型。</p>

以下例子显示如何处理选择按钮 RadioButton 的选中状态：

```
myRadio.addEventListener(Event.SELECT, onRadioToggle);
function onRadioToggle(e:Event):void {
    // Do something
}
```

2.2.2 ButtonGroup

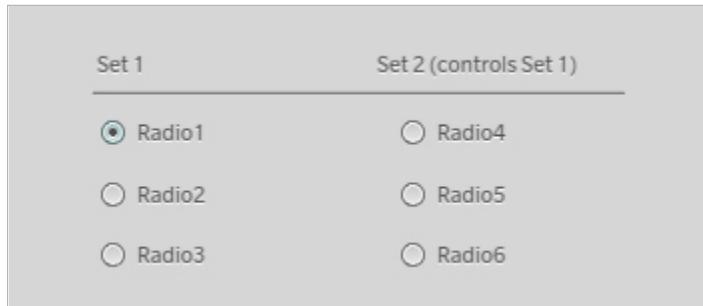


图 21: 无皮肤按钮分组 **ButtonGroup**.

CLIK ButtonGroup (`scaleform.clik.controls.ButtonGroup`)本身不是一个组件，但是有重要作用用于管理按钮集。可以使在集中的一个按钮被选中，确保其余的未选中。如果用户选择集中的另一个按钮，则当前选中按钮将变为未选中。任何从 CLIK 按钮组件继承的组件（如复选框 **CheckBox** 和选择按钮 **RadioButton**）能够使用按钮分组 **ButtonGroup** 实例。

2.2.2.1 用户交互

按钮组不具有用户交互功能，因为不是可视组件。但是，在下属的选择按钮 **RadioButton** 被点击时起到间接的作用。

2.2.2.2 组件设置

使用 CLIK 按钮组 **ButtonGroup** 类的 **MovieClip** 不需要任何子单元，因为不具有可视化外观。

2.2.2.3 状态

按钮组 **ButtonGroup** 在场景中不具有可视外观。因此无关联状态。

2.2.2.4 检查属性

按钮组 **ButtonGroup** 在场景中不具有可视外观。因此无需检查属性。

2.2.2.5 事件

所有事件回调均会收到单个 **Event**（事件）或 **Event** 子类参数，该参数包含有相应事件的相关信息。下列属性是所有事件所共有的。

- **type:** 事件类型。
- **target:** 事件目标。

- **currentTarget**: 通过一个事件侦听器积极处理 Event 对象的对象。
- **eventPhase**: 事件流中的当前阶段。（EventPhase.CAPTURING_PHASE、EventPhase.AT_TARGET、EventPhase.BUBBLING_PHASE）。
- **bubbles**: 指明事件是不是起泡事件。
- **cancelable**: 指明与事件关联的行为是否可以避免

ButtonGroup 组件产生的事件列表如下所示。事件旁列出的属性为通用属性的补充。

Event.CHANGE	在组中选择一个新的按钮
ButtonEvent.CLICK	组中的按钮被点击 <i>target</i> : 选择按钮, CLIK 按钮 Button 类型

下例展示了如何判断按钮组 ButtonGroup 中哪个按钮被选中:

```
myGroup.addEventListener(Event.CHANGE, onNewSelection);
function onNewSelection(e:Event):void {
    if (e.item == myRadio) {
        // Do something
    }
}
```

2.2.3 ButtonBar

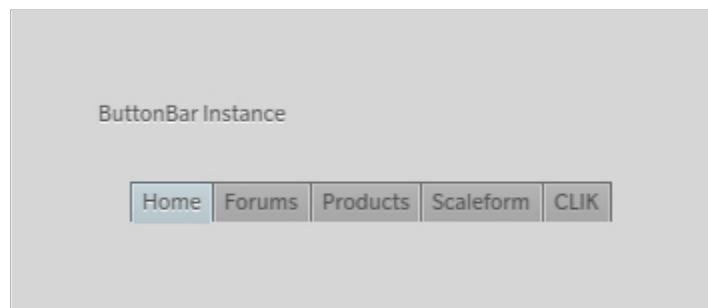


图 22: 无皮肤按钮栏 ButtonBar.

按钮栏 ButtonBar (scaleform.clik.controls.ButtonBar)与按钮组 ButtonGroup 类似，尽管具有可视化外观。也可以基于数据源 dataProvider (参见[编程细节](#)描述 dataProvider 章节) 动态创建按钮实例。按钮栏可用与创建动态 tab 栏 UI 元素。

```
buttonBar.dataProvider = ["item1", "item2", "item3", "item4", "item5"];
```

2.2.3.1 用户交互

按钮栏与按钮组 **ButtonGroup** 具有相同的行为，也不能与用户直接交互，按钮栏也由按钮点击事件简介影响。

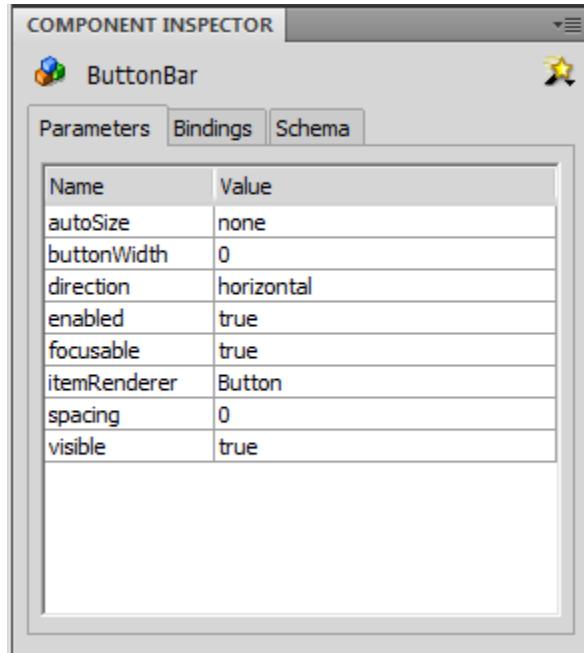
2.2.3.2 组件设置

使用 CLIK 按钮栏 **ButtonBar** 类的视频剪辑 **MovieClip** 不需要任何子单元，因为不具有可视化外观。

2.2.3.3 状态

CLIK 按钮栏不具有任何可视状态，因为其管理按钮组件只用来显示组的状态。

2.2.3.4 检查属性



尽管按钮栏组件没有内容（只为 Flash IDE 场景中的一个简单小圆圈），但包含几个检查属性。主要由按钮栏 **ButtonBar** 创建的按钮实例分布设置决定。

autoSize	如果设置为 true ，则重新调整 Button （按钮）实例的大小以适应显示的标签。
Direction	按钮放置。“横向”(Horizontal) 将把 Button 实例并排放置，而“纵向”(Vertical) 则把这些实例依次堆叠在上面。
buttonWidth	给所有 Button 实例设置一个通用的宽度。如果将 autoSize 设为 true ，则忽略此属性。
Enabled	如果设置为 false ，就禁用该按钮。禁用的组件将不再收到用户输入。
Focusable	启用/禁用对于组件的焦点管理。将 focusable （可聚焦）属性设置为 false 将会取消

	对 tab 键、方向键以及基于鼠标的焦点更改的支持。
itemRenderer	按钮组件符号的链接 ID，该符号将根据按钮栏的数据分配需求进行实例化
Spacing	按钮实例间隔，只影响当前方位（见 <i>direction</i> 属性）
Visible	如果设置为 false，就隐藏 ButtonBar。

2.2.3.5 事件

所有事件回调均会收到单个 Event (事件) 或 Event 子类参数，该参数包含有相应事件的相关信息。下列属性是所有事件所共有的。

- **type:** 事件类型。
- **target:** 事件目标。
- **currentTarget:** 通过一个事件侦听器积极处理 Event 对象的对象。
- **eventPhase:** 事件流中的当前阶段。（EventPhase.CAPTURING_PHASE、EventPhase.AT_TARGET、EventPhase.BUBBLING_PHASE）。
- **bubbles:** 指明事件是不是起泡事件。
- **cancelable:** 指明与事件关联的行为是否可以避免

ButtonBar 组件产生的事件列表如下所示。事件旁列出的属性为通用属性的补充。

ComponentEvent.SHOW	运行时可视属性已设置为 true
ComponentEvent.HIDE	运行时可视属性已设置为 false
FocusHandlerEvent.FOCUS_IN	该按钮收到了焦点。
FocusHandlerEvent.FOCUS_OUT	该按钮失去了焦点。
IndexEvent.INDEX_CHANGE	<p>已经选定该组中的一个新按钮。</p> <p><i>index:</i> ButtonBar 的选定索引。int 类型。值 -1 (如果当前未选定任何项目) 到按钮数量减去 1。</p> <p><i>lastIndex:</i> ButtonBar 的以前选定的索引。int 类型。值 -1 (如果以前未选定任何项目) 到按钮数量减去 1。</p> <p><i>data:</i> 选定的dataProvider 的数据值。AS3 Object 类型。</p>
ButtonBarEvent.BUTTON_SELECT	<p>已经选定该组中的一个新按钮。</p> <p><i>index:</i> ButtonBar 的选定索引。Int 类型。值 -1 (如果当前未选定任何项目) 到按钮数量减去 1。</p> <p><i>renderer:</i> ButotnBar 内现在选定的 Button 实例。Button 类型。</p>

以下例子展示了当在按钮栏中的按钮实例被点击后如何进行事件监听。

```
myBar.addEventListener(ButtonBarEvent.BUTTON_SELECT, onItemClick);
function onItemClick(e:ButtonBarEvent) {
    processData(e.renderer.data);
    // Do something
}
```

2.3 滚动类型

滚动类型包括 ScrollIndicator、ScrollBar 和 Slider。滚动指示器 ScrollIndicator 无需交互用来显示目标对象组件的滚动位置，而滚动条 ScrollBar 支持用户交互来改变滚动位置。滚动条由四个按钮组成：翻页按钮、轨道、向上方向箭和向下方向键。滑动条 Slider 与滚动条类似，但是只包含一个交互的翻页按钮和轨道，不根据目标组件的单元数来改变翻页按钮大小。

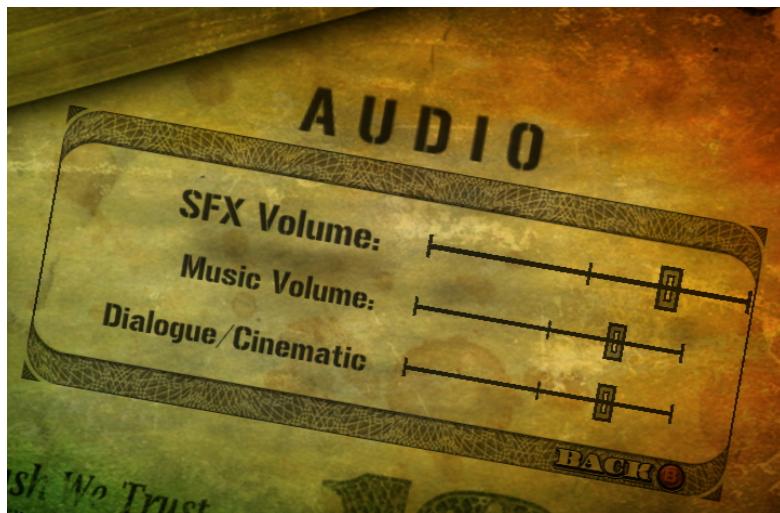


图 23: 来自 **Mercenaries 2** 的音频菜单滑动条实例

2.3.1 ScrollIndicator



图 24: 无皮肤 ScrollIndicator.

CLIK 滚动指示器 ScrollIndicator (`scaleform.clik.controls.ScrollIndicator`) 显示了其它组件的滚动位置，如多行文本区域 `textField`。可以在文本区域用来自动显示滚动位置。所有基于列表的组件，包括文本区 `TextArea`，具有滚动条属性可以由滚动指示器或滚动条（见下节）实例或链接 ID 进行显示。

2.3.1.1 用户交互

滚动指示器与具有用户交互功能，因为只用来显示。

2.3.1.2 组件设置

一个使用 CLIK ScrollIndicator 类的 MovieClip 必须具备下面所列的子单元。也列出了可选元素：

- **thumb**: CLIK 按钮 Button 类型，滚动指示块。
- **track**: 动画剪辑 MovieClip 类型，滚动指示轨道，轨道的边界决定了滚动块可以移动的位置。

2.3.1.3 状态

滚动指示器没有显性状态，使用子单元的状态：滚动块和轨道按钮组件。

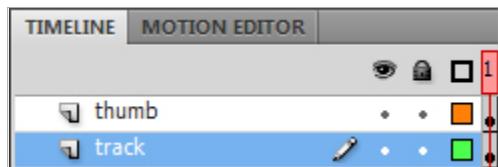
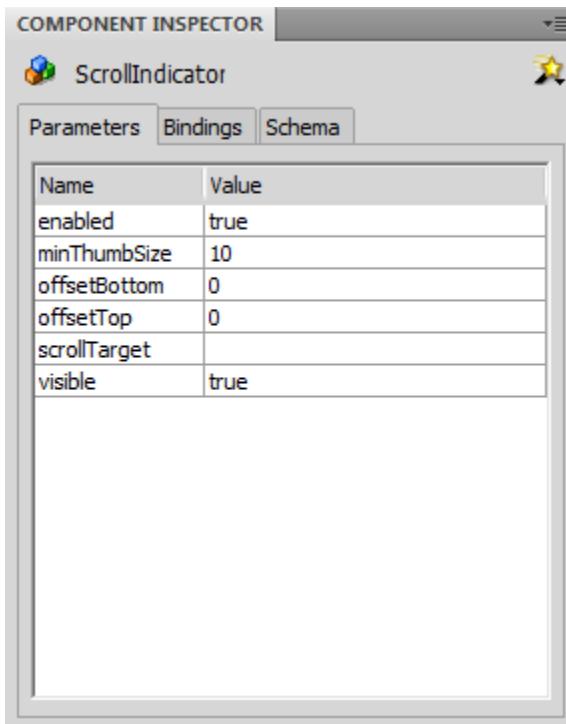


图 25:滚动指示器 ScrollIndicator 时间轴

2.3.1.4 检查属性



滚动指示器的检查属性如下所示：

Enabled

如果设置为 `false`，就禁用组件。禁用的组件将不再收到用户输入。

offsetTop	拖动点顶部偏移。正值将使拖动点向顶部位置的更高处移动。
offsetBottom	拖动点底部偏移。正值将使拖动点向底部位置的更低处移动。
scrollTarget	设置一个文本区域 <code>TextArea</code> 或者常用多行文本域 <code>textField</code> 作为滚动目标自动响应滚动事件。非文本域 <code>textField</code> 类型需要手动更新滚动指示器 <code>ScrollIndicator</code> 属性。
Visible	如果设置为 <code>false</code> 则隐藏组件

2.3.1.5 事件

所有事件回调均会收到单个 `Event` (事件) 或 `Event` 子类参数, 该参数包含有相应事件的相关信息。下列属性是所有事件所共有的。

- **type:** 事件类型。
- **target:** 事件目标。
- **currentTarget:** 通过一个事件侦听器积极处理 `Event` 对象的对象。
- **eventPhase:** 事件流中的当前阶段。（`EventPhase.CAPTURING_PHASE`、`EventPhase.AT_TARGET`、`EventPhase.BUBBLING_PHASE`）。
- **bubbles:** 指明事件是不是起泡事件。
- **cancelable:** 指明与事件关联的行为是否可以避免

`ScrollIndicator` 组件产生的事件列表如下所示。事件旁列出的属性为通用属性的补充。

ComponentEvent.SHOW	<code>visible</code> (可见) 属性已在运行时设置为 <code>true</code> 。
ComponentEvent.HIDE	<code>visible</code> 属性已在运行时设置为 <code>false</code> 。
Event.SCROLL	已经滚动 <code>ScrollIndicator</code> 。

下例显示如何监听滚动事件：

```
mySI.addEventListener(Event.SCROLL, onTextScroll);
function onTextScroll(e:Event) {
    trace("mySI.position: " + e.target.position);
    // Do something
}
```

2.3.2 ScrollBar



图 26: 无皮肤滚动条 ScrollBar.

CLIK 滚动条 ScrollBar(`scaleform.clik.controls.ScrollBar`)显示和控制其它组件的滚动位置。通过拖动滚动块按钮增加交互功能到滚动指示器，同时还包括向上和向下方向键和一个可以点击的轨道。

2.3.2.1 用户交互

滚动条 ScrollBar 上的滚动块可以由鼠标或类似控制器进行控制，可以在滚动条轨迹边界之内拖动。当鼠标光标位于滚动条上方时移动鼠标滚轮可以进行滚动操作。点击向上方向键可以向上滚动，点击向下方键可以向下滚动。点击轨道有两种行为：滚动块继续沿点击点位置滚动，或者立即跳转到点击点位置并设为拖动状态。轨道模式由 `trackMode` 检查属性决定（见检查属性小节）。忽略轨道模式 `trackMode` 设置，按下(Shift)键并点击轨道将立即将滚动快移动到鼠标光标处并设为拖动状。

2.3.2.2 组件设置

一个使用 Clik ScrollBar 类的 MovieClip 必须具备下面所列的子单元。也列出了可选元素：

- **upArrow:** Clik 按钮类型，向上滚动按钮；通常位于滚动条顶部。
- **downArrow:** Clik 按钮类型，向下滚动按钮；通常位于滚动条底部。
- **thumb:** Clik 按钮类型，滚动条上的滚动块。
- **track:** Clik 按钮类型，滚动条轨道，其边界决定了滚动块的活动方位。

2.3.2.3 状态

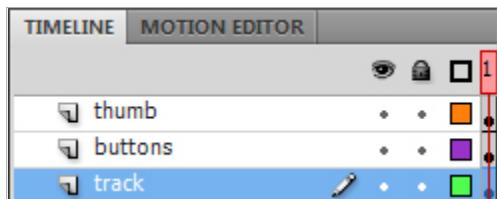
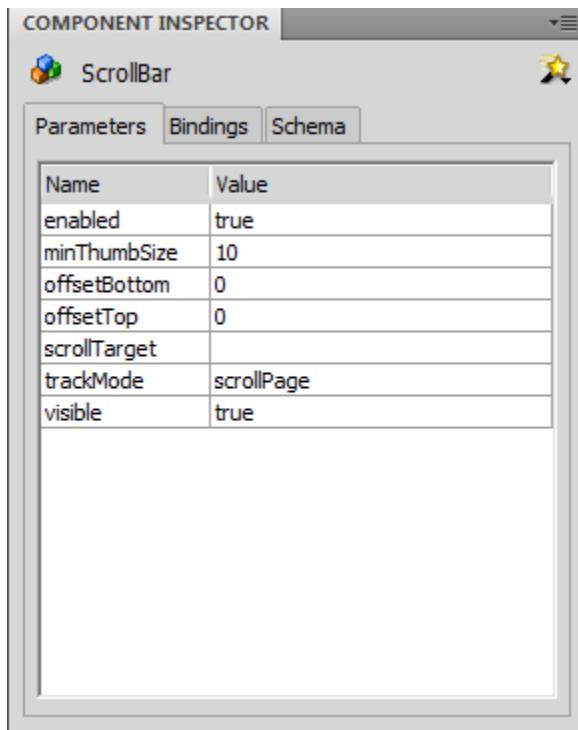


图 27: 滚动条 ScrollBar 时间轴

滚动条 ScrollBar，与滚动指示器 ScrollIndicator 类似，不需要显性状态。使用子单元的状态包括：滚动块、向上按钮、向下按钮和轨道按钮组件。

2.3.2.4 检查属性



滚动条检查属性和滚动指示器类似，只增加一条：

Enabled	如果设置为 <code>false</code> ，就禁用组件。禁用的组件将不再收到用户输入。
offsetTop	拖动点顶部偏移。正值将使拖动点向顶部位置的更高处移动。
offsetBottom	拖动点底部偏移。正值将使拖动点向底部位置的更低处移动。
scrollTarget	响应滚动事件时设置文本区 <code>TextArea</code> 或常用多行文本域 <code>textField</code> 的滚动目标位置
trackMode	当用户用鼠标点击滚动条，滚动页 <code>scrollPage</code> 设置将导致滚动块翻页知道释放鼠标。 <code>scrollToCursor</code> 设置可以使滚动块立即跳转到鼠标光标所在位置并转入拖动模式直到释放鼠标。
Visible	如果设置为 <code>false</code> 则隐藏组件

2.3.2.5 事件

所有事件回调均会收到单个 `Event`（事件）或 `Event` 子类参数，该参数包含有相应事件的相关信息。下列属性是所有事件所共有的。

- **type:** 事件类型。
- **target:** 事件目标。
- **currentTarget:** 通过一个事件侦听器积极处理 `Event` 对象的对象。

- **eventPhase**: 事件流中的当前阶段。（EventPhase.CAPTURING_PHASE、EventPhase.AT_TARGET、EventPhase.BUBBLING_PHASE）。
- **bubbles**: 指明事件是不是起泡事件。
- **cancelable**: 指明与事件关联的行为是否可以避免

ScrollBar 组件产生的事件列表如下所示。事件旁列出的属性为通用属性的补充。

ComponentEvent.SHOW	visible (可见) 属性已在运行时设置为 true。
----------------------------	-------------------------------

ComponentEvent.HIDE	visible 属性已在运行时设置为 false。
----------------------------	---------------------------

Event.SCROLL	已经滚动 ScrollIndicator。
---------------------	-----------------------

下列代码指示如何监听滚动事件：

```
mySB.addEventListener(Event.SCROLL, onTextScroll);
function onTextScroll(e:Event) {
    trace("mySB.position: " + e.target.position);
    // Do something
}
```

2.3.3 Slider



图 28: 无皮肤滑动条 Slider.

滑动条 Slider(scaleform.clik.controls.Slider)显示了一个范围内的数值，用滚动块来表示值，通过拖动来改变值。

2.3.3.1 用户交互

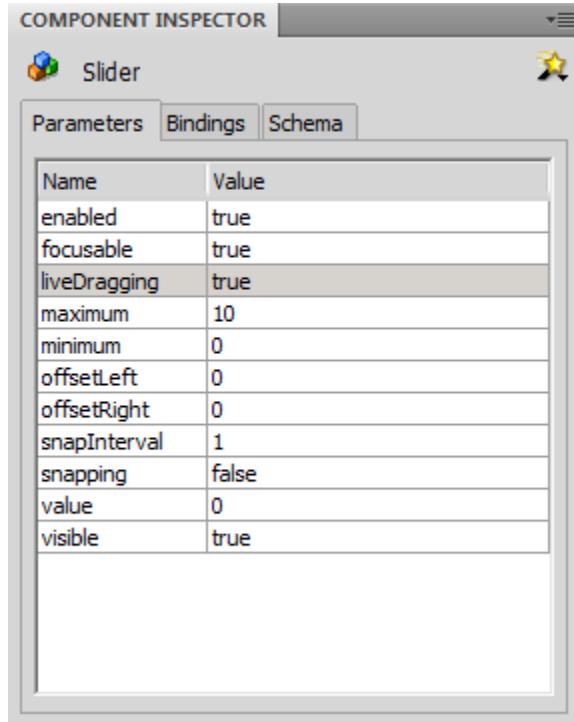
滑动块可以被鼠标或类似控制器在滑动轨道边界内拖动。在轨道上点击可以立即移动滚动块到鼠标光标所在位置并设置为拖动状。当获得焦点时，左右方向键在对应的方向移动滚动块，而 home 和 end 键可以将滚动块移动到滚动条的开始和末尾处。

2.3.3.2 组件设置

使用 CLIK Slider 类的 MovieClip 必须具备下面所列的子单元。也列出了可选元素：

- **thumb:** CLIK 按钮类型，滑动快。
- **Track:** CLIK 按钮类型，滑动轨道边界决定了滑动块可以移动位置。

2.3.3.3 状态



与滚动指示器和滚动条类似，滑动条没有显性状态，使用子单元的状态包括：滚动块和轨道按钮组件。

Enabled	如果设置为 <code>false</code> , 就禁用组件。禁用的组件将不再收到用户输入。
Focusable	启用/禁用对于组件的焦点管理。将 <code>focusable</code> (可聚焦) 属性设置为 <code>false</code> 将会取消对 <code>tab</code> 键、方向键以及基于鼠标的焦点更改的支持。
Value	滑动条 Slider 显示的数值
minimum	滑动条范围内的最小值
maximum	滑动条范围内的最大值
snapping	如果设置为 <code>true</code> , 滚动快将按照多被间隔进行移动
snapInterval	滚动间隔决定滚动块跳转间隔函数, 设置为 <code>false</code> 时该值无效
liveDragging	如果设置为 <code>true</code> , 拖动滚动块时则滑动条 Slider 将产生一个改变事件, 滑动条只在拖动结束后产生事件改变
offsetLeft	拖动点向左偏移。正值将会向内挤压拖动点。

offsetRight	拖动点向右偏移。正值将会向内挤压拖动点。
Visible	如果设置为 <code>false</code> 则隐藏组件

2.3.3.4 事件

所有事件回调均会收到单个 `Event` (事件) 或 `Event` 子类参数, 该参数包含有相应事件的相关信息。下列属性是所有事件所共有的。

- **`type`**: 事件类型。
- **`target`**: 事件目标。
- **`currentTarget`**: 通过一个事件侦听器积极处理 `Event` 对象的对象。
- **`eventPhase`**: 事件流中的当前阶段。(`EventPhase.CAPTURING_PHASE`、`EventPhase.AT_TARGET`、`EventPhase.BUBBLING_PHASE`)。
- **`bubbles`**: 指明事件是不是起泡事件。
- **`cancelable`**: 指明与事件关联的行为是否可以避免

`Slider` 组件产生的事件列表如下所示。事件旁列出的属性为通用属性的补充。

ComponentEvent.SHOW	运行时可视属性已设置为 <code>true</code>
ComponentEvent.HIDE	运行时可视属性已设置为 <code>false</code>
ComponentEvent.STATE_CHANGE	组件的状态已更改。
FocusHandlerEvent.FOCUS_IN	组件已收到焦点。
FocusHandlerEvent.FOCUS_OUT	组件已失去焦点。
SliderEvent.VALUE_CHANGE	改变滑动条 <code>Slider</code> 's 值

下例显示了如何监听滑动条 `Slider` 值的改变:

```
mySlider.addEventListener(SliderEvent.VALUE_CHANGE, onValueChange);
function onValueChange(e:SliderEvent) {
    trace("mySlider.value: " + e.target.value);
    // Do something
}
```

2.4 列表类型

CLIK 列表类型包括 `NumericStepper`、`OptionStepper`、`ScrollingList`、`TileList` 和 `DropdownMenu` 组件。所有组件, 除了 `NumericStepper`, 都用 `DataProvider` 来管理信息并显示。`ListItemRenderer` 组件也包括在此目录中, 因为 `ScrollingList` 和 `TileList` 组件需要用来显示列表项目。

NumericStepper 和 OptionStepper 组件只用一次显示一个单元，而 ScrollingList 和 TileList 能显示多个单元。最后两个组件可以支持 ScrollIndicator 或 ScrollBar 组件。DropdownMenu 组件在静止状态显示一个单元，但是在打开时使用 ScrollingList 或 TileList 可以显示多个单元。



图 30:来自 *Mercenaries 2* 的滚动指示器滚动列表实例

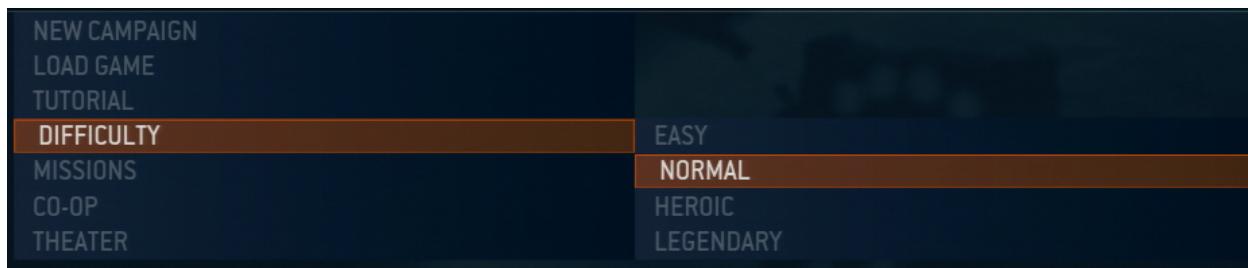


图 31:来自 *Halo Wars* 的'Difficulty Settings'下拉菜单实例

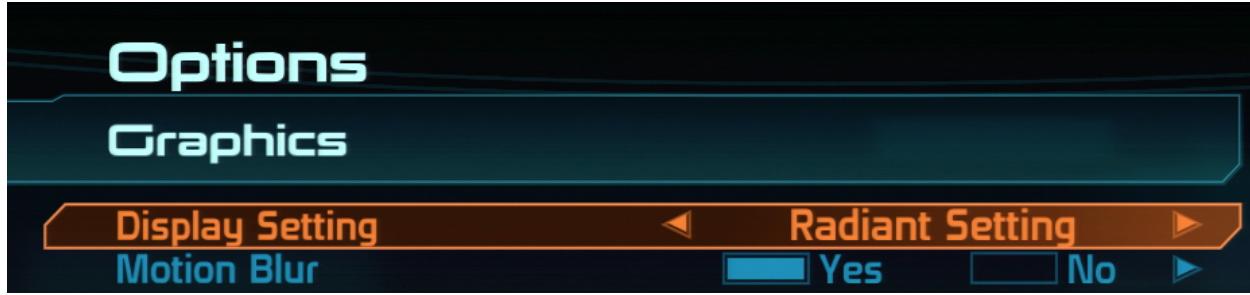


图 32:来自 *Mass Effect* 的'Display Setting'选项实例

2.4.1 NumericStepper

NumericStepper (scaleform.clik.controls.NumericStepper) 显示在分配范围内的一个数字，支持任意步长值的增加和减少。

2.4.1.1 用户交互

NumericStepper 包括两个方向键可以通过鼠标或类似控制器点击以改变数字值。当获得焦点时，数字值可以通过键盘左右方向键或类似控制器改变。这些键可以根据步长增加和减少当前值。按下(Home) 和 (End)键或类似控制器将在对应的最大值和最小值之间改变数字值。

2.4.1.2 组件设置

使用 CLIK NumericStepper 类的 MovieClip 必须具备下面所列的子单元。也列出了可选元素：

- **textField:** TextField 类型，用来显示当前值。
- **nextBtn:** CLIK 按钮 Button 类型，点击可以根据步长增加当前值。
- **prevBtn:** CLIK 按钮 Button 类型，点击可以根据步长减少当前值。

2.4.1.3 状态

NumericStepper 组件支持三种状态，基于焦点和禁止属性：

- **default** 或 enabled 状态；
- **focused** 状态，突出显示 textField 区域；
- **disabled** 状态；

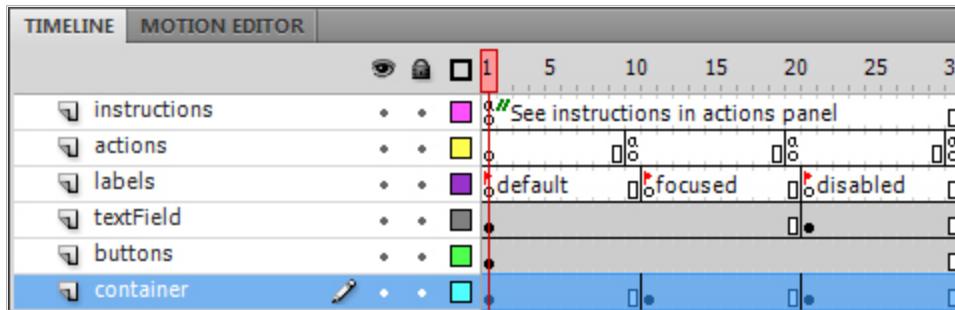
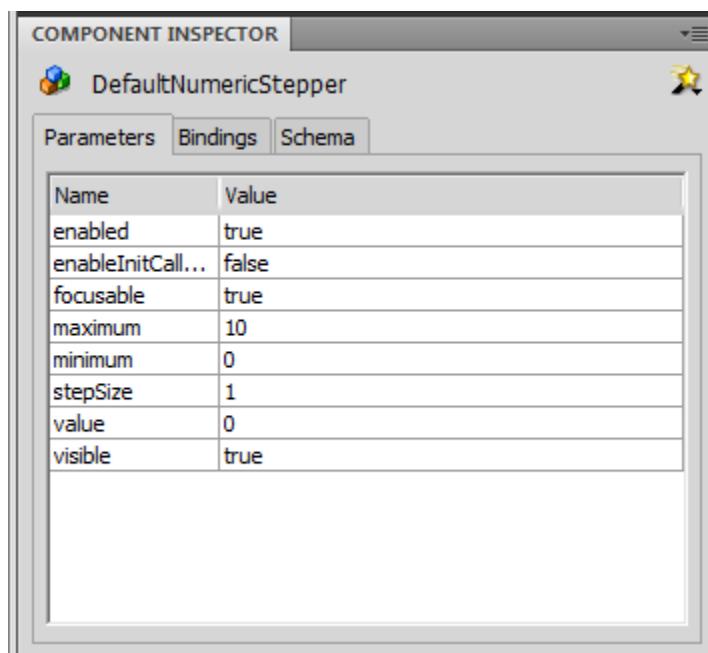


图 34: NumericStepper 时间轴

2.4.1.4 检查属性



从数字步长 NumericStepper 组件继承来的动画剪辑 MovieClip 将有以下检查属性:

Enabled	如果设置为 <code>false</code> , 就禁用组件。禁用的组件将不再收到用户输入。
Focusable	启用/禁用对于组件的焦点管理。将 <code>focusable</code> (可聚焦) 属性设置为 <code>false</code> 将会取消对 <code>tab</code> 键、方向键以及基于鼠标的焦点更改的支持。
minimum	NumericStepper 数值范围内的最小值
maximum	NumericStepper 数值范围内的最大值
stepSize	每次点击 NumericStepper 的一个按钮, 就应递增/递减多少 NumericStepper 的值。
Value	NumericStepper.显示的数值
Visible	如果设置为 <code>false</code> 则隐藏组件

2.4.1.5 事件

所有事件回调均会收到单个 Event (事件) 或 Event 子类参数，该参数包含有相应事件的相关信息。下列属性是所有事件所共有的。

- **type:** 事件类型。
- **target:** 事件目标。
- **currentTarget:** 通过一个事件侦听器积极处理 Event 对象的对象。
- **eventPhase:** 事件流中的当前阶段。（EventPhase.CAPTURING_PHASE、EventPhase.AT_TARGET、EventPhase.BUBBLING_PHASE）。
- **bubbles:** 指明事件是不是起泡事件。
- **cancelable:** 指明与事件关联的行为是否可以避免

NumericStepper 组件产生的事件列表如下所示。事件旁列出的属性为通用属性的补充。

ComponentEvent.SHOW	visible (可见) 属性已在运行时设置为 true。
ComponentEvent.HIDE	visible 属性已在运行时设置为 false。
ComponentEvent.STATE_CHANGE	组件的状态已更改。
FocusHandlerEvent.FOCUS_IN	组件已收到焦点。
FocusHandlerEvent.FOCUS_OUT	组件已失去焦点。
ComponentEvent.STATE_CHANGE	组件的状态已更改。
IndexEvent.INDEX_CHANGE	<p>NumericStepper 的值已发生变化。</p> <p><i>index:</i> NumericStepper 的选定的索引。int 类型。值 -1 (如果当前未选定任何项目) 到按钮数量减去 1。</p> <p><i>lastIndex:</i> NumericStepper 的以前选定的索引。int 类型。值 -1 (如果以前未选定任何项目) 到按钮数量减去 1。</p> <p><i>data:</i> 选定的dataProvider 的数据值。AS3 Object 类型。</p>

下例显示了对 NumericStepper 值改变的监听：

```
myNS.addEventListener(IndexEvent.INDEX_CHANGE, onValueChange);
function onValueChange(e:IndexEvent) {
    trace("myNS.value: " + e.target.value);
    // Do something
}
```

2.4.2 OptionStepper



图 35: 无皮肤 OptionStepper.

OptionStepper (`scaleform.clik.controls.OptionStepper`),与 `NumericStepper` 类似, 用来显示一个值, 但可以显示更多信息。使用数据源 `dataProvider` 实例查询当前值; 因此支持不同类型元素的任意数值。应用 `OptionStepper` 的选择项索引属性通过代码对显示值进行设置, 该索引是附在数据提供者中的一个从零开始的索引。数据源 `dataProvider` 通过代码进行指派, 如下例所示:

```
optionStepper.dataProvider = new DataProvider(["item1", "item2", "item3",
"item4"]);
```

2.4.2.1 用户交互

与 `NumericStepper` 类似, `OptionStepper` 包括两个方向键, 可以通过鼠标或类似控制器点击改变当前值。当获得焦点时, 当前值可以通过左右方向键或类似控制器进行改变。这些键按照顺序向前或者向后改变当前值。按下(Home)和(End)键或类似控制器可以将当前值改变为数据源 `dataProvider` 的第一个和最后一个单元。

2.4.2.2 组件设置

使用 CLIK `NumericStepper` 类的动画剪辑 MovieClip 必须具备下面所列的子单元。也列出了可选元素:

- **textField**: `TextField` 类型, 用来显示当前值。
- **nextBtn**: CLIK 按钮 `Button` 类型, 改变当前值为数据源 `dataProvider` 中的下一个值。
- **prevBtn**: CLIK 按钮 `Button` 类型, 当前值为数据源 `dataProvider` 中的上一个值。

2.4.2.3 状态

`OptionStepper` 组件支持三种状态, 基于焦点和禁止属性:

- **default** 或 `enabled` 状态;
- **focused** 状态, 突出显示 `textField` 区域;
- **disabled** 状态;

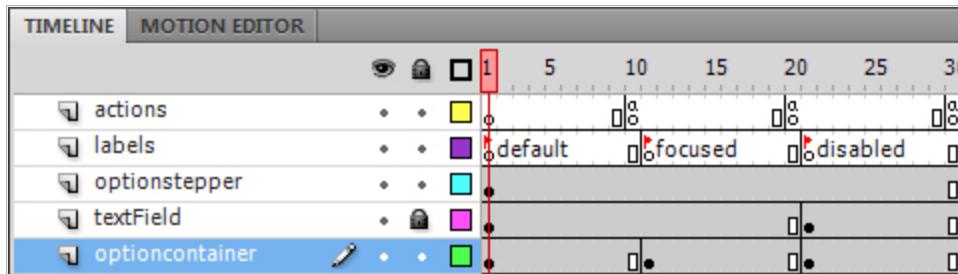
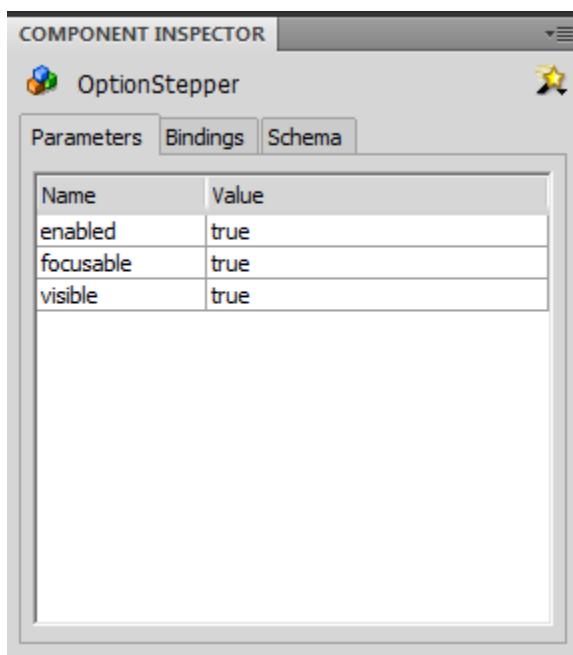


图 36: OptionStepper 时间轴

2.4.2.4 检查属性



来自选项步长 OptionStepper 组件的视频剪辑 MovieClip 具有以下检查属性。

Enabled	如果设置为 <code>false</code> , 就禁用组件。禁用的组件将不再收到用户输入。
Focusable	启用/禁用对于组件的焦点管理。将 <code>focusable</code> (可聚焦) 属性设置为 <code>false</code> 将会取消对 <code>tab</code> 键、方向键以及基于鼠标的焦点更改的支持。
Visible	如果设置为 <code>false</code> 则隐藏组件

2.4.2.5 事件

所有事件回调均会收到单个 `Event` (事件) 或 `Event` 子类参数, 该参数包含有相应事件的相关信息。下列属性是所有事件所共有的。

- **type:** 事件类型。
- **target:** 事件目标。

- **currentTarget**: 通过一个事件侦听器积极处理 Event 对象的对象。
- **eventPhase**: 事件流中的当前阶段。（EventPhase.CAPTURING_PHASE、EventPhase.AT_TARGET、EventPhase.BUBBLING_PHASE）。
- **bubbles**: 指明事件是不是起泡事件。
- **cancelable**: 指明与事件关联的行为是否可以避免

OptionStepper 组件产生的事件列表如下所示。事件旁列出的属性为通用属性的补充。

ComponentEvent.SHOW	visible (可见) 属性已在运行时设置为 true。
ComponentEvent.HIDE	visible 属性已在运行时设置为 false。
ComponentEvent.STATE_CHAN	组件的状态已更改。
GE	
FocusHandlerEvent.FOCUS_IN	组件已收到焦点。
FocusHandlerEvent.FOCUS_OU	组件已失去焦点。
T	
IndexEvent.INDEX_CHANGE	OptionStepper 的值已发生变化。 <i>index</i> : NumericStepper 的选定的索引。int 类型。值 -1 (如果当前未选定任何项目) 到按钮数量减去 1。 <i>lastIndex</i> : NumericStepper 的以前选定的索引。int 类型。值 -1 (如果以前未选定任何项目) 到按钮数量减去 1。 <i>data</i> : 选定的dataProvider 的数据值。AS3 Object 类型。

下列显示了 OptionStepper 值改变的监听:

```
myOS.addEventListener(IndexEvent.INDEX_CHANGE, onValueChange);
function onValueChange(e:IndexEvent) {
    trace("myOS.selectedItem: " + e.target.selectedItem);
    // Do something
}
```

2.4.3 ListItemRenderer



图 37:无皮肤 ListItemRenderer.

列表项渲染器 `ListItemRenderer` (`scaleform.clik.controls.ListItemRenderer`) 从 CLIK 按钮 `Button` 类继承而来，扩展了列表相关特性，在容器组件中需要用到。但是，不是作为一个单独组建设计，只与滚动列表 `ScrollingList`、标题列表 `TitleList` 和下拉菜单 `DropdownMenu` 组件共同使用。

2.4.3.1 用户交互

由于 `ListItemRenderer` 从按钮 `Button` 组件继承而来，与按钮具有相同的用户交互如使用鼠标点击。滚动鼠标光标到 `ListItemRenderer` 或将光标移开都将对组件产生影响，拖动鼠标光标效果也一样。

`ListItemRenderer` 的容器组件定义了键盘或类似控制器的交互。

2.4.3.2 组件设置

使用 CLIK `ListItemRenderert` 类的动画剪辑 `MovieClip` 必须具备下面所列的子单元。也列出了可选元素：

- **`textField`**: (可选) `TextField` 类型，列表项标签。
- **`focusIndicator`**: (可选) `MovieClip` 类型，一个独立的动画剪辑 `MovieClip` 用来显示焦点状态。如果被使用，该动画剪辑必须拥有两个帧分别命名为：“`show`” 和“`hide`”。

2.4.3.3 状态

由于可以在一个容器组件内部被选择，列表项渲染器 `ListItemRenderer` 需要关键帧为 `selected` 设置来表示其选择状态。组件状态包括：

- **`up`** 或默认状态；
- 当鼠标箭头在组件上方或者获得焦点时为 **`over`** 状态；
- 当按钮被点击时候为 **`down`** 状态；
- **`disabled`** 状态；
- **`selected_up`** 或者默认状态；
- 当鼠标箭头位于组件上方或获得焦点时为 **`selected_over`** 状态；
- 当按钮被按下时为 **`selected_down`** 状态；
- **`selected_disabled`** 状态；

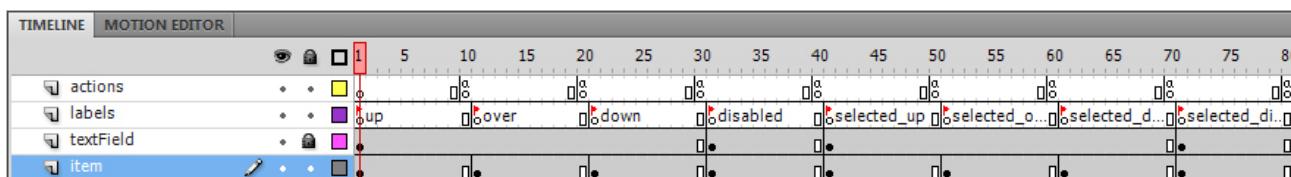
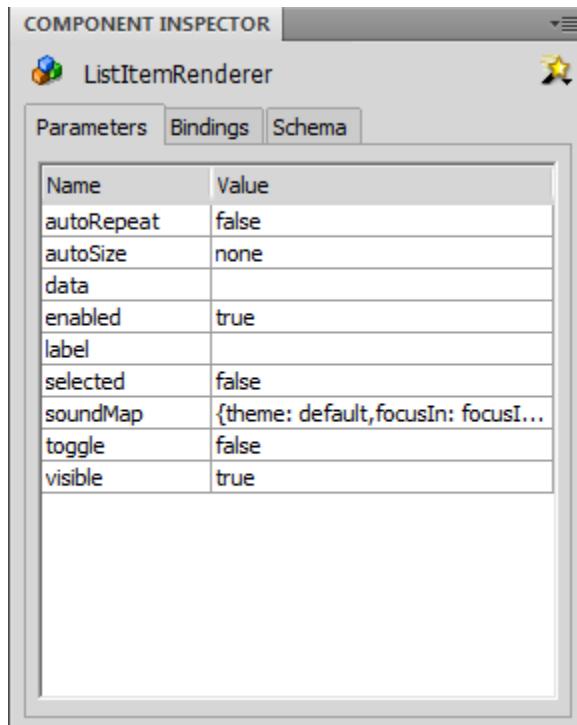


图 38: `ListItemRenderer` 时间轴

这里为选择按钮 `ListItemRenderer` 所需要的最少关键帧设置。按钮 `Button` 组件支持状态和关键帧的扩展设置，与 `ListItemRenderer` 组件相同，在 [CLIK 按钮入门](#) 文档中有详细描述。

2.4.3.4 检查属性



由于列表项渲染器 `ListItemRenderere` 由一个容器组件按控制，用户无需手动配置，值包含了按钮的一小部分检查属性。

autoRepeat	确定在按下并保持住时 <code>ListItemRender</code> 是否发出 "click" (点击) 事件。
autoSize	确定 <code>ListItemRender</code> 是否会缩放以适应其所包含的文本以及已调整大小的 <code>ListItemRender</code> 将向哪个方向对齐。将 <code>autoSize</code> 属性设置为 <code>autoSize=TextFieldAutoSize.NONE</code> 将会使其当前大小保持不变。
Data	与 <code>ListItemRender</code> 相关的数据。在一个 <code>List</code> (列表) 组件 (<code>ScrollingList / TileList</code>) 中使用 <code>ListItemRender</code> 时此属性尤其有用。
Enabled	如果设置为 <code>false</code> ，就禁用组件。禁用的组件将不再收到用户输入。
Label	<code>ListItemRenderer</code> 标签设置
selected	设置 <code>ListItemRender</code> 的选定状态。 <code>ListItemRender</code> 可以有两组鼠标状态：已选定状态和未选定状态。当一个 <code>ListItemRender</code> 的 <code>toggle</code> 属性为 <code>true</code> 时，点击该按钮时就会更改已选定的状态，不过，即使 <code>toggle</code> 属性为 <code>false</code> ，也可以使用 ActionScript 设置已选定状态。
Toggle	设置 <code>ListItemRender</code> 的 <code>toggle</code> 属性。如果设置为 <code>true</code> ，则 <code>ListItemRender</code> 的行为将会与一个切换按钮相同。

Visible	如果设置为 <code>false</code> 则隐藏组件
----------------	--------------------------------

2.4.3.5 事件

所有事件回调均会收到单个 `Event` (事件) 或 `Event` 子类参数, 该参数包含有相应事件的相关信息。下列属性是所有事件所共有的。

- **`type`**: 事件类型。
- **`target`**: 事件目标。
- **`currentTarget`**: 通过一个事件侦听器积极处理 `Event` 对象的对象。
- **`eventPhase`**: 事件流中的当前阶段。(`EventPhase.CAPTURING_PHASE`、`EventPhase.AT_TARGET`、`EventPhase.BUBBLING_PHASE`)。
- **`bubbles`**: 指明事件是不是起泡事件。
- **`cancelable`**: 指明与事件关联的行为是否可以避免

下面列出 `ListItemRenderer` 组件生成的事件。除常见属性外, 还提供了该事件旁边列出的属性。

一般情况下, 用户不要对 `ListItemRenderer`'s 发出的事件进行侦听, 而应将一个事件侦听器添加到 `List` (列表) 本身。当与其子 `ListItemRenderer` 之一发生交互时, 列表就会发出包含 `ITEM_PRESS` 和 `ITEM_CLICK` 的 `ListEvents`。这使用户可以将一个 `EventListener` 添加到针对某个鼠标点击事件 (`ListEvent.ITEM_CLICK`) 的列表, 而不是上述列表内每个 `ListItemRenderer` 上的一个 `EventListener`。

ComponentEvent.SHOW	<code>visible</code> (可见) 属性已在运行时设置为 <code>true</code> 。
ComponentEvent.HIDE	<code>visible</code> 属性已在运行时设置为 <code>false</code> 。
ComponentEvent.STATE_CHAN	组件的状态已更改。
GE	
FocusHandlerEvent.FOCUS_IN	组件已收到焦点。
FocusHandlerEvent.FOCUS_OU	组件已失去焦点。
T	
Event.SELECT	选定的属性已发生变化。
MouseEvent.ROLL_OVER	鼠标光标滑过该按钮。 <code>mouselidx</code> : 用来生成该事件的鼠标光标的索引 (仅适用于多鼠标光标环境)。uint 类型。仅限于 <code>Scaleform</code> , 需要将该事件归于 <code>MouseEventEx</code> 。 <code>buttonIdx</code> : 指明为哪个按钮生成该事件 (基于零的索引)。仅限于 <code>Scaleform</code> , 需要将该事件归于 <code>MouseEventEx</code> 。
MouseEvent.ROLL_OUT	鼠标光标已滑离该按钮。 <code>mouselidx</code> : 用来生成该事件的鼠标光标的索引 (仅适用于多鼠标光标环境)。uint 类型。仅限于 <code>Scaleform</code> , 需要将该事件归于 <code>MouseEventEx</code> 。 <code>buttonIdx</code> : 指明为哪个按钮生成该事件 (基于零的索引)。仅限于

	Scaleform, 需要将该事件归于 MouseEventEx。
ButtonEvent.PRESS	<p>已按该按钮。</p> <p><i>controllerIdx</i>: 用来生成该事件的控制器的索引（仅适用于多鼠标光标环境）。uint 类型。</p> <p><i>isKeyboard</i>: 为 true - 假如事件是由计算机键盘 (Keyboard)/游戏键盘 (Gamepad) 生成的；为 false – 假如事件是由鼠标生成的。</p> <p><i>isRepeat</i>: 为 true – 假如事件是由一个被按住的 autoRepeating 按钮生成的；为 false – 该按钮当前不重复。</p>
MouseEvent.DOUBLE_CLICK	<p>已双击该按钮。仅在 <i>doubleClickEnabled</i> 属性为 true 时激发。</p> <p><i>mouseIdx</i>: 用来生成该事件的鼠标光标的索引（仅适用于多鼠标光标环境）。uint 类型。仅限于 Scaleform, 需要将该事件归于 MouseEventEx。</p> <p><i>buttonIdx</i>: 指明为哪个按钮生成该事件（基于零的索引）。仅限于 Scaleform, 需要将该事件归于 MouseEventEx。</p>
ButtonEvent.CLICK	<p>单击了该按钮。</p> <p><i>controllerIdx</i>: 用来生成该事件的控制器的索引（仅适用于多鼠标光标环境）。uint 类型。</p> <p><i>isKeyboard</i>: 为 true - 假如事件是由计算机键盘 (Keyboard)/游戏键盘 (Gamepad) 生成的；为 false – 假如事件是由鼠标生成的。</p> <p><i>isRepeat</i>: 为 true – 假如事件是由一个被按住的 autoRepeating 按钮生成的；为 false – 该按钮当前不重复。</p>
ButtonEvent.DRAG_OVER	<p>鼠标光标拖动到按钮上方（鼠标左键被按下）</p> <p><i>controllerIdx</i>: 用来生成该事件的控制器的索引（仅适用于多鼠标光标环境）。uint 类型。</p>
ButtonEvent.DRAG_OUT	<p>鼠标箭头从按钮拖开（鼠标左键被按下）。</p> <p><i>controllerIdx</i>: 用来生成该事件的控制器的索引（仅适用于多鼠标光标环境）。uint 类型。</p>
ButtonEvent.RELEASE_OUTSIDE	<p>鼠标箭头从按钮拖开鼠标左键松开。</p> <p><i>controllerIdx</i>: 用来生成该事件的控制器的索引（仅适用于多鼠标光标环境）。uint 类型。</p>

2.4.4 ScrollingList



图 39: 无皮肤滚动列表 ScrollingList.

滚动列表 ScrollingList (scaleform.clik.controls.ScrollingList)为一个组件可以滚动其元素。可以自身展示列表项或者使用现存的场景中的列表项。可以附加滚动指示器 ScrollIndicator 或滚动条 ScrollBar 组件提供滚动反馈和控制功能。该组件可与数据源dataProvider一起使用。数据源通过代码指派，如下面实例所示：

```
scrollingList.dataProvider = new DataProvider(["item1", "item2", "item3",  
"item4"]);
```

默认情况下滚动列表 crollingList 使用列表项渲染器 ListItemRenderer 组件作为内容。因此列表项渲染器必须在 FLA 文件库中存在，除非列表项渲染器检查属性改变成另外一个组件。见检查属性小节获得更多信息。

2.4.4.1 用户交互

点击一个列表项或者一个附属的滚动条 ScrollBar 实例将焦点传递到滚动列表 ScrollingList 组件。当获得焦点，按下向上向下方向键或类似的控制器控制列表选项的滚动选择一个元素。如果没有选中元素，则默认选中顶部元素。如果光标位于滚动列表 ScrollingList 顶部则鼠标滚轮能够在列表中滚动。

在边界上的滚动行为由滚动列表 ScrollingList 的 *wrapping* 属性决定，无检查项。如果 *wrapping* 设置为“normal”，当选项达到列表开始处或者结束处则焦点将离开组件。如果 *wrapping* 设置为“wrap”，则选项将围绕开始处或结束处。如果 *wrapping* 设置为“stick”，则选项在达到数据结尾时停止，焦点不转移到相邻组件。

按下键盘(Page Up) 和 (Page down)或类似控制器将按页滚动选项，例如，列表中的可视选项数。按下(Home) 和 (End)键，或者类似控制器，将滚动列表到相应元素的开始和末尾处。与附属滚动条 ScrollBar 组件交互将按预期影响滚动列表 ScrollingList。见滚动条 ScrollBar 小节了解其用户交互功能。

开发者可以简单得将游戏控制杆映射到键盘和鼠标控制键。例如，键盘方向键通常引导控制器上的 D-Pad。这个对应关系可以使 CLIK 构建的 UI 界面工作在更加广泛的平台之上。

2.4.4.2 组件设置

滚动列表 ScrollingList 不需要任何子单元，但是，在场景上的滚动列表 ScrollingList 组件上放置一个组件或者调整组件大小时一个可视化背景就能起到辅助作用。

2.4.4.3 状态

滚动列表 ScrollingList 组件支持三种状态，基于焦点和禁止属性：

- **default** 或 enable 状态；
- **focused** 状态，通常突出显示组件边框区域；
- **disabled** 状态。

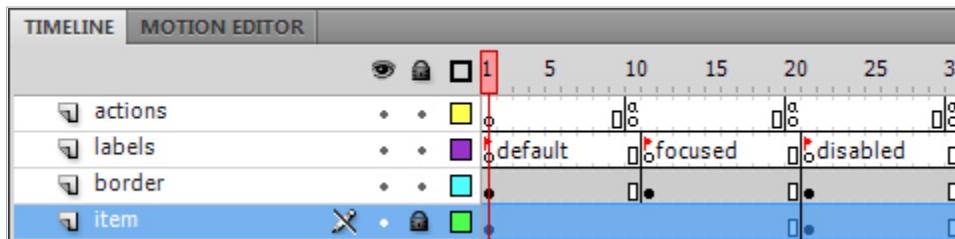
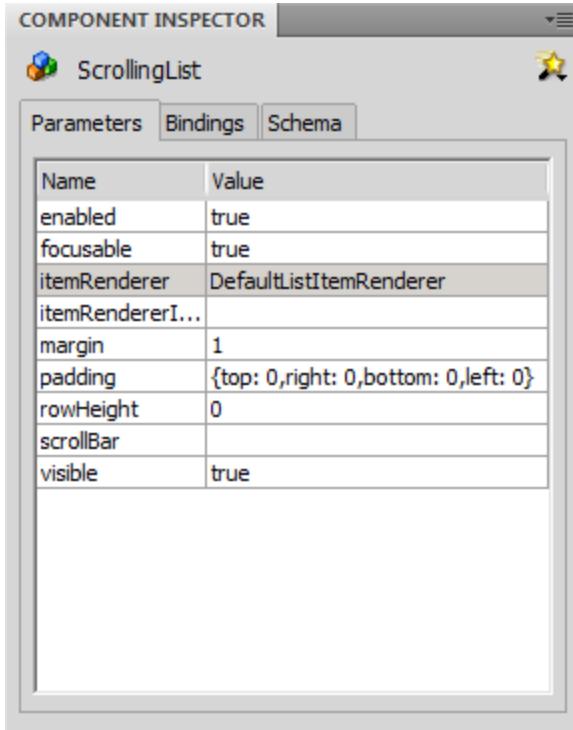


图 40:滚动列表 ScrollingList 时间轴

2.4.4.4 检查属性



来自滚动列表 ScrollingList 组件的视频剪辑 MovieClip 具有以下检查属性。

Enabled	如果设置为 <code>false</code> , 就禁用组件。禁用的组件将不再收到用户输入。
Focusable	启用/禁用对于组件的焦点管理。将 <code>focusable</code> (可聚焦) 属性设置为 <code>false</code> 将会取消对 tab 键、方向键以及基于鼠标的焦点更改的支持。
itemRenderer	<code>ListItemRenderer</code> 的符号名称。用来在内部创建列表项实例。如果设置了 <code>itemRendererInstanceName</code> 属性 (即如果使用外部 <code>ListItemRenderers</code>) , 则没有影响。
itemRendererInstanceName	列表项渲染器扩展前缀, 与滚动列表 <code>ScrollingList</code> 组件一起使用。场景中的列表项实例必须用该属性值作为前缀。如果属性设置为‘r’, 则所有本组件使用的列表项实例必须具有以下值: ‘r1’、‘r2’、‘r3’.....第一个项应该为数值 1。
Margin	内部创建的列表组件和列表项组件的页面边缘。如果设置了 <code>rendererInstanceName</code> 属性该值不产生影响。
Padding	针对列表项的额外填充 (Padding)。如果设置了 <code>itemRendererInstanceName</code> 属性 (即如果使用外部 <code>ListItemRenderer</code>) , 则此值没有影响。填充不会影响自动生成的 <code>ScrollBar</code> (滚动条) 。
rowHeight	内部创建的列表项实例的高度。如果设置了 <code>itemRendererInstanceName</code> 属性 (即如果使用外部 <code>ListItemRenderer</code>) , 则此值没有影响。
Visible	如果设置为 <code>false</code> , 就隐藏组件。这不会隐藏附加的 <code>ScrollBar</code> 或任何外部 <code>ListItemRenderer</code> 。

2.4.4.5 事件

所有事件回调均会收到单个 Event (事件) 或 Event 子类参数，该参数包含有相应事件的相关信息。下列属性是所有事件所共有的。

- **type:** 事件类型。
- **target:** 事件目标。
- **currentTarget:** 通过一个事件侦听器积极处理 Event 对象的对象。
- **eventPhase:** 事件流中的当前阶段。（EventPhase.CAPTURING_PHASE、EventPhase.AT_TARGET、EventPhase.BUBBLING_PHASE）。
- **bubbles:** 指明事件是不是起泡事件。
- **cancelable:** 指明与事件关联的行为是否可以避免

滚动列表 ScrollingList 组件产生的事件列表如下所示。事件旁列出的属性为通用属性的补充。

ComponentEvent.SHOW	visible (可见) 属性已在运行时设置为 true。
ComponentEvent.HIDE	visible 属性已在运行时设置为 false。
ComponentEvent.STATE_CHANGE	组件的状态已更改。
E	
FocusHandlerEvent.FOCUS_IN	组件已收到焦点。
FocusHandlerEvent.FOCUS_OUT	组件已失去焦点。
ListEvent.INDEX_CHANGE	选定的索引已发生变化。 <ul style="list-style-type: none">• <i>itemRenderer:</i> 双击过的列表项。 IListItemRenderer 类型。• <i>itemData:</i> 与列表项关联的数据。此值是从列表的数据Provider 中检索的。 ActionScript Object 类型。• <i>index:</i> 列表的新的选定索引。 int 类型。值 -1 (未设置选定的索引) 到列表项数量减去 1。• <i>controllerIdx:</i> 用来生成该事件的控制器/鼠标的索引 (仅适用于多鼠标光标环境)。
ListEvent.ITEM_PRESS	已按下一个列表项。 <ul style="list-style-type: none">• <i>itemRenderer:</i> 双击过的列表项。 IListItemRenderer 类型。• <i>itemData:</i> 与列表项关联的数据。此值是从列表的数据Provider 中检索的。 ActionScript Object 类型。• <i>index:</i> 列表的新的选定索引。 int 类型。值 -1 (未设置选定的索引) 到列表项数量减去 1。

	<ul style="list-style-type: none"> • <i>controllerIdx</i>: 用来生成该事件的控制器/鼠标的索引（仅适用于多鼠标光标环境）。
ListEvent.ITEM_CLICK	已点击一个列表项。 <ul style="list-style-type: none"> • <i>itemRenderer</i>: 双击过的列表项。 IListItemRenderer 类型。 • <i>itemData</i>: 与列表项关联的数据。此值是从列表的 dataProvider 中检索的。 ActionScript Object 类型。 • <i>index</i>: 列表的新的选定索引。 int 类型。值 -1 (未设置选定的索引) 到列表项数量减去 1。 • <i>controllerIdx</i>: 用来生成该事件的控制器/鼠标的索引（仅适用于多鼠标光标环境）。
ListEvent.ITEM_DOUBLE_CLICK	已双击一个列表项。 <ul style="list-style-type: none"> • <i>itemRenderer</i>: 双击过的列表项。 IListItemRenderer 类型。 • <i>itemData</i>: 与列表项关联的数据。此值是从列表的 dataProvider 中检索的。 ActionScript Object 类型。 • <i>index</i>: 列表的新的选定索引。 int 类型。值 -1 (未设置选定的索引) 到列表项数量减去 1。 • <i>controllerIdx</i>: 用来生成该事件的控制器/鼠标的索引（仅适用于多鼠标光标环境）。
ListEvent.ITEM_ROLL_OVER	鼠标光标滑过一个列表项。 <ul style="list-style-type: none"> • <i>itemRenderer</i>: 双击过的列表项。 IListItemRenderer 类型。 • <i>itemData</i>: 与列表项关联的数据。此值是从列表的 dataProvider 中检索的。 ActionScript Object 类型。 • <i>index</i>: 列表的新的选定索引。 int 类型。值 -1 (未设置选定的索引) 到列表项数量减去 1。 • <i>controllerIdx</i>: 用来生成该事件的控制器/鼠标的索引（仅适用于多鼠标光标环境）。
ListEvent.ITEM_ROLL_OUT	鼠标光标滑离一个列表项。 <ul style="list-style-type: none"> • <i>itemRenderer</i>: 双击过的列表项。 IListItemRenderer 类型。 • <i>itemData</i>: 与列表项关联的数据。此值是从列表的 dataProvider 中检索的。 ActionScript Object 类型。 • <i>index</i>: 列表的新的选定索引。 int 类型。值 -1 (未设置选定的索引) 到列表项数量减去 1。

- **controllerIdx**: 用来生成该事件的控制器/鼠标的索引（仅适用于多鼠标光标环境）。

下例显示了如何监听列表项的点击动作：

```
myList.addEventListener(ListEvent.ITEM_CLICK, onItemClicked);
function onItemClicked(e:ListEvent) {
    trace("ListItemRenderer Clicked: " + e.itemRenderer);
    // Do something
}
```

2.4.5 TileList



图 41: 无皮肤标题列表 TitleList.

标题列表 **TitleList** (`scaleform.clik.controls.TitleList`) 与滚动列表 **ScrollingList** 类似，为一个可以滚动其元素的组件。可以自身展示列表项或者使用现存的场景中的列表项。可以附加滚动指示器 **ScrollIndicator** 或滚动条 **ScrollBar** 组件提供滚动反馈和控制功能。标题列表 **TitleList** 和滚动列表 **ScrollingList** 的区别为标题列表同时支持多个行和列，列表项选择可以向四个主要方向移动。该组件可与数据源 **dataProvider** 一起使用。数据源通过代码指派，如下面实例所示：

```
tileList.dataProvider = new DataProvider(["item1", "item2", "item3",
"item4", "item5"]);
```

默认情况下标题列表 **TitleList** 使用列表项渲染器 **ListItemRenderer** 组件作为内容。因此列表项渲染器必须在 **FLA** 文件库中存在，除非列表项渲染器检查属性改变成另外一个组件。见检查属性小节获得更多信息。

2.4.5.1 用户交互

点击一个列表项或者一个附属的滚动条 **ScrollBar** 实例将焦点传递到标题列表 **TitleList** 组件。当获得焦点，按下向上向下方向键或者类似的控制器，如果包含多行逐个单元垂直滚动列表选项。也可以用向左向右方向键或类似控制器，如果包含多列逐个单元水平滚动列表选项。如果标题列表 **TitleList** 包含多行和多列，

四个方向键或类似控制器可以用来导航列表项。如果没有选中元素，顶部元素自动作为默认选项。如果光标位于标题列表边界上鼠标滚轮可以滚动列表。

按下键盘(Page Up) 和 (Page down)或类似控制器将按页滚动选项，例如，列表中的可视选项数。按下(Home) 和 (End)键，或者类似控制器，将滚动列表到相应元素的开始和末尾处。与附属滚动条 ScrollBar 组件交互将按预期影响标题列表 TileList。见滚动条 ScrollBar 小节了解其用户交互功能。

2.4.5.2 组件设置

标题列表 TileList 不需要任何子单元，但是，在场景上的标题列表 TileList 组件上放置一个组件或者调整组件大小时一个可视化背景就能起到帮助作用。

2.4.5.3 状态

标题列表 TileList 组件支持三种状态，基于焦点和禁止属性：

- **default** 或 enable 状态；
- **focused** 状态，通常突出显示组件边框区域；
- **disabled** 状态。

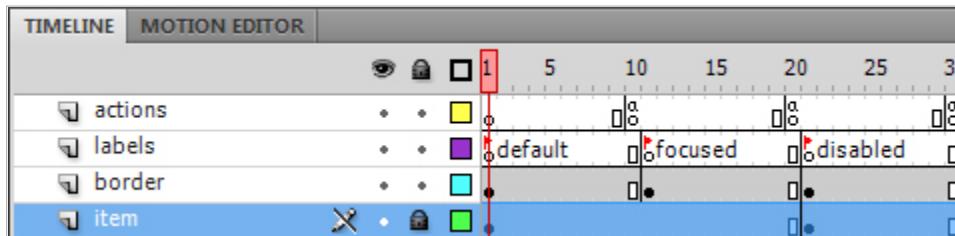
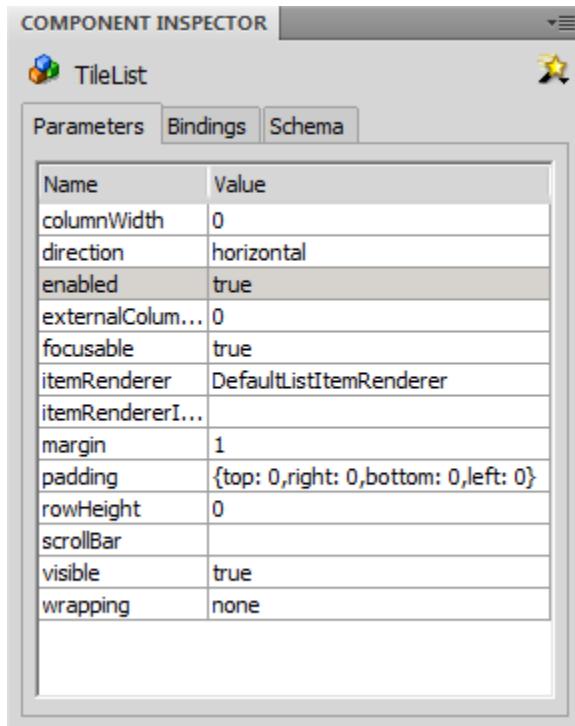


图 42: 标题列表 TileList 时间轴

2.4.5.4 检查属性



来自标题列表 **TileList** 组件的视频剪辑 **MovieClip** 具有以下检查属性：

columnWidth	内部创建的列表项实例宽度，如果设置了 <i>rendererInstanceName</i> 属性该值不产生影响。
Direction	滚动方向。行和列的语义不会随此值而变化。
Enabled	如果设置为 <i>false</i> ，就禁用组件。禁用的组件将不再收到用户输入。
externalColumnCount	当设置了 <i>rendererInstanceName</i> 属性，该值用来指示外部渲染器使用的列数的标题列表 TileList
Focusable	启用/禁用对于组件的焦点管理。将 <i>focusable</i> （可聚焦）属性设置为 <i>false</i> 将会取消对 <i>tab</i> 键、方向键以及基于鼠标的焦点更改的支持。
itemRenderer	<i>ListItemRenderer</i> 的符号名称。用来在内部创建列表项实例。如果设置了 <i>itemRendererInstanceName</i> 属性（即如果使用外部 <i>ListItemRenderer</i> ），则没有影响。
itemRendererInstanceName	要与此 <i>ScrollingList</i> 组件一起使用的外部列表项渲染器的前缀。 <i>Stage</i> 上的列表项实例必须以此属性值为前缀。如果将此属性设置为值 ‘r’，则要与此组件一起使用的所有列表项实例必须具有下列值：‘r1’，‘r2’，‘r3’，... 第一项应具有数字 1。
Margin	列表组件边界与内部创建的列表项之间的空余空间。如果设置了 <i>itemRendererInstanceName</i> 属性（即如果使用外部 <i>ListItemRenderer</i> ），则此值没有影响。

Padding	针对列表项的额外填充 (Padding)。如果设置了 <i>itemRendererInstanceName</i> 属性（即如果使用外部 <i>ListItemRenderer</i> ），则此值没有效果。填充不会影响自动生成的 <i>ScrollBar</i> （滚动条）。
rowHeight	内部创建的列表项实例的高度。如果设置了 <i>itemRendererInstanceName</i> 属性（即如果使用外部 <i>ListItemRenderer</i> ），则此值没有影响。
Visible	如果设置为 <code>false</code> ，就隐藏组件。这不会隐藏附加的 <i>ScrollBar</i> 或任何外部 <i>ListItemRenderer</i> 。

2.4.5.5 事件

所有事件回调均会收到单个 `Event`（事件）或 `Event` 子类参数，该参数包含有相应事件的相关信息。下列属性是所有事件所共有的。

- **`type`**: 事件类型。
- **`target`**: 事件目标。
- **`currentTarget`**: 通过一个事件监听器积极处理 `Event` 对象的对象。
- **`eventPhase`**: 事件流中的当前阶段。（`EventPhase.CAPTURING_PHASE`、`EventPhase.AT_TARGET`、`EventPhase.BUBBLING_PHASE`）。
- **`bubbles`**: 指明事件是不是起泡事件。
- **`cancelable`**: 指明与事件关联的行为是否可以避免

标题列表 `TileList` 组件产生的事件列表如下所示。事件旁列出的属性为通用属性的补充。

ComponentEvent.SHOW	<code>visible</code> （可见）属性已在运行时设置为 <code>true</code> 。
ComponentEvent.HIDE	<code>visible</code> 属性已在运行时设置为 <code>false</code> 。
ComponentEvent.STATE_CHANGE	组件的状态已更改。
FocusHandlerEvent.FOCUS_IN	组件已收到焦点。
FocusHandlerEvent.FOCUS_OUT	组件已失去焦点。
ListEvent.INDEX_CHANGE	选定的索引已发生变化。 <ul style="list-style-type: none"> • <i>itemRenderer</i>: 双击过的列表项。<code>IListItemRenderer</code> 类型。 • <i>itemData</i>: 与列表项关联的数据。此值是从列表的 <code>dataProvider</code> 中检索的。<code>ActionScript Object</code> 类型。 • <i>index</i>: 列表的新的选定索引。<code>int</code> 类型。值 <code>-1</code>（未设置选定的索引）到列表项数量减去 <code>1</code>。 • <i>controllerIdx</i>: 用来生成该事件的控制器/鼠标的索引（仅适用于多鼠标光标环境）。

ListEvent.ITEM_PRESS	已按下一个列表项。
	<ul style="list-style-type: none"> • <i>itemRenderer</i>: 双击过的列表项。 <code>IListItemRenderer</code> 类型。 • <i>itemData</i>: 与列表项关联的数据。此值是从列表的 <code>dataProvider</code> 中检索的。ActionScript Object 类型。 • <i>index</i>: 列表的新的选定索引。int 类型。值 -1 (未设置选定的索引) 到列表项数量减去 1。 • <i>controllerIdx</i>: 用来生成该事件的控制器/鼠标的索引 (仅适用于多鼠标光标环境)。
ListEvent.ITEM_CLICK	已点击一个列表项。
	<ul style="list-style-type: none"> • <i>itemRenderer</i>: 双击过的列表项。 <code>IListItemRenderer</code> 类型。 • <i>itemData</i>: 与列表项关联的数据。此值是从列表的 <code>dataProvider</code> 中检索的。ActionScript Object 类型。 • <i>index</i>: 列表的新的选定索引。int 类型。值 -1 (未设置选定的索引) 到列表项数量减去 1。 • <i>controllerIdx</i>: 用来生成该事件的控制器/鼠标的索引 (仅适用于多鼠标光标环境)。
ListEvent.ITEM_DOUBLE_CLICK	已双击一个列表项。
	<ul style="list-style-type: none"> • <i>itemRenderer</i>: 双击过的列表项。 <code>IListItemRenderer</code> 类型。 • <i>itemData</i>: 与列表项关联的数据。此值是从列表的 <code>dataProvider</code> 中检索的。ActionScript Object 类型。 • <i>index</i>: 列表的新的选定索引。int 类型。值 -1 (未设置选定的索引) 到列表项数量减去 1。 • <i>controllerIdx</i>: 用来生成该事件的控制器/鼠标的索引 (仅适用于多鼠标光标环境)。
ListEvent.ITEM_ROLL_OVER	鼠标光标滑过一个列表项。
	<ul style="list-style-type: none"> • <i>itemRenderer</i>: 双击过的列表项。 <code>IListItemRenderer</code> 类型。 • <i>itemData</i>: 与列表项关联的数据。此值是从列表的 <code>dataProvider</code> 中检索的。ActionScript Object 类型。 • <i>index</i>: 列表的新的选定索引。int 类型。值 -1 (未设置选定的索引) 到列表项数量减去 1。 • <i>controllerIdx</i>: 用来生成该事件的控制器/鼠标的索引 (仅适用于多鼠标光标环境)。

ListEvent.ITEM_ROLL_OUT

鼠标光标滑离一个列表项。

- *itemRenderer*: 双击过的列表项。 `IListItemRenderer` 类型。
- *itemData*: 与列表项关联的数据。此值是从列表的 `dataProvider` 中检索的。 `ActionScript Object` 类型。
- *index*: 列表的新的选定索引。 `int` 类型。值 -1 (未设置选定的索引) 到列表项数量减去 1。
- *controllerIdx*: 用来生成该事件的控制器/鼠标的索引 (仅适用于多鼠标光标环境)。

下例显示如何判断标题列表 `TileList` 接收焦点:

```
myList.addEventListener(FocusHandlerEvent.FOCUS_IN, onListFocused);
function onListFocused(e:FocusHandlerEvent) {
    trace("myList is now focused!");
    // Do something
}
```

2.4.6 DropdownMenu

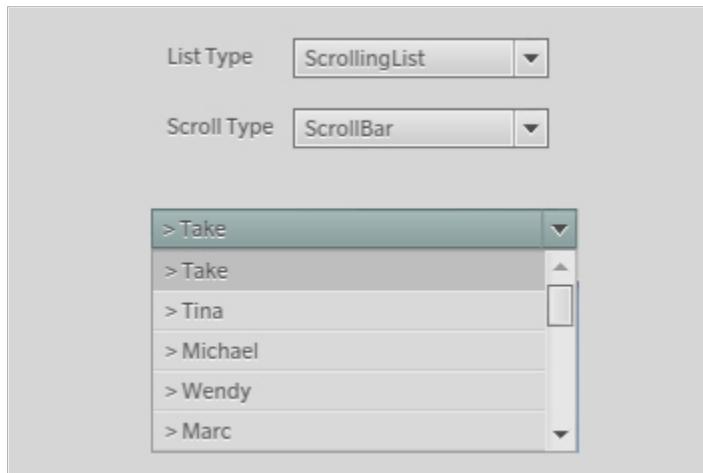


图 43: 无皮肤下拉菜单 `DropdownMenu`.

下拉菜单 `DropdownMenu` (`scaleform.clik.controls.DropdownMenu`) 包含了按钮和列表的行为。点击组件打开列表包含了选择元素。下拉菜单 `DropdownMenu` 只在静止状态显示选择元素。可以配置成使用滚动列表 `ScrollingList` 或标题列表 `TileList`，并可以配上滚动条 `ScrollBar` 或滚动指示器 `ScrollIndicator`。列表与数据源 `dataProvider` 相关联，下拉菜单 `DropdownMenu` 列表元素与数据源 `dataProvider` 进行连接。

数据源 `dataProvider` 通过代码指派，如下例所示:

```
dropdownMenu.dataProvider = new DataProvider([ "item1", "item2", "item3",
"item4" ]);
```

默认情况下下拉菜单 **DropdownMenu** 使用列表项渲染器 **ListItemRenderer** 组件作为内容。因此下拉菜单 **DropdownMenu** 和列表项渲染器 **ListItemRenderer** 必须在 FLA 文件库中存在，除非下拉属性改变成另外一个组件。参考检查属性小节获得更多信息。

也需注意默认情况下下拉菜单 **DropdownMenu** 在列表元素中不附带一个滚动条，滚动条 **ScrollBar** 或滚动指示器 **ScrollIndicator** 必须通过代码附加到下拉菜单 **DropdownMenu** 列表元素。见提示和技巧小节获得更多信息。

2.4.6.1 用户交互

点击下拉菜单 **DropdownMenu** 实例或者按下(**Enter**)键或类似控制器将打开可选元素列表。当打开时焦点转移到该列表，如在用户交互小节所描述用户能够与列表中 **ScrollingList**、**TileList** 和 **ScrollBar** 组件进行交互。点击一个列表项将其选中，关闭列表并在下拉菜单 **DropdownMenu** 组件中显示选择项。在列表边界外用鼠标点击将自动关闭列表，焦点将传递回下拉菜单组件。

2.4.6.2 组件设置

下拉菜单 **DropdownMenu** 继承了按钮组件的绝大多数功能。从而视频剪辑 **MovieClip** 使用下拉菜单类必须有以下名称的子单元。列表和滚动条动态创建，注出了对应的可选单元：

- **textField:** (可选) **TextField** 类型，按钮标签。
- **focusIndicator:** (可选) **MovieClip** 类型，一个单独的视频剪辑用来显示焦点状态，如果被使用，则视频剪辑必须拥有两个帧名为：“show”和“hide”。

2.4.6.3 状态

下拉菜单 **DropdownMenu** 打开时为选中状态，因此需要与 **ToggleButton** 和 **CheckBox** 具有相同的状态来指示选中状态，这些状态包括：

- **up** 或默认状态；
- 当鼠标箭头在组件上方或者获得焦点时为 **over** 状态；
- 当按钮被点击时候为 **down** 状态；
- **disabled** 状态；
- **selected_up** 或者默认状态；
- 当鼠标箭头位于组件上方或获得焦点时为 **selected_over** 状态；
- 当按钮被按下时为 **selected_down** 状态；

- ***selected_disabled*** 状态;

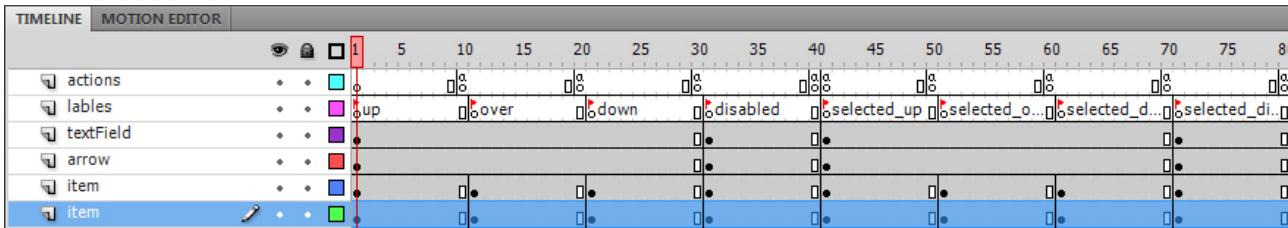
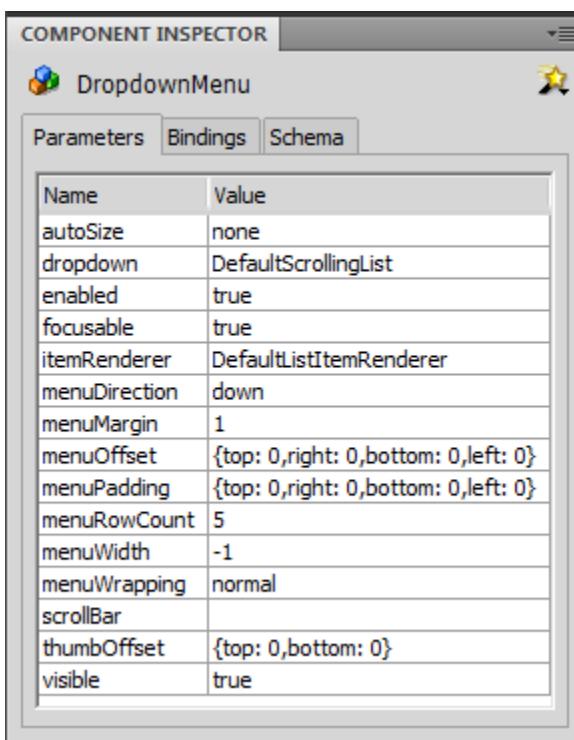


图 44: 下拉菜单 DropdownMenu 时间轴

这里为选择按钮下拉菜单 DropdownMenu 所需要的最少关键帧设置。按钮 Button 组件支持状态和关键帧的扩展设置，与 DropdownMenu 组件相同，在[CLIK 按钮入门](#)文档中有详细描述。

2.4.6.4 检查属性



下拉菜单 DropdownMenu 组件的检查属性为：

autoSize	确定已关闭的 DropdownMenu（下拉菜单）是否会缩放以适应其所包含的文本以及已调整大小的按钮将向哪个方向对齐。将 <code>autoSize</code> 属性设置为 <code>autoSize="none"</code> 将会使当前大小保持不变。
Dropdown	列表组件（ <code>ScrollingList</code> 或 <code>TileList</code> ）的符号名称，与下拉菜单 <code>DropdownMenu</code> 组件一起使用
Enabled	Disables the component if set to false.

Focusable	默认情况下，组件可以收到用于用户互动的焦点。将此属性设置为 <code>false</code> 将会禁用焦点获取。
menuDirection	下拉的列表将打开的方向。有效值为 "up"（向上）和 "down"（向下）。
menuMargin	列表部件与内部创建的列表项之间的边距。该边距还会对自动生成的滚动条产生影响。
menuOffset	下拉列表从下拉按钮位置的水平和垂直偏移量。正水平值会将列表移动到下拉按钮水平位置的右边。正垂直值会将列表移离该按钮。
menuPadding	针对列表项的顶部、底部、左侧和右侧的额外填充。不会影响自动生成的滚动条。
menuRowCount	列表应显示的行数。
menuWidth	If set, this number will be enforced as the width of the drop down's list.
thumbOffset	滚动条拇指顶部和底部偏移量。如果该列表不自动创建一个滚动条实例，则此属性没有影响。
scrollbar	下拉列表的滚动条的符号名称。由下拉列表实例创建。如果值为空，则下拉列表将没有滚动条。
Visible	如果设置为 <code>false</code> 则隐藏组件

2.4.6.5 事件

所有事件回调均会收到单个 `Event`（事件）或 `Event` 子类参数，该参数包含有相应事件的相关信息。下列属性是所有事件所共有的。

- **type:** 事件类型。
- **target:** 事件目标。
- **currentTarget:** 通过一个事件侦听器积极处理 `Event` 对象的对象。
- **eventPhase:** 事件流中的当前阶段。（`EventPhase.CAPTURING_PHASE`、`EventPhase.AT_TARGET`、`EventPhase.BUBBLING_PHASE`）。
- **bubbles:** 指明事件是不是起泡事件。
- **cancelable:** 指明与事件关联的行为是否可以避免

`DropdownMenu` 组件产生的事件列表如下所示。除 `change` 事件意外，与 `Button` 组件类似，事件旁列出的属性为通用属性的补充。

ComponentEvent.SHOW	<code>visible</code> 属性已在运行时设置为 <code>true</code> 。
ComponentEvent.HIDE	<code>visible</code> 属性已在运行时设置为 <code>false</code> 。
ComponentEvent.STATE_CHAN	该按钮的状态已发生变化。
GE	
Event.SELECT	选定的属性已发生变化。
FocusHandlerEvent.FOCUS_IN	该按钮收到了焦点。
FocusHandlerEvent.FOCUS_OUT	该按钮失去了焦点。

ListEvent.INDEX_CHANGE	选定的索引已发生变化。
	<ul style="list-style-type: none"> • <i>itemRenderer</i>: 双击过的列表项。 <code>IListItemRenderer</code> 类型。 • <i>itemData</i>: 与列表项关联的数据。此值是从列表的 <code>dataProvider</code> 中检索的。 <code>ActionScript Object</code> 类型。 • <i>index</i>: 列表的新的选定索引。 <code>int</code> 类型。值 -1 (未设置选定的索引) 到列表项数量减去 1。 • <i>controllerIdx</i>: 用来生成该事件的控制器/鼠标的索引 (仅适用于多鼠标光标环境)。
MouseEvent.ROLL_OVER	<p>鼠标光标滑过该组件。</p> <p><i>mouseIdx</i>: 用来生成该事件的鼠标光标的索引 (仅适用于多鼠标光标环境)。 <code>uint</code> 类型。仅限于 <code>Scaleform</code>, 需要将该事件归于 <code>MouseEventEx</code>。</p> <p><i>buttonIdx</i>: 指明为哪个按钮生成该事件 (基于零的索引)。仅限于 <code>Scaleform</code>, 需要将该事件归于 <code>MouseEventEx</code>。</p>
MouseEvent.ROLL_OUT	<p>鼠标光标已滑出该组件。</p> <p><i>mouseIdx</i>: 用来生成 <code>uint</code> 的鼠标光标的索引 (仅适用于多鼠标光标环境)。 <code>Number</code> (数字) 类型。仅限于 <code>Scaleform</code>, 需要将该事件归于 <code>MouseEventEx</code>。</p> <p><i>buttonIdx</i>: 指明为哪个按钮生成该事件 (基于零的索引)。仅限于 <code>Scaleform</code>, 需要将该事件归于 <code>MouseEventEx</code>。</p>
ButtonEvent.PRESS	<p>已按 <code>DropdownMenu</code> (下来菜单)。</p> <p><i>controllerIdx</i>: 用来生成该事件的控制器的索引 (仅适用于多鼠标光标环境)。 <code>uint</code> 类型。</p> <p><i>isKeyboard</i>: 为 <code>true</code> - 假如事件是由计算机键盘 (<code>Keyboard</code>)/游戏键盘 (<code>Gamepad</code>) 生成的; 为 <code>false</code> - 假如事件是由鼠标生成的。</p> <p><i>isRepeat</i>: 为 <code>true</code> - 假如事件是由一个被按住的 <code>autoRepeating</code> 按钮生成的; 为 <code>false</code> - 该按钮当前不重复</p>
MouseEvent.DOUBLE_CLICK	<p>已双击该组件。仅在 <code>doubleClickEnabled</code> 属性为 <code>true</code> 时激发。</p> <p><i>mouseIdx</i>: 用来生成该事件的鼠标光标的索引 (仅适用于多鼠标光标环境)。 <code>uint</code> 类型。仅限于 <code>Scaleform</code>, 需要将该事件归于 <code>MouseEventEx</code>。</p> <p><i>buttonIdx</i>: 指明为哪个按钮生成该事件 (基于零的索引)。仅限于 <code>Scaleform</code>, 需要将该事件归于 <code>MouseEventEx</code>。</p>
ButtonEvent.CLICK	<p>单击了该 <code>DropdownMenu</code>。</p> <p><i>controllerIdx</i>: 用来生成该事件的控制器的索引 (仅适用于多鼠标光标环境)。 <code>uint</code> 类型。</p> <p><i>isKeyboard</i>: 为 <code>true</code> - 假如事件是由计算机键盘 (<code>Keyboard</code>)/游戏键盘 (<code>Gamepad</code>) 生成的; 为 <code>false</code> - 假如事件是由鼠标生成的。</p> <p><i>isRepeat</i>: 为 <code>true</code> - 假如事件是由一个被按住的 <code>autoRepeating</code> 按钮生</p>

	成的；为 <code>false</code> – 该按钮当前不重复
ButtonEvent.DRAG_OVER	鼠标光标拖到了 <code>DropdownMenu</code> 上（同时按鼠标左按钮）。 <code>controllerIdx</code> : 用来生成该事件的控制器的索引（仅适用于多鼠标光标环境）。 <code>uint</code> 类型。
ButtonEvent.DRAG_OUT	鼠标光标拖离 <code>DropdownMenu</code> （同时按鼠标左按钮）。 <code>controllerIdx</code> : 用来生成该事件的控制器的索引（仅适用于多鼠标光标环境）。 <code>uint</code> 类型。
ButtonEvent.RELEASE_OUTSIDE	鼠标光标被拖出该按钮，并释放了鼠标左按钮。 <code>controllerIdx</code> : 用来生成该事件的控制器的索引（仅适用于多鼠标光标环境）。 <code>uint</code> 类型。

2.5 进度类型

进度类型用来显示状态或事件或动作的进度。The Scaleform CLIK 框架包含了两个组件属于此分类，状态指示器 `StatusIndicator` 和进度条 `ProgressBar`。状态指示器 `StatusIndicator` 组件用来显示事件或动作的状态。而进度条组件 `ProgressBar` 与状态指示器拥有相同的语义，但是包含了一些附加功能，可以监听其他产生进度事件的组件或动作。

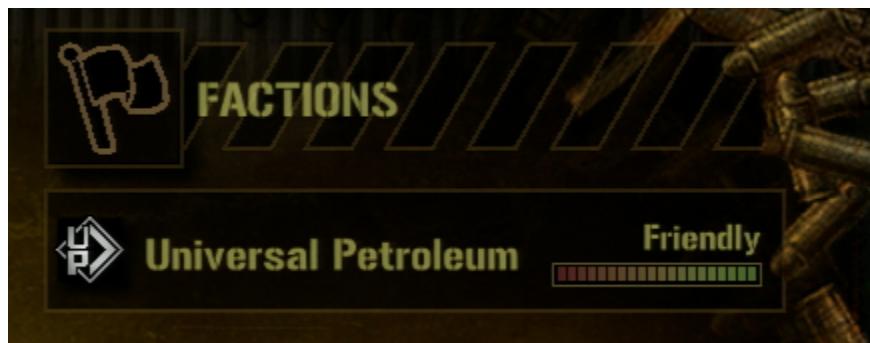


图 45:来自 *Mercenaries 2* 的 Faction 状态指示器实例

2.5.1 StatusIndicator

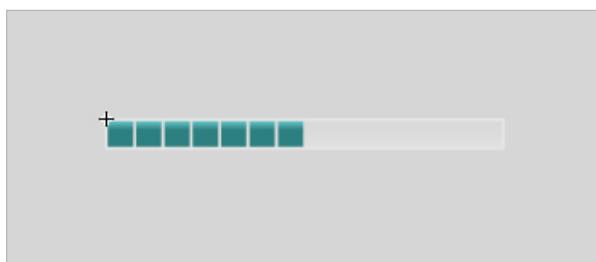


图 46: 无皮肤状态指示器 `StatusIndicator`.

状态指示器 StatusIndicator component (scaleform.clik.controls.StatusIndicator)显示事件或动作状态，使用时间轴作为视觉指示器。状态指示器的值将为一个最大和最小值之间的值用来产生一个帧的序号，以在组件时间轴上播放。由于组件时间轴用来显示状态，为创建创新的视觉指示器提供了绝对自由的发挥空间。

2.5.1.1 用户交互

状态指示器 StatusIndicator 无用户交互。

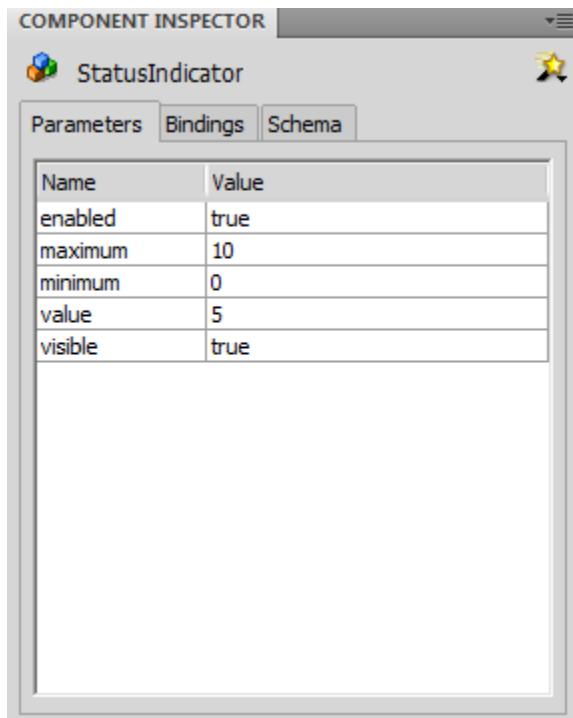
2.5.1.2 组件设置

使用 CLIK 状态指示器 StatusIndicator 的视频剪辑 MovieClip 不需要任何子单元。但是状态指示器需要至少两个帧以正确操作。确保在第一帧插入 stop()命令避免播放该帧。状态指示器组件在值属性产生的对应帧执行 gotoAndStop()命令。

2.5.1.3 状态

状态指示器 StatusIndicator 组件无状态，组件帧用来显示事件或动作

2.5.1.4 检查属性



来自状态指示器 StatusIndicator 组件的视频剪辑 MovieClip 具有以下检查属性：

Visible	如果设置为 <code>false</code> 则隐藏组件
Enabled	如果设置为 <code>false</code> 则禁止组件
Value	一个事件或动作的状态值，为一个最大值到最小值之间的播放的帧序号。
Minimum	插入目标帧的最小值
Maximum	插入目标帧的最大值

2.5.1.5 事件

所有事件回调均会收到单个 `Event` (事件) 或 `Event` 子类参数，该参数包含有相应事件的相关信息。下列属性是所有事件所共有的。

- **`type`**: 事件类型。
- **`target`**: 事件目标。
- **`currentTarget`**: 通过一个事件侦听器积极处理 `Event` 对象的对象。
- **`eventPhase`**: 事件流中的当前阶段。 (`EventPhase.CAPTURING_PHASE`、
`EventPhase.AT_TARGET`、`EventPhase.BUBBLING_PHASE`) 。
- **`bubbles`**: 指明事件是不是起泡事件。
- **`cancelable`**: 指明与事件关联的行为是否可以避免

`StatusIndicator` 组件产生的事件列表如下所示。事件旁列出的属性为通用属性的补充。

ComponentEvent.SHOW	<code>visible</code> 属性已在运行时设置为 <code>true</code> 。
ComponentEvent.HIDE	<code>visible</code> 属性已在运行时设置为 <code>false</code> 。

2.6 其他类型

Scaleform CLIK 框架页包括若干个组件且不能简单区分，但是为 UI 开发提供了宝贵的功能。 `Window` (窗口) 组件允许显示可用来显示内容或用作对话框的有模态和无模态窗口。

`DragSlot` 是一个 CLIK 组件的基础实现，该 CLIK 组件设计为用于需要拖放功能的 UI。拖放是作为 CLIK 内的一个自定义框架实现的，而且必须安装 CLIK DragManager (请参阅 `scaleform.clik.managers.DragManager.as`) 。

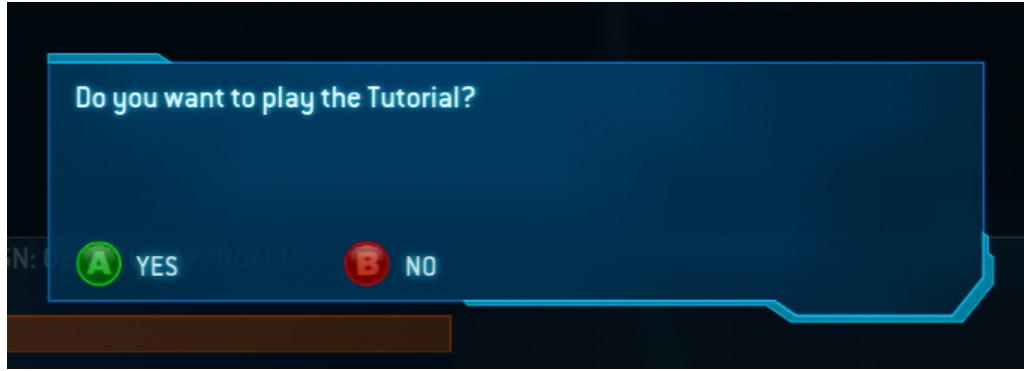


图 48:来自 *Halo Wars* 的对话框 Window 实例

2.6.1 Window

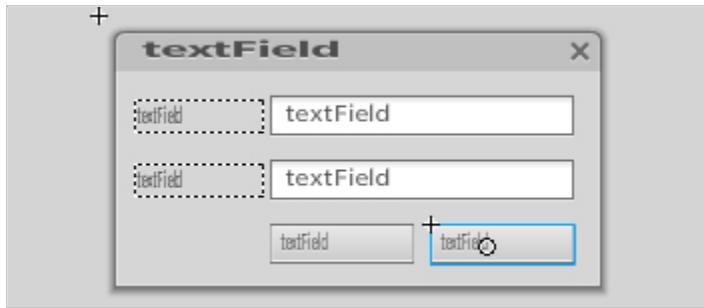


图 49:无皮肤对话框 Window 实例

CLIK Window 组件 (`scaleform.clik.controls.Window`) 显示一个窗口或一个对话视图，如一个 Alert (警告) 对话框。

注意内置组件没有任何内容，因为设计成完全由用户定义。但是，通过简单的编辑组件符号可以添加内容。

PopUpManager 提供如下功能：作为对话框打开一个窗口（参阅 `PopUpManager.showModal()`）和同时管理多个对话框。

2.6.1.1 用户交互

对话框 Dialog 的用户交互在创建的对话框视图里定义。

2.6.1.2 组件设置

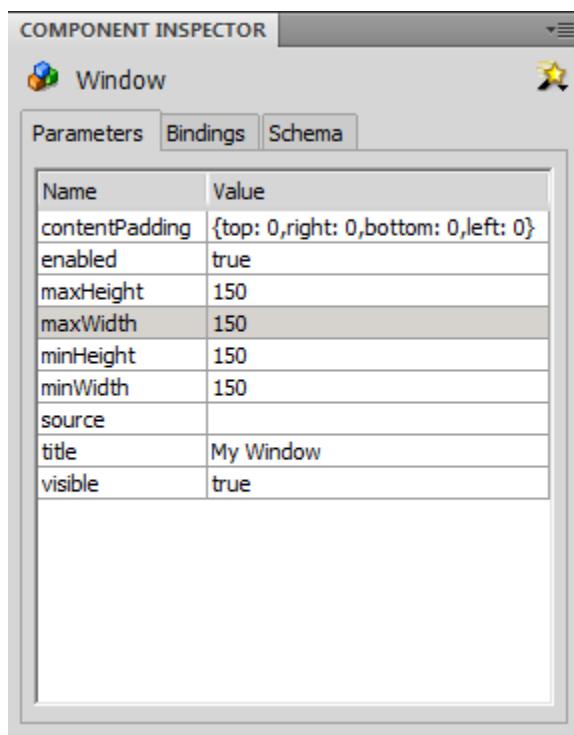
使用 Clik Dialog 类的 MovieClip 必须具备下面所列的子单元。也列出了可选元素：

- **closeBtn**: (可选) CLIK Button 类型。一个关闭该窗口的按钮。
- **titleBtn**: (可选) CLIK Button 类型。一个用于对话框的可拖曳标题栏。
- **okBtn**: (可选) CLIK Button 类型。一个“Accept”(接受)或“OK”(确定)按钮，单击时会导致窗口关闭。
- **resizeBtn**: (可选) CLIK Button 类型。一个按钮，当按住该按钮然后拖动时，它使用户可以重新调整窗口大小。
- **background**: (可选) MovieClip 类型。窗口的背景。如果重调窗口大小，该背景就会被缩放并变化大小。
- **hit**: (可选) MovieClip 类型。窗口的 hitArea。

2.6.1.3 状态

对话框 Dialog 组件无状态。视频剪辑 MovieClip 作为对话框视图显示，或许有或许没有自身状态。

2.6.1.4 检查属性



来自对话框 Window 组件的视频剪辑 MovieClip 具有以下检查属性：

contentPadding	顶部、底部、左侧和右侧填充，应该应用到加载到窗口中的内容。
Enabled	如果设置为 false 则禁止组件
maxHeight	通过重新调整大小按钮调整大小时窗口的最大高度。
maxWidth	通过重新调整大小按钮调整大小时窗口的最大宽度。

minHeight	通过重新调整大小按钮调整大小时窗口的最小高度。
minWidth	通过重新调整大小按钮调整大小时窗口的最小宽度。
Source	应加载到窗口中的符号的导出名称。
Title	如果窗口有一个 titleBtn 子元素，此字符串就会被用作其标签。
Visible	如果设置为 false 则隐藏组件

2.6.1.5 事件

所有事件回调均会收到单个 Event (事件) 或 Event 子类参数，该参数包含有相应事件的相关信息。下列属性是所有事件所共有的。

- **type:** 事件类型。
- **target:** 事件目标。
- **currentTarget:** 通过一个事件侦听器积极处理 Event 对象的对象。
- **eventPhase:** 事件流中的当前阶段。（EventPhase.CAPTURING_PHASE、EventPhase.AT_TARGET、EventPhase.BUBBLING_PHASE）。
- **bubbles:** 指明事件是不是起泡事件。
- **cancelable:** 指明与事件关联的行为是否可以避免

Window 组件产生的事件列表如下所示。事件旁列出的属性为通用属性的补充。

ComponentEvent.SHOW visible 属性已在运行时设置为 true。

ComponentEvent.HIDE visible 属性已在运行时设置为 false。

以下例子显示如何处理对话框提交事件：

```
myWindow.addEventListener(ComponentEvent.HIDE, onWindowClosed);
function onWindowClosed(e:ComponentEvent) {
    // Do something.
}
```

3 艺术细节

本章将帮助美工设计师开发 Scaleform CLIK 组件的皮肤，包括为小的组件实例绘制皮肤的详细过程和最优方法，以及动画和内置字体。

3.1 最优方法

本章包括了为 Scaleform CLIK 组件创建皮肤的最优方法。

3.1.1 像素图像

当开发基于 CLIK 组件绘制矢量皮肤，建议所有资源都用像素图像。一个完美的像素资源在 Flash 栅格化中完美排列。

为使得能够改变栅格：

1. 从 Flash 顶部菜单选择视图；
2. 选择栅格，点击使栅格显示；
3. 从 Flash 顶部菜单再次选择视图；
4. 选择栅格然后点击编辑栅格；
5. 在垂直和水平位置输入 1px；
6. 点击 OK。

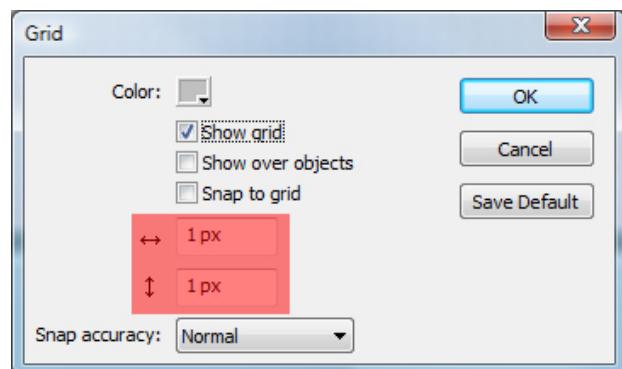


图 51: 编辑栅格 Grid 视窗

现在应该可以看到覆盖在场景上的一个栅格。每个栅格表示一个像素。确保当创建美术资源时对齐栅格。这回确保最终的 SWF 中图像不会模糊。**注意：**保持所有图像尺寸为 2 的指数倍；否则可能会导致模糊，见下面的 2 的指数倍标题。

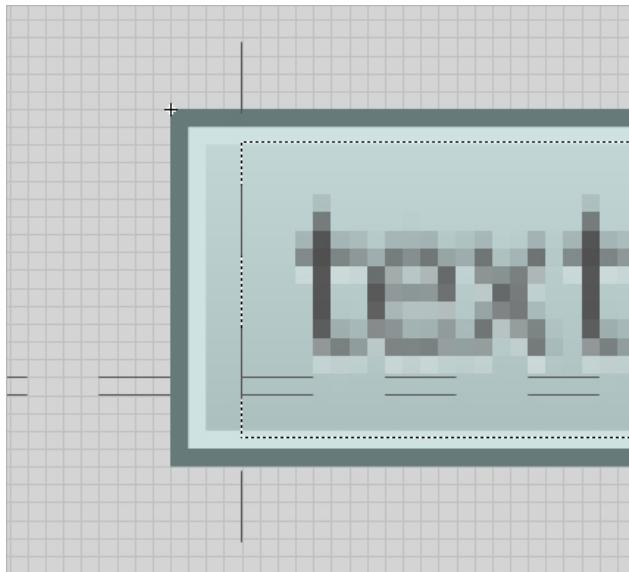


图 52: 一个像素的矢量图像

3.1.2 蒙版

在 Flash 中蒙板可以使设计师在运行时隐藏部分图像。蒙板经常在动画效果中使用，但是，蒙板消耗大量资源。Scaleform 建议尽量避免蒙板的使用。如果蒙板必须使用，保持在个位数之内并测试添加蒙板前后动画性能。

在 Flash 中使用蒙板的一个可能选择为在 Photoshop®中创建 PNG，用透明混合处理需要遮盖的区域。但是，这只能用于非动画图像的透明区域。

3.1.3 动画

最好避免动画中矢量图形的转换，如将一个正方形转换成圆形。这些动画类型开销非常大，因为这些形状在每一个帧都需要重新进行评估。

如果可以避免在矢量图形中缩放动画，额外的栅格化影响内存和性能 – 将矢量图形转换为三角元素图像的过程。开销最小的动画为使用平移和旋转，不需要额外的栅格化。

避免使用可编程映射动画，选择以映射动画为基础的时间轴选择，因为时间轴映射动画在性能上更加有优势。使用时间轴映射动画，保持在既定帧速率下实现一个平滑的动画所需要的最少帧数量。

3.1.4 图层和绘制元素

在 Flash 中创建皮肤使用尽可能少的图层。每使用一个图层至少增加一个绘制元素。绘制元素使用越多，内存需求就越大，性能也将受到影响。

3.1.5 复杂皮肤界面

在复杂皮肤中最好使用位图；但是，在简单皮肤中使用矢量图可以节省更多内存并可以在任何分辨率下缩放而不失真（模糊）。

3.1.6 2 的指数倍

确保位图在尺寸上为 2 的指数倍，2 的指数倍位图尺寸如下所示：

- 16x16
- 32x32
- 64x64
- 128x128
- 128x256
- 256x256
- 512x256
- 512x512

3.2 已知问题和推荐工作流程

本节包括了已知艺术相关 Flash 问题列表和在 Scaleform CLIK 下的推荐工作流程。

3.2.1 复制组件

在 Flash 中复制组件存在几个问题，复制组件导致原来组件的链接信息和组件定义信息不拷贝到新的组件中去。同样的，有些组件没有这些链接信息将无法工作。本节描述了两种方法来解决这个问题。

3.2.1.1 复制组件（方法 1）

从外部 FLA 文件复制一个无更改（无皮肤）CLIK 组件，如按钮，到目标 FLA 文件最快的方如下：

1. 打开 *CLIK_Components.fla* 文件。
2. 从文件的库中拷贝组件（例如，按钮）到目标 FLA 文件，在源文件库中点击并选择 *Copy*，然后点击目标 FLA 中的库并选择 *Paste*。
3. 点击目标 FLA 库中的组件并选择 *Rename* 对其重新命名，不要与‘Button’重名。
4. 再次点击目标 FLA 库中的组件，选择 *Properties*。
5. 改变 *Identifier* 区域匹配步骤 3 中选择组件的新名称。
6. 点击库视窗中的空白区域并选择 *Paste*，一个新的原始组件拷贝将粘贴进来包括所有的完整的链接信息。

3.2.1.2 复制组件 (方法2)

当在相同的库中复制符号（组件），链接信息将不会被拷贝到新的复制符号中去。组件功能执行需要这些信息，实现的方法为：

1. 点击 *library* 面板中需要复制的组件并选择 *Properties*。
2. 在 *Properties* 视窗，双击文本（例如 `scaleform.clik.controlsButton`）突出显示 *Class* 文本区域并按下(CTRL+C)键进行拷贝。
3. 按下 *Cancel*。
4. 再次点击需要拷贝的组件，并选择 *Duplicate*。
5. 在 *Duplicate Symbol* 视窗中，点击 *Export* 使 *ActionScript* 复选框使能。
6. 点击 *Class* 区域并按下(CTRL+V)来粘贴链接信息到该文本区域 *textField*。
7. 按下 *OK*。
8. 点击 *library* 面板中新拷贝的组件并选择 *Component Definition*。
9. 点击 *Class textField* 空白区域并按下(CTRL+V)来粘贴链接信息到文本区域 *textField*。
10. 按下 *OK*。

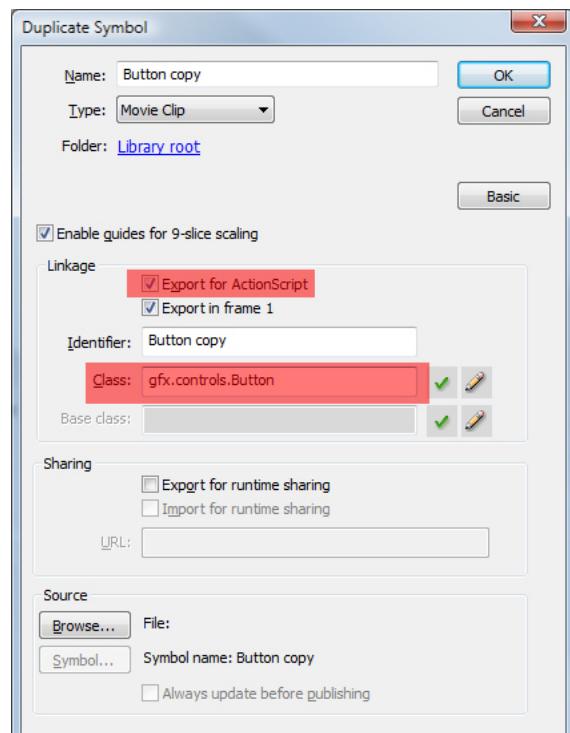


图 53: 复制符号链接（必须填充红色显亮区域）

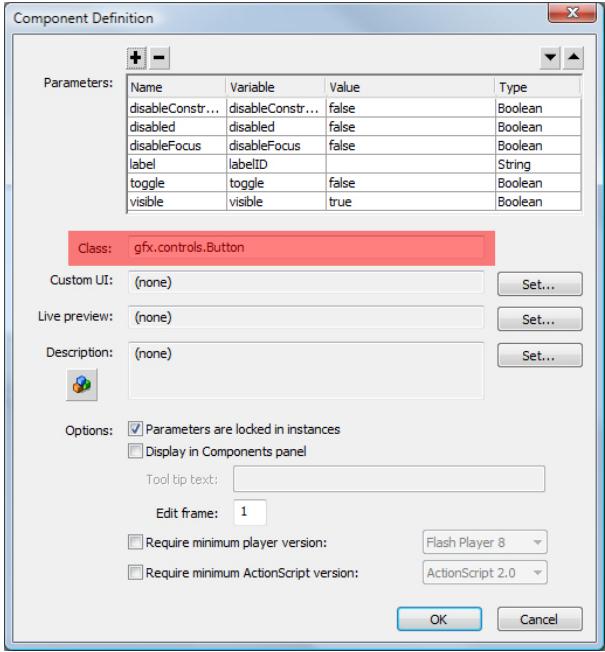


图 54: 组件定义 (必须填入类)

3.3 皮肤绘制实例

Scaleform CLIK 的主要优势为视觉和函数层的分离。分离能够使大部分部件中编程人员和艺术设计师可以各自独立工作。轻松自定义外观和风格为这种分离的主要优势之一。

尽管内置 CLIK 组件具有一个标准的外观和风格，但有时候需要自定义以满足客户自身需求。以下部分即描述了自定义内置组件的方法。

CLIK 组件设置皮肤与任何 Flash 标准符号相同，双击 FLA 文件库中的一个组件获得组件任意一个时间轴：

- 在 Flash 中修改默认皮肤；
- 在 Flash 中从头开始创建自定义皮肤；或者
- 导入 Photoshop 或 Illustrator® 中创建的艺术资源到 Flash。

请浏览文档 [CLIK 入门](#) 获得关于皮肤绘制指南的详细信息。

3.3.1 StatusIndicator 皮肤绘制

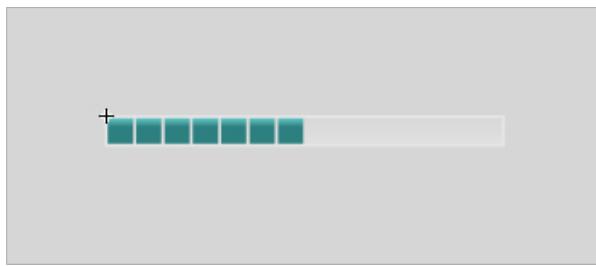


图 55: 无皮肤 StatusIndicator.

状态指示器 StatusIndicator 为一个唯一的组件在绘制皮肤时与其他大多数基于按钮的组件需要略微不同的方法，打开状态指示器组件，记录时间轴，具有两个图层：*indicator* 和 *track*。*Track* 用来显示背景图像；无其他功能。*Indicator* 图层使用若干关键帧包括位图或矢量图来从最低到最高的逐步递增状态。

指南中使用 Photoshop 文件中创建的若干个图层的位图来绘制状态指示器。这为状态指示器皮肤绘制的方法之一，但是，还有其他的方法可以在场景中创建和装配图形。最终的结果应该都相同，能够使状态指示器正确工作。



图 56: 状态指示器 StatusIndicator PSD 文件

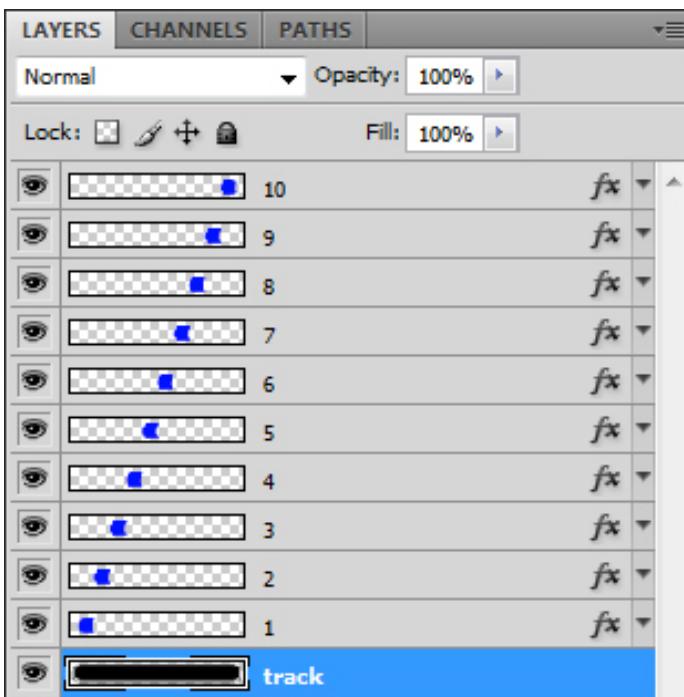


图 57: 在 Photoshop 中为 StatusIndicator PSD 文件设置的图层

1. 在 Photoshop 中创建状态指示器 StatusIndicator 位图与上面描述类似。注意图层顺序和编号以及在图层中每个图像的位置。确保背景透明，为本使用手册创建一个类似文件；
2. 保存文件为 PSD 格式；
3. 在 Flash 中，从菜单选择 *File*，然后选择 *Import -> Import to Stage*；
4. 浏览并选择状态指示器 StatusIndicator 皮肤 PSD 文件；
5. 在 *Import* 窗口，确保 *Convert layers to:* 下拉菜单设置为‘Layers’；
6. 按下 *OK*；
7. 一个新的 Flash 时间轴图层应该在 PSD 文件中的每个层中创建，在上面图像例子中，需要创建 10 个图层，因为 PSD 文件内部拥有 10 个图层（每个渲染为一个层）。每个图层标签分别从 1 到 10，1 为最底部图层 10 为最顶部图层，选择 *layer 1*；
8. 在场景中的位图图像周围绘制一个复选框；
9. 将图像移动到老的无皮肤轨道位置，根据要求进行缩放；
10. 在 *layer 1* 上选择第一个关键帧并拖动到第 6 帧；
11. 点击帧选择 *Insert* 关键帧添加新的帧到 *layer 1* 图层的第 11 帧；
12. 在 *layer 2* 图层选择位图并按下(Ctrl+X)进行剪切；
13. 选择 *layer 1* 图层中第 11 帧位置为新关键帧并按下(Ctrl+Shift+V)放置位图到 *layer 1* 图层；
14. 重复此过程直到 10 个位图都在同一个图层(*layer 1*)，位于正确的关键帧。每个并排的位图应该拷贝到最后关键帧第 5 帧后的一个关键帧中。图层 *layer 2* 中的位图应该被拷贝到关键帧 11；图层 *layer 3* 中的位图应该被拷贝到第 16 帧；图层 *layer 4* 中的位图应该被拷贝到第 21 帧，等等。使用 10 个 PSD 图像，最后关键帧应该位于第 51 帧；
15. 删除空图层 (2-10) 进行清理；
16. 如果在最后的位图关键帧之后时间轴上有附加的帧，加上另外五个关键帧，然后选择剩余帧并删除，删除操作为首先选择然后右键点击并选择 *Remove Frames* 即可。在本使用手册中，最后帧应该位于第 55 帧；
17. 选择原来的 *indicator layer* 并删除；
18. 选择原来的 *track* 图层并删除，确保不删除新导入的 *track* 图层；
19. 选择 *layer 1* 并重命名为‘indicator’；
20. 拖动 *indicator* 图层和 *track* 图层到 *actions layer* 图层下方，确保 *track* 图层在 *indicator* 图层下面；

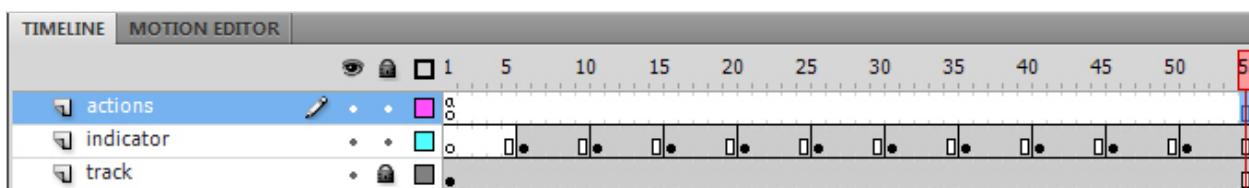


图 58: 最终时间轴（注意关键帧位置和最终帧位置）

21. 推出状态指示器 StatusIndicator 时间轴；
22. 设置检查参数值为 1 到 10 之间的任何值；

23. 保存文件；
24. 发布文件查看新绘制皮肤的指示器。

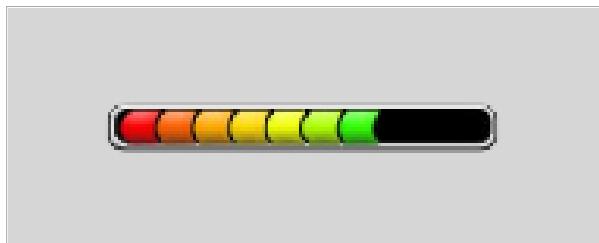


图 59: StatusIndicator 皮肤绘制

3.4 字体和本地化

本节详细描述了在 Scaleform CLIK 组件中字体使用。

3.4.1 概要

为了在 Scaleform 中字体能被正确渲染，所需字形（字符）必须内置到 SWF 或使用 Scaleform 本地化系统进行设置。以下为探究在 Flash UI 界面中管理字体的方法。

3.4.2 内置字体

和 Flash 播放器不同，Scaleform 需要内置字体以便显示。Scaleform 播放器在未嵌入字体情况下将显示空的进行矩形，甚至这些字体在系统中已存在也是如此。这个效果的优势为方便判断在 Scaleform Scaleform 中字体是否正确嵌入。在内置组件设置中，在每个文本区域 `textFidld` 实例中都嵌入了 `Slate Mobile`。注意 `Slate Mobile` 不包含任何中文、日文或韩文（CJK）字符或字形，内置组件只包含了 ASCII 字形。这可以通过将 `Slate Mobile` 替换为包含 CJK 字形并设置适当的嵌入选项进行改变。

注意直接内置字体到文本区域 `textFields` 与 Scaleform 本地化系统不兼容（在 3.4.4 小节有所描述）。Scaleform 实际上推荐使用 Scaleform 本地化系统来设置字体，比直接嵌入具有很多优势。但是在某些情况下，如不需要本地化，内置字体最好为可选项。与 UI 界面管理类似，字体管理也需要几点考虑如本地化和内存管理。

3.4.3 在 `textField` 嵌入字体

只需在动态或输入文本区域 `textFields` 中嵌入字体，静态文本在编译时自动转换为字形轮廓（原始矢量图形）。在动态文本区域 `textFidld` 中嵌入字体，在场景中选择动态文本区域的属性检查(Windows >

Property Inspector)框中点击 *Embed* 按钮。选择应该嵌入的字符或字符集并选择 *OK*。一旦完成了这些步骤，在你的 **FLA** 文件中就可以使用该字体。在相同的 **FLA** 文件的多行文本区域中如果需要使用相同的字体，这些步骤只需要执行一次。同样，多次嵌入相同字体不会增加内存使用量。注意字符集包含了大量的字形如中文在导入时占用大量的内存。

3.4.4 本地化系统

Scaleform 本地化系统使用热交换机制无论在当前位置是否发生变化均可导入和导出字体库。这些字体库本身为 **SWF** 文件包括内置字体字形可以被 **SWF** 文本使用。**Scaleform** 能够使用来自字体库的字形，为根据需求动态改变字体提供了强大的方法。

由于 **Scaleform** 本地化系统在文本区域 **textField** 直接嵌入字形时不能交换字体，文本区域必须使用一个导入的字形符号。通常字体符号在一个名为 **gfxfontlib.fla** 的文件中创建并导入到 **swf** 文本中。这个导入的字体设置到所有支持字体交换的文本区域中。**Scaleform** 当前可以截留该字体符号链接并基于当前位置导入不同的字体。

字体库不需要导出任何字体，只需要插入适当的字体（见前面小节的如何插入字体）。**Scaleform** 本地化系统使用 **fontconfig.txt** 文件来定义每个位置的字体库，以及文本区域使用的字体符号之间的字体映射关系和插入到字体库中的实体字体。

fontconfig.txt 文件也包含转换对应关系。**Scaleform** 本地化系统可以自行文本的快速转换，这些文本显示在每个场景中的动态或输入文本区域。例如，如果一个文本区域包含文本‘\$TITLE’，转换映射具有一个对照表如\$TITLE=Scaleform，然后文本区域将自动显示‘Scaleform’来代替。

本章中描述的信息作为 **Scaleform** 字体和本地化系统的一个概括。为深入理解 **Scaleform** 中字体和本地化，请参考文档[字体概述](#)。

4 编程详述

本章描述了子系统框架和亮点的具体细节，提供了 Scaleform CLIK 组件架构的上层理解。

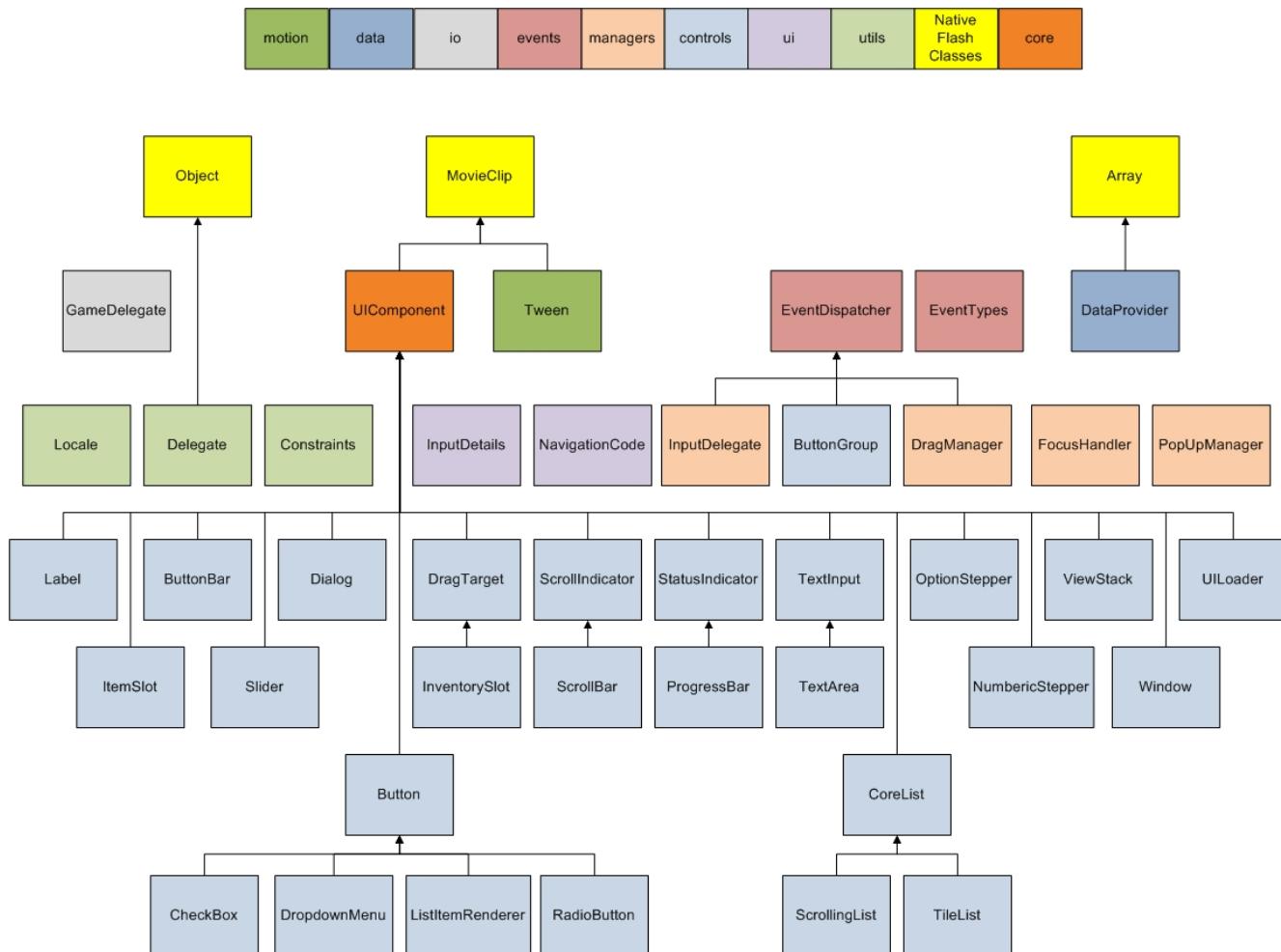


图 60: Scaleform CLIK 类层次结构

4.1 UI 组件 **UIComponent**

内置组件章描述了与 Scaleform CLIK 捆绑的组件使用的类。所有这些组件都继承了 **UIComponent** 类 (`scaleform.clik.core.UIComponent`) 的核心功能。这些类为所有 CLIK 组件的基础，Scaleform 推荐自定义组件为 **UIComponent** 的子类。

UIComponent 类本身从 Flash 8 视频剪辑 **MovieClip** 类继承而来，因此继承了标准 Flash 视频剪辑 **MovieClip** 的所有属性和方法。一些自定义读/写属性也被 **UIComponent** 类所支持。

- **enabled;**
- **visible;**
- **focused;**

- ***focusable***;
- ***width***;
- ***height***;
- ***scaleX***;
- ***scaleY***;
- ***displayFocus***: 如果组件应该显示焦点状态则设置为 true, 更多信息请参考[焦点处理](#) 小节。
focusTarget: Settings this property to another MovieClip will cause that MovieClip to receive focus rather than this component if focus is ever set to this MovieClip. For more information see the [Focus Handling](#) section.

UIComponent 拥有几个需要子类实现的空方法。这些方法包括:

- ***configUI***: 组件配置调用;
- ***draw***: 但组件无效时候调用, 更多信息, 请参考[失效](#) 小节;
- ***changeFocus***: 当组件接收或失去焦点时被调用;
- ***scrollWheel***: 当鼠标光标在组件上方滚动滚轮时被调用。

UIComponent 混合到 EventDispatcher 类用来支持事件预订和指派, 因此, 所有的子类都支持事件预订和指派。更多信息, 请参考[事件模型](#) 小节。

4.1.1 初始话

改类在 `onLoad()`事件句柄内执行下列初始化步骤:

- 设置 MovieClip 默认尺寸大小;
- 执行组件配置, 改步骤调用 `configUI()`方法;
- 绘制内容, 该步骤调用 `validateNow()`方法, 立即执行画面刷新, 例如调用 `draw()`函数。

4.2 组件状态

几乎所有的 Scaleform CLIK 组件支持视觉状态, 状态通过组件时间轴上的一个特殊关键帧的导航来设置, 或者传递状态信息到子单元。具有三个常用的状态设置, 详细内容见下面内容。

4.2.1 按钮组件

任何行为类似按钮的组件, 响应鼠标动作, 可以被选到此分类。例如使用此类的 CLIK 组件为 Button 及其变量、ListItemRenderer、RadioButton 和 CheckBox。复杂组件的的子元素如滚动条 ScrollBar 也属于按钮 Button 组件。归到此类的 CLIK 组件可以直接使用按钮 Button 类(`scaleform.clik.controls.Button`)或者使用从按钮 Button 类继承而来的类。

按钮组件支持的基本状态为:

- ***up*** 或默认状态;
- 当鼠标箭头在组件上方或者获得焦点时为 ***over*** 状态;

- 当按钮被点击时候为 **down** 状态;
- **disabled** 状态;

按钮 **Button** 组件也支持前缀状态，可以根据其他属性的值进行设置。默认情况下，核心 **CLIK** 按钮 **Button** 组件只支持一个“**selected_**”前缀，但组件处于选中状态时追加到帧的标签。

按钮组件支持的基本状态，包括选中状态，为以下所示：

- **up** 或默认状态;
- 当鼠标箭头在组件上方或者获得焦点时为 **over** 状态;
- 当按钮被点击时候为 **down** 状态;
- **disabled** 状态;
- **selected_up** 或者默认状态;
- 当鼠标箭头位于组件上方或获得焦点时为 **selected_over** 状态;
- 当按钮被按下时为 **selected_down** 状态;
- **selected_disabled** 状态;

注释：本节中涉及的状态只是 **CLIK** 按钮组件支持的状态类型里的一部分。详细的状态列表请参考 [CLIK 按钮入门](#) 文档。

CLIK 按钮类提供了一个 **getStatePrefixes()**方法，使开发者能够根据组件属性改变列表前缀。该方法定义如下：

```
protected function getStatePrefixes():Vector.<String> {
    return _selected ? statesSelected : statesDefault;
}
```

如之前提到的，**CLIK** 按钮默认情况下只支持“**selected_**”前缀。**getStatePrefixes()**方法根据选择属性返回不同前缀序列。该前缀序列将于适当的状态标签协同使用来确定帧的播放。

当状态在内部进行设置，例如鼠标滚动，出现一个帧标签列表的查找表。按钮类中的 **stateMap** 属性定义了帧标签映射的状态。以下为 **CLIK** 按钮类中定义的状态映射关系：

```
protected var _stateMap:Object = {
    up:[ "up" ],
    over:[ "over" ],
    down:[ "down" ],
    release: [ "release", "over" ],
    out:[ "out", "up" ],
    disabled:[ "disabled" ],
    selecting: [ "selecting", "over" ],
```

```

        toggle: [ "toggle", "up" ],
        kb_selecting: [ "kb_selecting", "up" ],
        kb_release: [ "kb_release", "out", "up" ],
        kb_down: [ "kb_down", "down" ]
    }

```

每个状态可能有多个目标标签，从状态表返回的值与 `getStatePrefixes()` 返回的前缀结合产生一个播放的目标帧列表。下图描述了用来决定正确帧播放的完整过程：

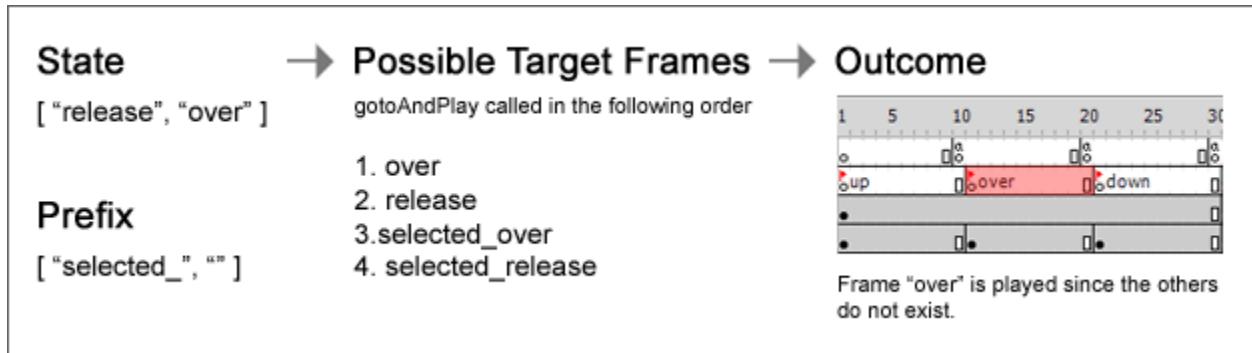


图 61:决定正确帧播放的过程

如果不能找到特定前缀的帧，播放位置总是跳转到最后的帧，组件为之前请求帧的默认状态。开发者可以忽略状态映射而创建自定义行为。

4.2.2 非按钮交互组件

这些涉及任何组件具有交互和接收焦点功能，但不响应鼠标事件。例如使用这类方法的 `CLIK` 组件为 `ScrollingList`、`OptionStepper`、`Slider` 和 `TextArea`。这些组件可能包含子单元可以响应鼠标事件。无按钮交互组件支持的状态为：

- **default** 状态；
- **focused** 状态；
- **disabled** 状态。

4.2.3 非交互组件

这些指向任何组件为非交互性质，但是可以被禁止。标签组件为状态所支持的默认设置组件中唯一的非交互组件。状态支持的非交互组件为：

- **default** 状态；
- **disabled** 状态。

4.2.4 特殊案例

有几个组件不遵守上面描述的状态规则，状态指示器 `StatusIndicator` 及其子类进度条 `ProgressBar` 使用时间轴显示组件值。播放位置将设置到帧的位置表示组件值的比例。例如，状态指示器中的一个 50 帧时

间轴最小值为 0，最大值为 10，当前值设置为 5（50%）将 `gotoAndStop()` 到第 25 帧（50 帧的 50% 处）。扩展这些组件管理这些显示程序非常容易。`updateValue()` 方法可以被修改或忽略以改变其行为。

在某些情况下，某些组件除默认行为外还具有特殊模式，支持额外组件状态。“文本输入”和“文本区域”组件在设置了“动作按钮”的情况下，能够支持“滚动”和“滑出”状态，以支持鼠标滚动和滑出事件。

4.3 事件模型

Scaleform CLIK 组件框架使用一个通信范例称之为事件模型 *event model*。组件在改变或交互时“分派”事件，容器组件可以访问不同的事件。这就可以向多个对象通知更改。ActionScript 3 实际上包括一个由 CLIK 利用的强大的事件系统。

有关 ActionScript 3 事件系统的详细信息，请访问：

http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/events/Event.html

4.3.1 最佳使用方法

4.3.1.1 预订一个事件

预订一个事件，可以使用 `addEventListener()` 方法，包括一个类型参数指定监听交互事件的类型。反过来使用 `removeEventListener()` 方法将取消事件预订。如果多个监听器包含相同的参数，只能触发一个。上述方法的每个方法还需要类型函数的一个侦听器参数，该类型函数与一个以前定义的函数或函数变量相关联。

CLIK 使用多种自定义事件，每个事件都有其自己的独特属性，这些属性用来提供有关此事件的更详细的信息。从 `Event`（事件）类派生的所有标准 CLIK 事件均可在 `scaleform.clik.events` 中找到。

请注意，`addEventListener()` 还接受另外三个参数，并为其提供有默认值：**useCapture**、**priority** 和 **useWeakReference**。一般说来，用户将会希望所有其事件侦听器使用弱引用，以避免维护对对象的强引用，如不维护，这些对象就会成为收集来的垃圾。要确保您的侦听器将会使用一个弱引用，您的 `addEventListener()` 语法应如下所示：

```
buttonInstance.addEventListener(ButtonEvent.CLICK, onButtonClick, false,  
0, true);
```

有关 `addEventListener()` 的参数的更多信息，请访问：

[http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/events/EventDispatcher.html#addEventListener\(\)](http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/events/EventDispatcher.html#addEventListener())

```
buttonInstance.addEventListener(ButtonEvent.CLICK, onButtonClick, false,  
0, true);
```

```
function onButtonClick (e:Event):void {
    buttonInstance.removeEventListener(ButtonEvent.CLICK,
onButtonClick, false);
}
```

在此示例中，此类正在创建一个侦听器，该侦听器针对从 `buttonInstance` 发出的 `ButtonEvent.CLICK` 事件进行侦听。无论何时收到此事件，都将会执行 `onButtonClick` 函数。然后此侦听器函数取消事件侦听器。

事件的起因和类型不同，针对该侦听器的 `Event`（事件）参数的属性有所不同。`CLIK` 组件产生的详细事件对象（及其属性）列表，请参考[内置组件](#)相关章节。

4.3.1.2 指派一个事件

`CLIK` 组件希望使用 `dispatchEvent()` 方法通知预订监听器发生的变化或交互动作。该方法需要一个参数：一个包含相关数据的对象，包括一个强制类型属性指定指派事件类型。组件的框架自动添加一个目标属性，作为事件指派对象的索引，但是能够手动设置为用自定义目标进行替换。

```
dispatchEvent(new Event(Event.CHANGE));
```

4.4 运行时创建组件

在 ActionScript 3 中，要在运行时动态创建一个组件，您应使用以下语法：

```
import flash.utils.getDefinitionByName;
import scaleform.clik.controls.Button;

public var myButton:Button;

// 检索对 Class (类) 定义的引用。
var classRef:Class = getDefinitionByName("ButtonLinkageID") as Class;
// 实例化该类的一个新实例，并将其存储起来。
if (classRef != null) { myButton = new classRef () as Button; }
myButton.addEventListener(ButtonEvent.CLICK, onButtonClick, false, 0, true);
```

在此示例中，`.FLA` 的库中存在一个符号，其 `Class`（类）设置为“`ButtonLinkageID`”，而其 `Base Class`（基类）设置为 `scaleform.clik.controls.Button`。

如果 `.FLA` 的库中存在一个有类但无基类的符号，只需通过使用该类的构造函数，就可以创建该符号的一个实例，因为该符号和该类会被联接在一起。

4.5 焦点处理

Scaleform CLIK 组件使用一个自定义焦点处理框架，在多数组件中执行，并且应该在无框架组件和符号中正常工作。一创建单个组件类，就会实例化 **CLIK FocusHandler** 管理器 (`scaleform.clik.managers.FocusHandler`)。没有必要直接实例化 **FocusHandler**。无需直接显示焦点句柄 **FocusHandler**。

所有焦点变化都是在 **Scaleform** 播放器层发生的，要么通过鼠标或键盘（游戏键盘）焦点变化，要么通过 ActionScript 变化。设置焦点的 ActionScript 方法包括：

- `myUIComponent.focused = 1;` // 其中 1 是索引 1 处的控制器的一个位。
- `stage.focus = myMovieClip;` // 只要带有此逻辑的 MovieClip 在 Stage 上。
- `FocusHandler.getInstance().setFocus(myMovieClip);` // FocusHandler 是一个静态类。

4.5.1 最佳使用方法

焦点必须在一个 **InteractiveDisplayObject** 上设置，否则焦点就会默认到播放器。如果未进行设置，焦点管理系统将不能正常工作（如 Tab 键不切换焦点等）。可以通过在任何组件上设置焦点属性为 `true` 使得焦点得以应用。另外一个应用焦点的方法，也是应用在非组件元素中，如下所示

将 **tabEnabled** 和 **mouseEnabled** 设置为 `false` 的 MovieClips 不能通过 Scaleform 或 Flash 播放器聚焦，而且不会生成焦点更改事件。默认情况下，多数 CLIK 组件为其本身及其子组件设置 **tabEnabled** 和 **mouseEnabled** 属性。

某些情况下，设计师可能希望某些组件同时具有 **tabEnabled** 和 **mouseEnabled** 属性，但却无法收到焦点；**UIComponent.focusable** 属性解决了这一问题，因为 AS3 中没有 `MovieClip.focusEnabled`。如果将此属性设置为 `false`，**UIComponent** 就无法通过 CLIK 焦点框架收到焦点。请注意，更改 **focusable** 属性将会在内部影响该组件的 **tabEnabled** 和 **mouseEnabled** 属性，因而无法维护以前应用到该组件的 **tabEnabled** 和 **mouseEnabled** 配置。

具有可获得焦点的子元素组件，如滚动条 **ScrollBar**，使用焦点目标属性传递焦点到它们自身组件。当组件焦点发生变化时，焦点句柄 **FocusHandler** 将递归搜索焦点目标链，将焦点传给上一个焦点不返回一个焦点目标。

有时当一个组件不是播放器或引用程序的实际焦点时需要出现一个焦点。**displayFocus** 属性可以设置为 `true` 使组件表现为获得焦点的样子。例如，当滑动条 **Slider** 获得焦点，滑动条轨道也需要获得焦点。注意当焦点属性发生改变时组件调用 **UIComponent.changeFocus()** 方法。

```
function changeFocus():void {
    track.displayFocus = _focused;
}
```

反过来，有时一个组件需要是可点击的，但它却不该通过 `tab` 接受焦点，例如，一个可拖曳的面板，或者将会受益于只鼠标控制的任何其它组件。在此情况下，用户可以将 `tabEnabled` 属性设置为 `false`。

```
background.tabEnabled = false;
```

4.5.2 在复合附件捕获焦点

复合组件为那些由其他组件组成，如 `ScrollingList`、`OptionStepper` 或 `ButtonBar`。组件本身可能拥有子组件的鼠标句柄，但是不具有自身鼠标句柄。意思为在 Flash 和 Scaleform 中的 Selection 引擎不支持项和内置导航焦点将无法识别组件，而错误得去检查其子组件。

使组件行为不受合成的约束作为一个独立的入口，需要以下步骤：

1. 在所有具有鼠标句柄的子组件上设置 `tabEnabled = mouseEnabled = focusable = false`（如 `OptionStepper` 中的箭头按钮）；
2. 在所有具有鼠标句柄的子组件上设置焦点目标属性。确保在容器组件中设置 `focusable= true`；
3. 如果有必要设置子组件中的 `displayFocus` 属性为 `true`，该属性位于容器组件 `changeFocus` 方法之内；

现在当子组件获得焦点，焦点将转递给容器组件。

4.6 输入处理

由于 Scaleform CLIK 组件从 Flash 8 MovieClip 类继承而来，当接收到用户输入时与任何其他 MovieClip 实例具有相同的行为。如果安装了鼠标句柄组件可以捕获鼠标事件。然而，在 CLIK 上相关键盘或类似空孩子气事件处理具有概念上的区别。

4.6.1 最佳使用方法

所有的非鼠标输入都可以被 `InputDelegate` 管理类 class (`scaleform.clik.managers.InputDelegate`) 截取，`InputDelegate` 将输入命令转换为内部的任意一个 `InputDetails` 对象(`scaleform.clik.ui.InputDetails`)，或者从游戏引擎请求输入命令的值。后者包括了修改 `InputDelegate.readInput()` 方法以支持目标应用程序。一旦 `InputDetails` 被创建，一个输入事件就从 `InputDelegate` 得到指派。

`InputDetails` 包含了以下属性：

- 类型，如“key”
- 编码，如按下键的键值
- 这是一个数值，它提供了诸如按钮矢量或按键类型等输入的附加信息。关键事件是由键的按下与弹起动作而产生的，相应的输入细节的数值参数也分别为 `InputValue.KEY_UP` (“keyUp”) 或 `InputValue.KEY_DOWN` (“keyDown”)；
- `navEquivalent(navigation equivalent)`，定义了可读的导航方向如 `NavigationCode.UP` (“up”), `NavigationCode.DOWN` (“down”), `NavigationCode.LEFT` (“left”), or `NavigationCode.RIGHT` (“right”),或者 “right”只要可以进行映射。`NavigationCode` 类 (`scaleform.clik.ui.NavigationCode`) 提供了一个手动通用导航枚举列表。

`FocusHandler` 对从 `InputDelegate` 发出的 `InputEvent` 进行侦听，然后从当前聚焦的组件重新发出同一个 `InputEvent`，以便由该组件或该事件的 `bubble` 链中的另一个组件进行处理。

获得焦点的组件用来决定焦点路径，为一个自上而下的组件列表，位于实现 `the handleInput()`方法的现实列表层次当中。

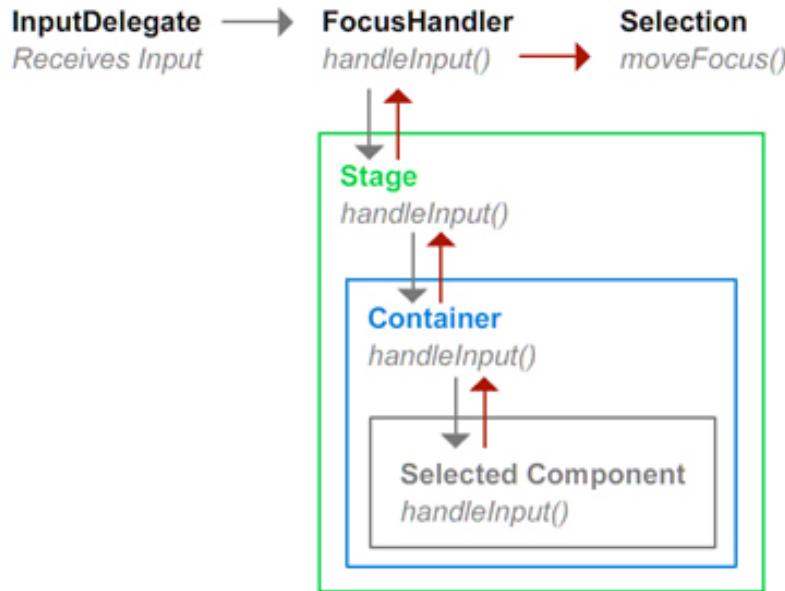


图 62: `handleInput()` 函数链

当 `InputEvent` 完成处理后，`handled` 属性应设置为 `true`。这使接收该事件的其它组件可以检查 `event.isDefaultPrevented()`，以识别该事件是否已被处理。

```

override public function handleInput(event:InputEvent):void {
    // Check whether the event has already been handled by another component.
    if (event.isDefaultPrevented()) { return; }
    var details:InputDetails = event.details;
  
```

```

        switch (details.navEquivalent) {
            case NavigationCode.ENTER:
                if (details.value == InputValue.KEY_DOWN) {
                    // Do something.
                    event.handled = true;
                }
                break;
            default:
                break;
        }
    }
}

```

`InputEvent` 将由 ActionScript 3 事件系统自动发泡 (bubble)，假定其 `.bubble` 属性未被修改。在某些情况下，如 `ScrollingList`，组件可能需要将 `InputEvent` 传递给其子组件，然后好尝试处理该事件本身。例如，假如按了 Enter 键，`ListItemRenderer` 应生成一个 `ButtonEvent.CLICK` 事件，并且发出该事件的列表将不会做任何事情。

```

override public function handleInput(event:InputEvent):void {
    if (event.isDefaultPrevented()) { return; }
    // Pass on to selected renderer first
    var renderer:IListItemRenderer = getRendererAt(_selectedIndex,
_scrollPosition);
    if (renderer != null) {
        // Since we are just passing on the event, it won't bubble, and should
properly stopPropagation.
        renderer.handleInput(event);
        if (event.handled) { return; }
    }
    . .
}

```

也可以添加一个事件监听器到 `InputDelegate.instance` 中手动监听输入事件，并按照这种方法处理输入信息。注意这种情况下输入信息仍然可以被 `FocusHandler` 句柄捕获并传递到焦点层次。

```

InputDelegate.instance.addEventListener(InputEvent.INPUT, handleInput);
function handleInput(e:InputEvent):void {
    var details:InputDetails = e.details;
    if (details.value = Key.TAB) { doSomething(); }
}

```

`InputDelegate` 应该在游戏中用来管理预期输入信息。默认 `InputDelegate` 处理键盘控制、方向键转换以及通用游戏导航键 W、A、S 和 D 直接转换到相等的导航键。

4.6.2 多鼠标光标

在一些系统中，如 `Nintendo Wii™`，支持多光标设备。`CLIK` 组件框架支持多个光标，但是不允许在单个 `SWF` 文件中多个组件获得焦点。如果两个用户都点击独立的按钮，最后的点击项即为焦点项。

所有在框架中指派的鼠标事件包含了一个鼠标索引属性，作为产生事件的输入设备索引。

```
myButton.addEventListener(ButtonEvent.CLICK, onCursorClick);
function onCursorClick(event:ButtonEvent):void {
    trace(event.controllerIdx);
}
```

4.7 失效

失效为组件使用的一种机制，用来限制组件在多个属性发生变化时候组件的刷新次数。当一个组件为失效状态，在下一帧将重绘组件。这使开发者可以暂时忽略组件发生的改变，组件只要在特定时间更新一次即可。在有些情况下组件需要立即更新；但是大多数情况下失效性非常有用。

```
btnInstance.setSize(100,22);
btnInstance.width = 200;
btnInstance.label = "text";
btnInstance.toggle = true;
btnInstance.selected = true;
btnInstance.data = obj;
```

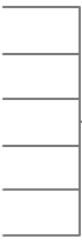


图 63: CLIK 组件当特定属性改变时自动变为无效

4.7.1 最佳使用方法

在任何内部组件发生改变后（通常由设置功能引起），则调用 `UIComponent.invalidate()` 函数，该函数最终调用 `UIComponent.draw()` 函数重绘组件。

为了实现最佳性能，`draw()` 内的逻辑被分为若干小节，它们由 `isInvalid()` 检验包装，这些检验识别该组件的哪些方面（例如，数据、大小、状态等）当前无法避免更新该组件尚未更改的部分。调用 `invalidate()` 时，该方法使该组件的各个方面无效，迫使在 `draw()` 内进行一次完整重绘。因此，为了实现最佳性能，子类应通过在 `isInvalid()` 检验内包装 `draw()` 逻辑来遵循同一范式，而且并非调用 `invalidate()`，而是使用专用的 `invalidate()` 方法（如 `invalidateData()`, `invalidateSize()`）来避免不必要的重绘。

`invalidate()` 方法在下一阶段无效 (`Event.RENDER`) 或下一帧 (`Event.ENTER_FRAME`) 上生成 `draw()` 调用，以便于该组件将多项更改批处理到单个 `draw()` 调用之中。

在需要立即重绘组件的情况下，开发者可以使用 `UIComponent.validateNow()` 方法函数。请注意，如果该组件中以前什么也没有标为无效，就不会调用 `draw()`。

4.8 组件缩放

Scaleform CLIK 组件按照两种方式进行缩放：

1. 使用重复绘制，重新安排组件尺寸，组件元素进行缩放适合原来的比例尺寸。具有子党员并没有背景的组件采用此方法。
2. 元素不进行缩放维持纵横比例关系，组件进行缩放，但是元素不缩放。该方法是用在有图形背景的组件中，`scale9grid` 进行了伸展，内部的文本缩放而不扭曲。

由于受到 Flash `scale9Grid` 的限制，组件通常使用回流方法。这要求开发者建立“皮肤”标识，类似于 Flash/Flex 组件。如果部件拥有可以放缩的子元素，则回流方法的效果最好，因此它应用于 `container` 和类似实体。

反缩放方法专门为 CLIK 创建，主要是由于 Scaleform 中的 `scale9Grid` 扩展功能。它允许利用帧状态创建一个单一资产组件，而无需使用设置其他组件时所用的密集分层方法。基本的 CLIK 组件都具有简约的特性，通常包括一个背景、一个标签和一个可选图标或子按钮，因此非常适合反缩放方法。然而，反缩放并不打算进行类 `container` 设置（面板设计等等）。在这种情况下，我们建议使用回流方法，因为它具有约束其余子元素的背景。

组件可以在 Flash IDE 的场景中进行缩放，或者使用宽度和高度属性进行动态缩放，或者使用 `setSize()` 方法函数。缩放后的组件外观可能在 Flash IDE 中不是那么精确。这就是其局限性，组件作为未编译的视频剪辑 MovieClips 不能进行实时预览 LivePreviews。在 Scaleform Player 中测试动画为确认缩放组件在游戏中外观是否精确的唯一方法。CLIK 扩展了一个启动面板改进了这项工作流。

4.8.1 Scale9Grid

多数组件使用第二种缩放方法。Scaleform Player 中的 Scale9Grid 视频剪辑 MovieClip 资源在多数部分都附带了 `scale9grid`，尽管 Flash 在包含了 MovieClips 的情况下将丢弃栅格。这意味着即使 `scale9grid` 在 Flash Player 中不能工作，在 Scaleform 中可能可以很好发挥作用。

需要注意的一点是当视频剪辑 MovieClip 使用了 `scale9grid`，其子单元也将根据此规则进行缩放。增加 Scale9grid 到子单元将导致忽略其上级栅格，正常进行绘制。

4.8.2 强制

`Constraints` 类(`scaleform.clik.utils.Constraints`)辅助缩放组件内部的资源缩放和定位。使开发者在场景中定位资源，使资源保持到上级组件边缘的距离。例如，滚动条 `ScrollBar` 组件将根据在场景中的位置重新缩放轨道大小及其位置。`Constraints` 在两种组件缩放方法中都被用到。

以下代码增加滚动条 ScrollBar 资源到 configUI()方法的强制对象，向下方向键底部对齐，缩放轨道根据其上级组件进行延伸。draw()方法函数包含了代码以更新强制对象，从而任何元素都用此进行登记。这项更新在 draw()函数中完成，因为在组件无效时被调用，通常在组件尺寸发生改变后。

```
override protected function initialize():void {
    super.initialize();

    // The upArrow doesn't need a constraints, since it sticks to top left.
    if (downArrow) {
        constraints.addElement("downArrow", downArrow, Constraints.BOTTOM);
    }
    constraints.addElement("track", track, Constraints.TOP |
                                Constraints.BOTTOM);
}

override protected function preInitialize():void {
    constraints = new Constraints(this, ConstrainMode.REFLOW);
}

protected function draw():void {
    constraints.update(_width, _height);
}
```

4.9 组件和数据设置

组件需要使用一个数据源 `dataProvider` 方法的列表数据。数据源 `dataProvider` 为一个数据存储和对象检索单元，开放了 Scaleform CLIK `IDataProvider` 类(`scaleform.clik.interfaces.IDataProvider`)中定义的所有或部分 API 函数。

数据源 `dataProvider` 方法是用一个调用函数的请求模型，代替直接的属性访问。这允许数据源在需要时可以从游戏引擎获取数据。这具有存储优势，也可以将大块数据设置为小块，易于管理的数据。

组件使用数据源 `dataProvider` 包括任何扩展的 `CoreList` (`ScrollingList`, `TileList`)、`OptionStepper` 和 `DropdownMenu`。

4.9.1 最佳使用方法

数据源 `DataProvider` 类(`scaleform.clik.data.DataProvider`)包含在框架中使用其静态 `initialize()`方法，将数据源 `dataProvider` 方法添加到任何 ActionScript 序列中去。组件使用一个数据源 `dataProvider` 将自动初始化序列使他们可以使用 `dataProvider` 方法进行访问。这意味着下列语法初始化一个静态声明序列作为完全可操作的数据源 `dataProvider`，以及在 `IDataProvider` 中描述的方法。

```
myComponent.dataProvider = new DataProvider([ "data1",
                                              4.3,
                                              {label:"anObjectElement", value:6}]);
```

数据源 `dataProvider` 应该实现内容为：

- `length`: 可以作为一个属性或获取函数返回数据设置的长度；
- `requestItemAt`: 从数据源 `dataProvider` 请求一个指定项，通常被列表组件使用以在任何时间显示一个项，如 `OptionStepper`；
- `requestItemRange`: 从数据源 `dataProvider` 请求一系列项包括一个开始和结束序号。通常由列表组件使用以显示更多项，如 `ScrollingList`；
- `indexOf`: 返回项的序号；
- `invalidate`: 标记数据源 `dataProvider` 变化，提供一个新的数据长度。该方法也应该指派一个“`dataChange`”事件来通知组件数据已更新。数据源 `dataProvider` 应该支持一个可以公开访问和反应数据长度的长度属性。

实例需要数据简介列表能够使用一个数据作为数据源 `dataProvider`。开发者应该明白在 ActionScript 2 中存储数据比在应用程序中自带数据需要更多的存储空间和性能开销。在大量数据设置推荐绑定数据源 `dataProvider` 和 `ExternalInterface.call` 基于需求从游戏引擎获取数据。

4.10 动态动画

Scaleform CLIK 提供一个自定义的 `Tween` 类 (`scaleform.clik.motion.Tween`)，该类具有与任何可比 `Tween` 类相似的行为，但却完全兼容 Scaleform。CLIK `Tween` 支持所有标准缓动函数，例如，`fl.transitions.easing.*` 包装下的缓动函数。

要启动一个 `tween`，请创建一个新的 `Tween()`，并提供一个持续时间（以毫秒为单位）、目标 `Sprite`、要 `tween` 的属性和这些属性应 `tween` 到的值，以及一个包含作为附加 `Tween` 参数的属性的对象。支持下列属性/参数：

- `ease`: `Function` (函数) 类型。缓动函数。
- `easeParam`: `Object` (对象) 类型。您可以传递到缓动函数的一个额外参数。
- `onComplete`: `Function` 类型。`Tween` 完成时将会调用此关闭 (closure)。
- `onChange`: `Function` 类型。当 `Tween` 更新其值 (每个刻度/帧 (tick/frame)) 时会调用此关闭。
- `data`: `Object` 类型。您需要附加到 `Tween` 的任何自定义数据 (这丝毫不会影响 `Tween` 的行为)。
- `nextTween`: `Tween` 类型。当您想要链接另一个 `Tween` 时使用。当前 `Tween` 完成时，就会把链接的 `Tween` 设置为 `pause=false`。
- `frameBased`: `Boolean` 类型。使用基于帧的定时，而不是实时。
- `delay`: `Number` 类型。延迟 `tween` x 毫秒。
- `fastTransform`: `Boolean`。使用显示对象属性的矩阵数学。

- **paused:** Boolean 类型。是否暂停 Tween。

然而 Tween 方法支持单个 MovieClip 的多个属性，允许同一个对象的不同属性在相同时间发生变化。下节中的例子展示了如何创建 Tweens 在同一时间影响多个属性。

4.10.1 最佳使用方法

Tween 的一个实例将会把目标 MovieClip 从该 MovieClip 的当前属性值 tween 到函数参数列表中指定的一个值。

要想获知 tween 完成时间，只需创建一个函数引用，并将其就像在 onComplete 属性中一样添加到作为 Tween 的构造函数的第四个参数传递的对象。此回调会在 Tween 完成时被调用并传递到一个参数，调用该参数的 Tween。

```
import fl.transitions.easing.*;
import scaleform.clik.motion.Tween;

// Perform a 1 second tween from the current horizontal position and
// alpha values to the ones specified
var myTween:Tween = new Tween(300,
                               myMovieClip,
                               { x:100, y:100, alpha:0 },
                               { ease:Strong.easeOut,
                                 onComplete:onCompleteCallback } );

function onCompleteCallback(t:Tween):void { // Do something. }
```

4.11 布局框架

CLIK 布局框架旨在帮助开发者创建涉及多个元素并容易适应多种分辨率的动态布局。虽然有点简单化，CLIK 布局框架还是应帮助开发者创建针对多种分辨率和平台的 UI，同时为在 CLIK 范围内开发自定义布局系统打下基础。

注意：Scaleform v4.0.14 是 CLIK AS3 布局框架的初始版本，而且其中的类和 API 可能随时更改。如果您有关于 CLIK AS3 布局框架的任何疑问、问题、建议或意见，请随时通过 Scaleform 开发者中心打开一张咨询单。

两个类构成 CLIK 布局框架的核心：

- **scaleform.clik.layout.Layout:** 一个将管理任何 Sprite 的布局的布局，这些 Sprite 在同一父属性内正确实现 .layoutData 属性。

- **scaleform.clik.layout.LayoutData:** 包含有关应如何对某个 **Sprite** 进行布局的信息的数据结构。此数据由相关布局实例读取，并用来创建该布局。

在 CLIK 内创建一个新布局的方法有多种。下面介绍创建一个新布局的最简单的方法：

1. 创建 **Sprite/MovieClip** 并将其添加到 **Stage** 上的一个 **DisplayObjectContainer**。
2. 用 **LayoutData** 类的新实例为这些 **Sprite/MovieClip** 定义 **.layoutData** 属性。
3. 根据需要填写针对每个 **Sprite/MovieClip** 的 **LayoutData** 对象。
4. 创建 **Layout** 类的一个新实例。将其添加到前面第 1 步中 **Sprites/MovieClips** 使用的同一 **DisplayObjectContainer**。

在此情况下，假如 **Layout**（布局）实例的任何属性均未被修改，该 **Layout** 将会管理父辈内的所有 **Sprite** 和 **MovieClip**，这些 **Sprite** 和 **MovieClip** 使用该父辈的宽度和高度作为用于定位的矩形（**x**、**y**、宽度和高度），来定义 **.layoutData** 属性。

Scaleform\Resources\AS3\CLIK\demos 目录中的 SDK 中包含有此布局框架的两个演示。第一个演示 **Layout_Main** 可通过 CLIK 组件浏览器使用，该浏览器可通过 Scaleform SDK 浏览器打开。
Layout_Main 演示在一个可重新调整大小的 **MovieClip** 内创建一个简单的布局。另一个演示 **Layout_FullScreen_Demo** 设计为在 **FxPlayer** 的一个独立实例中运行，以便于其使用 **Stage** 的完整大小。
Layout_FullScreen_Demo 演示用户如何将其全屏内容用于多种分辨率以及 **Layout** 系统如何处理各种参数。

4.11.1 布局

布局 (**Layout**) 是一个组件，它可用来对可按大小和/或比例发生变化的上下文内的布局元素进行布局。布局只能管理同一父辈内存在的 **Sprite**（以及其中的任何子类）。也就是说，任何元素都不应作为布局元素本身的一个子实例进行添加，但可以在该布局实例的同一层级添加。例如，在同一层级拥有两个子实例 (**myWindow.myMovieClip** 和 **myWindow.layoutInstance**) 就允许 **layoutInstance** 管理 **myMovieClip**。

布局仅管理 **Sprite**，该 **Sprite** 用 **LayoutData** 类的一个实例定义了 **.layoutData** 属性。多个布局可以存在于同一父辈内，条件是，每个布局都具有一个唯一的 **.identifier** 属性集，而且同一层级 **Sprite** 内的所有 **LayoutData** 还定义其 **.layoutIdentifier** 属性。

当把布局添加到 **Stage** 时，布局将查看一个父辈内的所有子项，并检查这些子项是否定义了 **.layoutData**。从此以后，Layout 就会跟踪添加到该父辈的新子项，并且，如果它们在被添加到 **Stage** 之前定义 **.layoutData**，Layout 就会将它们添加到其受到管理的 **Sprite/MovieClip** 列表。请注意，受到管理的 **Sprite/MovieClip** 偏移量以及它们的布局顺序仅在初次将其添加到 **Stage** 时更新。假如用户想要重新计算从一个已更新位置的偏移量，或者对列表进行重新排序，他们可以使用下列函数：

public function reset():void

通过清空其受到管理的 Sprite 的列表、搜索包含 LayoutData 的新 Sprite 并重新计算偏移量 / 从头回流那些新的 Sprite，来重新设置 Layout。

public function resortManagedSprites():void

按其 layoutData.layoutIndex 属性对受到管理的 Sprite 进行排序，该属性定义应用布局的顺序。如果最初设置布局之后任何 layoutIndex 被更改，就应调用此函数。

用户可以将 Layout 类附加到其 Flash 文件的库中的符号 (Symbol) 上，以便于美术师在设计时将布局放在 Stage 上。假如没有将 Layout 连接到一个 Symbol（而是通过宽度和高度均为 0 的代码进行了实例化），该 Layout 的大小将默认设置为父辈的大小。这可以通过设置 Layout 的 .rect 属性进行更改，该属性定义 Layout 用来对其受到管理的元素进行布局的区域。

理想的情况是，Layout 应作为最后一个元素添加到父辈，以确保已经正确定义了其它 Sprite/MovieClip 的所有 LayoutData。

4.11.1.1 公共属性

Rect	布局的“大小”。里面的元素将按照此属性的 x、y、宽度和高度进行布局。假如 Layout 的宽度 != 0 (联接到一个 Symbol，并放置到 Stage 上)，则 Layout 将使用 MovieClip 的 x、y、宽度和高度。假如 Layout 的宽度 == 0 (addChild() 不包含一个支持 Symbol)，则 Layout 将使用父项的 x、y、宽度和高度。您也可以使用 .rect 属性指定一个自定义的 Rectangle (矩形)。
tiedToStageSize	为 true – 假如此 Layout 始终更新以匹配 Stage 大小，否则为 false。
tiedToParent	true – 假如此 Layout 的大小始终更新以匹配其父辈的大小，否则为 false。
Hidden	为 true - 假如在运行时隐藏此 Layout，否则就是 false。允许 Layout 拥有一个可见背景或占位符图像，该图像将在运行时立即被设置为 visible = false。

4.11.2 LayoutData

定义某个特定 Sprite 的布局的数据。必须与一个有效的 scaleform.clik.layout.Layout 实例一起使用。

4.11.2.1 公共属性

alignH	此 Sprite 的水平对齐。有效值：LayoutMode.ALIGN_NONE、LayoutMode.ALIGN_LEFT、LayoutMode.ALIGN_RIGHT。
alignV	此 Sprite 的垂直对齐。有效值：LayoutMode.ALIGN_NONE、LayoutMode.TOP、LayoutMode.BOTTOM.property 到 TextFieldAutoSize.NONE 将保持大小不变。

offsetH	从 Layout 或 relativeToH Sprite 的边缘的水平偏移量。如果该偏移量保持不变 (-1), Layout 就会自动将其设置为从 Stage 上的 Layout/Sprite 的原始水平偏移量 (由美术师在设计时定义)。
offsetV	从 Layout 或 relativeToH Sprite 的边缘的垂直偏移量。如果该偏移量保持不变 (-1), Layout 就会自动将其设置为从 Stage 上的 Layout/Sprite 的原始垂直偏移量 (由美术师在设计时定义)。
offsetHashH	一个哈希表 (Hash Table), 包含用户定义的适用于各种长宽比的水平偏移量。如果将 Layout 联接到 Stage 的大小 (Layout.tiedToStageSize == true), 就会查询这些值。例如, 假如 Layout 联接到 Stage 的大小, 而且长宽比当前为 4:3, "LayoutData.offsetHashH[LayoutData.ASPECT_RATIO_4_3] = 70;" 将会导致此 Sprite 使用 70px 的水平偏移量。
offsetHashV	一个哈希表 (Hash Table), 包含用户定义的适用于各种长宽比的垂直偏移量。如果将 Layout 联接到 Stage 的大小 (Layout.tiedToStageSize == true), 就会查询这些值。例如, 假如 Layout 联接到 Stage 的大小, 而且长宽比当前为 4:3, "LayoutData.offsetHashV[LayoutData.ASPECT_RATIO_4_3] = 70;" 将会导致此 Sprite 使用 70px 的垂直偏移量。
relativeToH	此 Sprite 应与其水平相关的 Sprite 的实例名称。假如保留为 null, Sprite 就会相对于 Layout 而对齐。
relativeToV	此 Sprite 应与其垂直相关的 Sprite 的实例名称。假如保留为 null, Sprite 就会相对于 Layout 而对齐。
layoutIndex	控制相对于同一 Layout 中的其它 Sprite 应对此 Sprite 进行布局的顺序的索引。应设置 – 假如使用 relativeToH 或 relativeToV 确保在此 Sprite 之前更新该 Sprite 的 Layout。假如 layoutIndex 保留不变 (-1), 就会任意将其添加到列表末尾。
layoutIdentifier	定义应将此 LayoutData 对象与哪个 Layout 关联 (假如单个 DisplayObjectContainer 内存在多个 Layout) 的字符串。假如保留不变 (null), 就会自动由同一父辈内的所有 Layout 对其进行管理。此属性 (如果设置了的话) 应匹配一个 Layout 实例的标识符属性。

4.12 弹出式支持

Scaleform CLIK 包含了 PopUpManager 类(scaleform.clik.managers.PopUpManager), 支持弹出视窗如对话框和提示工具。

4.12.1 最佳使用方法

PopUpManager 拥有几个静态方法辅助弹出视窗的创建和维护。createPopUp()方法能够被用到任何视频剪辑 MovieClip 符号 (包括 CLIK 组件) 的链接 ID 创建弹出视窗。第一个参数是一个用于唯一链接 ID 的字符串。第二个参数 initProperties 允许您提供一个对象, 该对象的属性将会被复制到创建的新 PopUp。第三和第四个参数分别针对相对于该弹出窗口的新弹出窗口定义 x 和 y 坐标。relativeTo 参数, Sprite 类型, 可用来定位相对于另一个 Sprite 的弹出窗口。

```
import scaleform.clik.managers.PopUpManager;
```

```
PopUpManager.createPopUp( "PopupLinkageID", {alpha:.5}, 100, 100  
myMovieClip);
```

Scaleform 扩展 InteractiveObjectEx.setTopmostLevel()参数用来保证弹出视窗总是显示在场景中所有其他部件的上面，由于与库相关的问题，使用此方案替代在 root 层创建弹出窗口。如果导入一个子 SWF 文件到上级组件，试图创建一个在上级内容所在的路径的库定义的符号实例，则将产生符号查找错误，因为上级组件无法访问子元素的库。不幸的是，没有专门工作区域来解决这个问题，因此 topmostLevel 方案为最佳选择。

注意 InteractiveObjectEx.setTopmostLevel()也能应用到非弹出视窗，在场景中保持此类元素的 Z 轴序列。因此，在弹出菜单顶部绘制鼠标光标，光标可以创建在最高处位置或层次并设置 topmostLevel 属性为 true。

4.13 拖放

DragManager 类 (scaleform.clik.managers.DragManager)支持初始化和拖放操作。该类是一单个组件，提供了一个实例属性访问唯一的对象。使用此对象，开发者可以开始和结束拖放操作。

要启动一个拖曳，请发出一个具有与新拖曳相关的属性的 DragEvent.DRAG_STATE。

DragManager.handleStartDragEvent() 将会收到该事件，并根据该事件的属性启动此拖曳。然后，此拖曳将会由 DragManager 进行管理，直到收到一个 MouseEvent.MOUSE_UP 事件，因而导致调用 DragManager.handleEndDragEvent()。

4.13.1 最佳使用方法



图 64:动作中的 DragDemo

DragManager 依靠 IDragSlot 接口，该接口定义通过 UI 高效地与可用 IDragSlot 进行通信的函数。 Scaleform CLIK 包括一个例子为拖放目标类名为 DragTarget (scaleform.clik.controls.DragTarget)。 DragTarget 扩展 UIComponent 因此分类为一个 CLIK 组件。拥有几个唯一特性作为 DragManager 的补充。

Scaleform CLIK 包括一个示例 IDragSlot 实现，称为 DragSlot (scaleform.clik.controls.DragSlot)。为实现这些拖动类型，DragSlot 也需要随 DragManager 为 dragBegin 和 dragEnd 安装事件监听器。这使得 DragTarget 可以显示其是否支持拖放操作。DragSlot 设计为包含一个将会被拖动的 MovieClip/Sprite/Bitmap。不过，DragSlot 也模拟按钮的许多行为，这样使得 DragSlot 也可用作一个按钮。

CLIK 捆绑的演示文件中使用两个组件，作为子类或者组成 DragSlot 来展示使用 DragManager 和 DragSlot 进行的拖放操作。这两个组件为 InventorySlot 和 ItemSlot，这些组件类作为 CLIK 框架的一部分提供，但是，注意他们只是作为一个例子表示可以实现的功能，并不是一个所有应用场合下的完整解决方案。

归根结底，DragSlot 只是 IDragSlot 的一个实现示例，它的许多函数已被废止，取代以更密切地与数据后端进行通信的子类。要想了解一个功能齐全的 DragSlot 子类的示例，请查看 MMO UI 工具箱，该工具箱将 DragSlot 用作整个 UI 内的可拖曳内容的基类。

5 实例

下面小节包括一些使用 Scaleform CLIK 组件的例子。

5.1 基础

一下实例比较简单，但展示了易于使用的 Scaleform CLIK 框架执行常规任务。

5.1.1 包含两个 `textField` 的 `ListItem Renderer`



图 65: 滚动列表 `ScrollingList` 展示包含两个标签的列表项

滚动列表组件默认使用 `ListItemRenderer` 来显示行内容。然而 `ListItemRenderer` 只支持单个文本区域 `textField`。很多情况下列表项需要多个文本区域 `textField` 用来显示，或者甚至不需要文本区域如图标。本例展示了如何添加两个文本区域到列表项。

首先，定义需求，对象为一个自定义 `ListItemRenderer` 支持两个文本区域。自定义 `ListItemRenderer` 应该与滚动列表 `ScrollingList` 相兼容。由于目的为使得每个列表项具有两个文本区域 `textFields`，数据也应该不止单个字符串列表。本例中我们使用以下数据源 `dataProvider`:

```
list.dataProvider = [{fname: "Michael", lname: "Jordan"},  
                    {fname: "Roger", lname: "Federer"},  
                    {fname: "Michael", lname: "Schumacher"},  
                    {fname: "Tiger", lname: "Woods"},  
                    {fname: "Babe", lname: "Ruth"},  
                    {fname: "Wayne", lname: "Gretzky"},  
                    {fname: "Usain", lname: "Bolt"}];
```

数据源包含的对象具有两个属性：`fname` 和 `lname`。该两个属性将显示在两个列表项的文本区域 `textField` 中。

由于默认的 `ListItemRenderer` 只支持一个文本区域 `textField`，需要能够支持两个文本区域 `textFields`。最简单的实现方法为将 `ListItemRenderer` 类作为子类。一下为调用 `MyItemRenderer` 类的源代码，其中使

用了 `ListItemRenderer` 子集并添加了两个文本区域的基本功能支持。（代码位于 `MyItemRenderer.as` 文件中，文件位于 FLA 工作目录）：

```
import scaleform.clik.controls.ListItemRenderer;

class MyItemRenderer extends ListItemRenderer {

    public var textField1:TextField;
    public var textField2:TextField;

    public function MyItemRenderer() { super(); }

    override public function setData(data:Object):void {
        this.data = data;
        textField1.text = data ? data.fname : "";
        textField2.text = data ? data.lname : "";
    }

    override protected function updateAfterStateChange():void {
        super.updateAfterStateChange();
        textField1.text = data ? data.fname : "";
        textField2.text = data ? data.lname : "";
    }
}
```

`ListItemRenderer` 的 `setData` 和 `updateAfterStateChange` 方法在本子类中没有涉及。`setData` 方法在列表项收到来自容器列表组件(`ScrollingList`、`TileList` 等)项数据时被调用。在 `ListItemRenderer` 中，本方法设置一个 `textField` 的值，而 `MyItemRenderer` 用来设置 `textField` 的值，同时也存储一个索引到内部项目对象。此项目对象在 `updateAfterStateChange` 方法中被重用，此方法在 `ListItemRenderer` 状态发生改变时被调用，此状态的变化需要文本区域 `textField` 刷新值。

到此，已经定义了具有复杂列表项支持列表项渲染的 `ListItemRenderer` 类的数据源 `dataProvider`。要连接所有相关列表组件，必须创建一个符号以支持该型的 `ListItemRenderer` 类。本例中，最快的实现方法为修改 `ListItemRenderer` 符号使其具有两个文本区域 `textFild` 调用‘`textField1`’和‘`textField2`’。下一步该符号标识和类必须修改为 `MyItemRenderer`。为在列表中使用 `MyItemRenderer` 组件，修改列表实例的 `itemRenderer` 检查属性，从‘`ListItemRenderer`’改变到‘`MyItemRenderer`’。

现在运行 FLA，列表应该显示出来包含列表元素显示两个标签：在数据源 `dataProvider` 中设置的 `fname` 和 `lname` 属性。

5.1.2 像素滚动视图

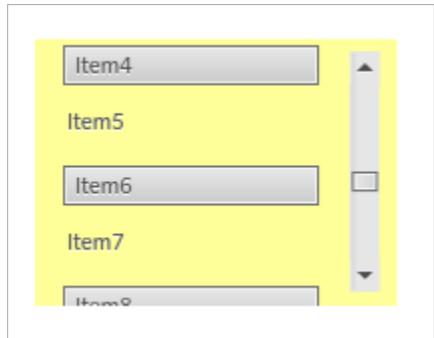


图 66: 按像素滚动包含 CLIK 元素的视图

按像素滚动通常用户复杂用户界面。本例展现了如何使用 CLIK 滚动条 ScrollBar 组件简单得实现此功能。

首先，创建一个新的符号用于滚动视图，这提供了一个容器，简化了所需的偏移量计算。在滚动视图容器中，从顶部到底部顺序设置一下图层。这些图层并不需要，但是用来作为说明：

- **actions:** 包含 ActionScript 代码使得滚动视图工作；
- **scrollbar:** 包含一个 CLIK 滚动条 ScrollBar 实例，该实例称为‘sb’；
- **mask:** 由矩形或 MovieClip 定义的蒙板图层，设置图层属性为一个蒙板；
- **content:** 包含需要滚动的内容；
- **background:** 可选层包含了一个背景突出无蒙板区域。

添加相关元素到图层并创建一个符号将其保存。通过创建一个符号保存所有的内容，滚动内容的任务变得十分简单。调用这些内容实例‘content’并设置 y 轴为 0。这确保滚动逻辑上不需要计算不同的偏移量。然而，这些偏移根据滚动视图的复杂性也许需要用到。

到此，定义了所需创建的一个简单滚动视图，以及需要互相关联的元素名称结构。将以下 ActionScript 代码放到 code 图层的第一帧：

```
// 133 is the view size (height of the mask)
sb.setScrollProperties(1, 0, content.height - 133);
sb.position = 0;

sb.addEventListener(Event.SCROLL, onScroll, false, 0, true);
function onScroll(e:Event) {
    content.y = -sb.position;
}
```

由于滚动条不与其他组件连接，需要手动配置。`setScrollProperties` 方法就是用来使其在一个位置滚动，并设置完全显示内容的最大值和最小值。本例中的滚动条位置为像素格式。运行文件，滚动视图可以通过与滚动条交互进行改变。

6 常见问题解答

1. 当我从 Flash Studio 发布时，我收到以下错误消息：

引用：

```
1152: A conflict exists with inherited definition scaleform.clik... in  
namespace public. (命名空间 public 中存在一个与继承的定义 scaleform.clik... 的冲  
突。)
```

您之所以收到此错误，是因为该类的某个元素被定义了两次。如果没有将一个 .FLA 设置为禁用 "Automatically Declare Stage Instances"（这会导致任何 ActionScript 定义与发布 .FLA 时 .FLA 自动生成的定义之间发生冲突），此错误消息最常出现。

CLIK 架构要求将此设置禁用才能使所有 CLIK 组件正常工作，因为通常要求明确定义 Stage 元素，以便于引用 ActionScript 类中的强类型子项。

您可以通过打开 File -> Publish Settings -> "Flash" tab -> Actionscript 3.0 Settings 并确保不选中 "Automatically declare stage instances"，来针对某个 .FLA 禁用 "Automatically Declare Stage Instances"。请注意，此设置不是 Flash Studio 全局设置，并且仅保存到特定 .FLA。

2. 我正从另一个 .FLA 的库中导入一个 CLIK 组件。当我将该组件的一个实例放在 Stage 上并更改其可检查 (inspectable) 属性时，收到以下错误：

代码：

```
1046: Type was not found or was not a compile-time constant: (类型未找到，或者  
不是一个编译时常数：) ...
```

当您的组件声明不适当时候最常发生此错误。一般情况下，您可以使用下列步骤解决此错误：

- 确保组件有一个有效的实例名称。
- 确保组件是在 ActionScript 中定义的。
- 确保 ActionScript 定义引用正确的类。有时，这可能不是组件的导出名称 – 而应该使用 ActionScript 类名称。例如，myButton:Button (scaleform.clik.controls.)，而不是 myButton:MyButtonSymbol，其中，MyButtonSymbol 是导入的符号的导出名称，而 myButton 是组件的实例名称。

3. 我的库中有两个组件，它们是从同一基类中派生而来的。当我发布我的 .SWF 文件时，我收到以下错误：

```
Symbol 'MySymbol', Layer 'actions', Frame 10, Line 1 -- 1024: Overriding a function  
that is not marked for override.
```

```
Symbol 'MySymbol', Layer 'actions', Frame 20, Line 1 -- 1024: Overriding a function  
that is not marked for override.
```

```
Symbol 'MySymbol', Layer 'actions', Frame 30, Line 1 -- 1024: Overriding a function  
that is not marked for override.
```

这些错误通常是由一个符号的类名称与其基类的名称相同导致的。例如，一个符号具有类“Button”和基类“scaleform.clik.controls.Button”。这会导致该符号的时间线定义与基类合并在一起。因此，共享同一基类的任何其它符号都会继承那些时间线定义，因而导致意外的行为。

要解决此问题，请检查了解是否存在一个基类，该基类有多个符号与其关联，然后确保没有任何符号的类名称匹配基类的名称。更改共享同一名称的符号的类就会解决此问题。

7 CLIK AS3 vs. CLIK AS2

This section outlines major differences, mostly programming related, between CLIK AS2 and CLIK AS3.

1. CLIK AS3 is designed to work in Flash Player.
2. Rather than using a custom EventDispatcher, CLIK AS3 uses ActionScript 3's native event system.
3. The invalidation system has been redesigned to help avoid unnecessary updates in draw(). Each component now stores an “invalid” table that tracks which aspects of the component are currently invalid. Default invalidation types are defined in scaleform.clik.constants.InvalidationType. Aspects can be marked as invalid using invalidate(), invalidate[InvalidationType](), or manually setting the property to true within the table.. Logic within draw() should be wrapped within an appropriate isInvalid() check to ensure that unnecessary updates to components are avoided.
4. invalidate() no longer uses a 1ms delay; instead, it uses the next stage invalidatation (Event.RENDER) or the next Event.ENTER_FRAME for the component and then calls validateNow().
5. Tween's syntax is no longer MovieClip.TweenTo; instead, var t:Tween = new Tween();
6. handleInput() is now tied to the native event system. This means that handleInput() now accepts one parameter of type InputEvent and the event will bubble up from the component that dispatched it rather than being passed down from FocusHandler.
7. handleInput() no longer returns a Boolean; instead, it should set event.handled = true; if the event was handled.
8. UIComponent has a new property, focusable, which can be used with the CLIK FocusManager to prevent focus from reaching a component. This is more or less a replacement for AS2's MovieClip.focusEnabled property.

9. enableInitCallback is no longer an inspectable of UIComponent. Instead, it should be set within a class's constructor or preInitialize() method.
10. DataProvider now requires that you provide the target array to DataProvider's constructor (`myComponent.dataProvider = new DataProvider(myArray);`), rather than simply setting `myComponent.dataProvider = myArray;`
11. ButtonBar will now only create Buttons that fit within its own size. If your ButtonBar's size is 200px wide and each Button is 150px wide, only one Button will be displayed. This allows users to resize their ButtonBar and have the Buttons within be added and removed dynamically.
12. UILoader has been removed. This functionality is now handled by the Flash MovieClip/Sprite and Loader classes.
13. UIComponent.SoundMap, introduced in CLIK 3.3, has been removed for CLIK AS3.