

Autodesk® Scaleform®

Scaleform CLIK AS3 用戶指南

本文包括了 Scaleform CLIK 框架及所帶內置元件執行方法的詳細使用說明

作者 : Prasad Silva, Matthew Doyle

版本 : 2.0

最後修訂 : 2010 年 8 月 19 號

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	Scaleform 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 簡介	1
1.1 概要	1
1.1.1. 在 Scaleform CLIK 包含的內容	1
1.2 UI 關注	2
1.3 元件理解	3
1.3.1 屬性檢查	3
1.4 框架基礎知識	4
1.4.1 事件	4
1.4.2 焦點	5
2 內置元件	6
2.1 基本按鈕和文本類型	6
2.1.1 Button	8
2.1.2 CheckBox	12
2.1.3 Label	17
2.1.4 TextInput	19
2.1.5 TextArea	23
2.2 分組類型	27
2.2.1 RadioButton	27
2.2.2 ButtonGroup	32
2.2.3 ButtonBar	34
2.3 滾動類型	37
2.3.1 ScrollIndicator	37
2.3.2 ScrollBar	39
2.3.3 Slider	42
2.4 列表類型	45
2.4.1 NumericStepper	46
2.4.2 OptionStepper	49
2.4.3 ListItemRenderer	52

2.4.4	ScrollingList.....	57
2.4.5	TileList.....	62
2.4.6	DropdownMenu.....	68
2.5	進度類型.....	73
2.5.1	StatusIndicator	73
2.6	其他類型.....	75
2.6.1	Window	76
3	藝術細節.....	79
3.1	最優方法.....	79
3.1.1	圖元圖像.....	79
3.1.2	蒙版	80
3.1.3	動畫	80
3.1.4	圖層和繪製元素	80
3.1.5	複雜皮膚介面.....	81
3.1.6	2 的指數倍	81
3.2	已知問題和推薦工作流程	81
3.2.1	複製元件.....	81
3.3	皮膚繪製實例.....	83
3.3.1	StatusIndicator 皮膚繪製	84
3.4	字體和本地化.....	86
3.4.1	概要	86
3.4.2	內置字體.....	86
3.4.3	在 textField 嵌入字體	87
3.4.4	本地化系統	87
4	編程詳述.....	88
4.1	UI 元件 UIComponent.....	88
4.1.1	初始化	89
4.2	元件狀態.....	89
4.2.1	按鈕元件.....	89

4.2.2	非按鈕交互元件	91
4.2.3	非交互元件	91
4.2.4	特殊案例.....	92
4.3	事件模型.....	92
4.3.1	最佳使用方法.....	92
4.4	運行時創建元件	93
4.5	焦點處理	94
4.5.1	最佳使用方法.....	94
4.5.2	在複合附件捕獲焦點	95
4.6	輸入處理	96
4.6.1	最佳使用方法.....	96
4.6.2	多滑鼠游標	98
4.7	失效	98
4.7.1	最佳使用方法.....	99
4.8	元件縮放	99
4.8.1	Scale9Grid.....	100
4.8.2	強制	100
4.9	元件和資料設置	101
4.9.1	最佳使用方法.....	101
4.10	動態動畫	102
4.10.1	最佳使用方法	103
4.11	佈局框架	103
4.11.1	佈局	104
4.11.2	LayoutData.....	105
4.12	彈出式支援	106
4.12.1	最佳使用方法	106
4.13	拖放	107
4.13.1	最佳使用方法	107
5	實例	109

5.1	基礎	109
5.1.1	包含兩個 textField 的 ListItem Renderer	109
5.1.2	圖元滾動視圖	111
6	常見問題解答	113
7	潛在缺陷	115
8	CLIK AS3 vs. CLIK AS2	116

1 簡介

本文檔提供了 Scaleform® 通用精簡介面工具（Common Lightweight Interface Kit，CLIK™）框架和元件的詳細使用說明。在深入認識 CLIK 用戶指南之前，希望開發者能夠先閱讀 [CLIK 入門](#) 和 [CLIK 按鈕入門](#)。這兩個使用指南介紹了創建和運行 Scaleform CLIK 所需要的步驟，介紹了基本的概念並提供了創建和美化 CLIK 元件的詳細指南。然而，專業級用戶更喜歡在學習入門文檔時直接查閱本文檔作為參考。

1.1 概要

Scaleform CLIK 為一組 ActionScript™ 2.0 (AS2) 用戶介面元素、庫和工作流增強工具集，Scaleform 4.0 用戶為遊戲控制器、PC 和掌上遊戲機應用快速實現豐富和高效的介面。框架的主要目標為構建一個精簡（依據記憶體和 CPU 使用量）、易於繪製皮膚並高度個性化的架構。另外，對於一個基本的 UI 內核類和系統基本框架，CLIK 包含了 15 個以上內置通用介面元素（例如，按鈕、捲軸、文本輸入框），將幫助開發者快速創建和重構用戶介面介面

1.1.1. 在 Scaleform CLIK 包含的內容

元件：簡單擴展內置 UI 控制元件

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

類：系統核心 API 函數

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

文檔和實例文件：

[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 關注

創建一個美觀的 **UI** 介面，第一步為在紙面或畫圖編輯器上如 **Microsoft Visio®** 上繪製草圖。第二步為在 **Flash** 中開始原型化 **UI** 介面，其目的為在專門圖形設計之前繪製出所有的介面項和流程圖。**Scaleform CLIK** 就是專門設計用於使用戶快速原型化和重構以完成整個過程。

構建一個完整的 **UI** 介面有很多不同的方法。在 **Flash** 中，不存在頁的概念，而在 **Visio** 或其他流程圖繪製軟體中有此概念，這裏使用了關鍵幀和動畫剪輯代替。如果所有的頁面都在同一個 **Flash** 文件中，文件將佔據更多記憶體，但是在頁間切換更加快速且容易。如果每個頁分別在單獨的文件內存放，整體的記憶體使用量可以降低，但是導入時間更長。將每個頁作為單獨的文件也具有一些優勢，可以使多個設計師和美工人員同時處理相同介面。而且作為技術和設計的需求，在決定 **UI** 專案結構時可以確保考慮到工作流程。

與傳統的桌面或 **web** 應用不同，這裏特別需要瞭解可以有很多種不同的方法來創建豐富的多媒體遊戲介面。每個引擎，甚至不同的平臺，實現方法也有所不同，有的方法使用起來更加高效，而有的則更加低效。例如，放置一個多頁介面到單個 **Flash** 文件。這樣管理起來更加方便，轉換操作也更加容易，但是增加了記憶體的消耗，對於老的遊戲控制器或移動終端則非常不利。如果一切都在單個 **Flash** 文件中進行，則不能使多個設計師同時工作在同一個介面的不同部分。如果專案為一個複雜的介面設計，需要一個龐大的設計團隊，或者具有其他特殊的技術和設計需求，則最好將每個 **UI** 頁面放置到不同的 **Flash** 文件。在很多情況下，這兩種方案都是可行的，但是，在特定的專案中，其中一種可能比另外一種方案更具有優勢。例如，螢幕可能需要根據需求導入和導出，或者在網路上動態更新和傳輸。衡量的底線為應該仔細評估 **UI** 介面資源的管理，從 **UI** 的邏輯規劃到工作流實現以及性能都需要考慮完善。

雖然 **Scaleform** 強烈建議開發者使用 **Flash** 和 **ActionScript** 作為 **UI** 介面主要開發手段，但沒有完美的答案，某些團隊也許更喜歡用應用引擎腳本語言（如 **Lua** 或 **UnrealScript**）來完成大多數繁重工作。在這些情況下，**Flash** 應該被主要用作動畫實現工具，只包含極少數量的動態腳本 **ActionScript** 和 **Flash** 文件內部的交互內容。

在早期瞭解技術、設計和工作流程的需求非常重要，然後繼續評估整個過程的整體方法，特別是在深入專案之前，確保順利和最終的成功。

1.3 元件理解

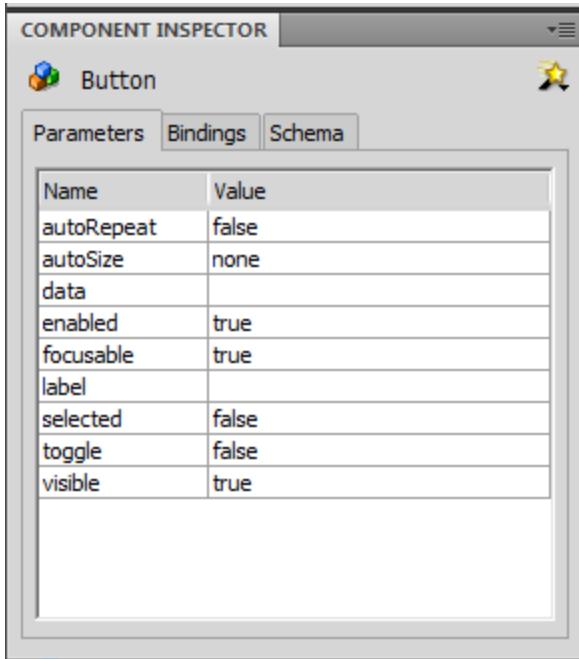
在開始之前，開發者需要理解 Flash 元件的確切技術細節。Flash 的一系列默認介面創建工具和編譯塊我們稱之為元件。但是，在本文檔中“元件”指使用 Scaleform CLIK 框架創建的內置元件，這些元件由 Scaleform 與世界知名的 gskinner.com 團隊聯合開發。

gskinner.com 由 Grant Skinner 領導，Grant Skinner 由 Adobe 任命負責為 Flash Creative Suite® 4 (CS4) 創建元件，為世界知名的 Flash 領先開發團隊之一。關於 gskinner.com 的更多資訊，請訪問 <http://gskinner.com/blog>。

為更深入理解內置 CLIK 元件，在 Flash studio 中開打默認的 CLIK 文件：
Scaleform SDK\Resources\AS3\CLIK\components\CLIK_Components_AS3.fla

1.3.1 屬性檢查

元件的重要屬性可以通過 Flash IDE 的 Component Inspector 面板或者 Parameters 標簽進行設置。需要在 CS4 中打開該面板，在上方工具條中選擇 Window 下拉功能表，點擊啟動 Component Inspector 視窗，或者按下(**Shift+F7**)鍵。這將打開 Component Inspector 面板。這些被稱之為“檢查屬性”。這為不熟悉 AS 編程美工和設計師配置元件行為和功能提供了一種便利的方法。



改變屬性只能在發佈包含該元件的 SWF 文件後才有效。Flash IDE 在設計階段場景中不顯示任何改變，因為 CLIK 元件不屬於編譯剪輯。這是用來確保元件易於使用和繪製皮膚。

1.4 框架基礎知識

1.4.1 事件

多陣列件產生進行使用者互動、狀態更改和焦點管理的事件。這些事件對於使用 CLIK 元件創建功能性使用者介面來說非常重要。

所有事件回檔均會收到單個 Event(事件)或 Event 子類參數,該參數包含有相應事件的相關資訊。下列屬性是所有事件所共有的。

- **type** : 事件類型。
- **target** : 事件目標。
- **currentTarget** : 通過一個事件偵聽器積極處理 Event 物件的物件。
- **eventPhase** : 事件流中的當前階段。（EventPhase.CAPTURING_PHASE、EventPhase.AT_TARGET、EventPhase.BUBBLING_PHASE）。
- **bubbles** : 指明事件是不是起泡事件。
- **cancelable** : 指明與事件關聯的行為是否可以避免

有關 ActionScript 3 Event 類的更多資訊,請參閱 Adobe 的 ActionScript 3 參考文檔 :

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

有關每個元件生成的所有事件的清單,請參閱本使用者指南中的“預建元件”一節,其中還包含有相應事件的任何獨特屬性。

在 Scaleform 內,某些本機事件類型將會被 Scaleform 的擴展所取代。例如,當把 Extensions.enabled 屬性設置為 true(真)時,Scaleform 始終生成 MouseEventEx,一個擴展 MouseEvent 事件(而不是標準 MouseEvent)的類。MouseEventEx 添加了支援多控制器和滑鼠右/中按鈕的屬性。使用者可以檢查收到的事件是不是 MouseEventEx 的一個實例,如果是,就把該事件物件歸到擴展類型中。

1.4.2 焦點

在 Flash 內,有一個焦點的概念。預設情況下,stage 焦點設置為通過滑鼠或鍵盤選定的最後一個 InteractiveObject 。

通常,只有一個獲得焦點的元件接收鍵盤事件。設置一個元件的焦點有多種方法。其中一些方法在本文的 [Programming 設計細節](#) 中有所描述。大多數 CLIK 元件當交互時,特別是按下滑鼠左鍵或類似控制器在上面按下(點擊)時也可以接收焦點。(Tab) 和(Shift+Tab)建(或對應導航控制)可以在顯示的焦點元件上移動焦點。這種特性也是大多數桌面應用軟體和網頁上所提供的。注意非 CLIK 元素使用的焦點也可以被 CLIK 元件使用。這意味著一個 Flash 開發者能夠將 CLIK 元素和非 CLIK 元素在場景相互混合和匹配並使焦點行為按意圖進行,特使在使用(Tab) 和(Shift+Tab)鍵時可以在場景中移動焦點。

默認情況下按鈕回應(Enter)鍵和空白鍵。將滑鼠箭頭移動到按鈕上面然後移開也會使元件產生動作,拖動滑鼠游標移入和移出也一樣。

在遊戲控制器或掌上遊戲機,開發者能簡單用對應遊戲杆來控制鍵盤和滑鼠控制事件。例如,(Enter)鍵在 Xbox360 或 PS3 控制器上通常映射為(A)或(X)按鈕。此映射使 UI 介面中使用 CLIK 應用到多種類型的平臺之上。

2 內置元件

初次觀察，Scaleform CLIK 為一個基本的內置 UI 元件集，但是 CLIK 的真實意圖是為創建豐富元件和介面提供一個框架。開發者可以自由地並可以按設想進行擴展—創建自定義元件適應自身需求並在 CLIK 框架上進行構建。

內置 CLIK 元件提供標準的 UI 功能，從基本的按鈕和核取方塊到列表框、下拉功能表和模式對話方塊等複合元件。開發者可以方便得將這些標準元件進行擴展，增加更多特性或在從頭創建自定義元件時作為簡單的參考元件。

以下選項詳細得描述了每個內置元件。他們按照複雜性和功能性進行分組。每個元件使用子章節列表進行描述。

- **User interaction**：用戶如何與元件進行交互。
- **Component setup**：當在 Flash 授權環境中構造元件時需要的元素。
- **States**：元件到函數所需要的的不同可視狀態（關鍵幀）。
- **Inspectable properties**：在 Flash 授權環境中公佈的屬性，便於不使用代碼配置特定元件特性。
- **Events**：在 UI 介面中其他物件可以監聽的元件所觸發的事件列表。
- **Tips and tricks**：完成各種與討論元件相關的任務的實例代碼。

2.1 基本按鈕和文本類型

基本類型包括 Button、CheckBox、Label、TextInput 和 TextArea 元件。按鈕 Button 構成了多數用戶介面的主幹，CheckBox 繼承了 Button 的功能。Label 為一個靜態標簽類型，而 TextInput 和 TextArea 分別可以表示單行文本和多行文本。



圖 1：來自 **Free Realms** 的主功能表按鈕實例

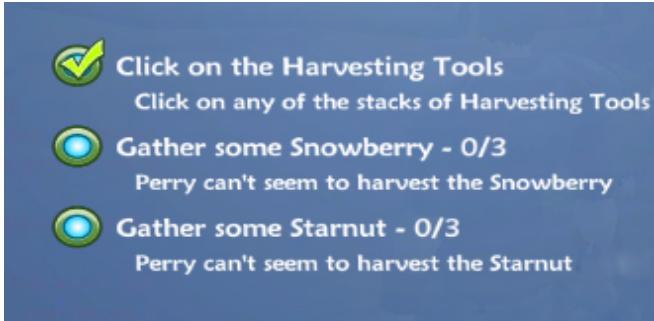


圖 2：來自 *Free Realms* 核取方塊（Check box）實例

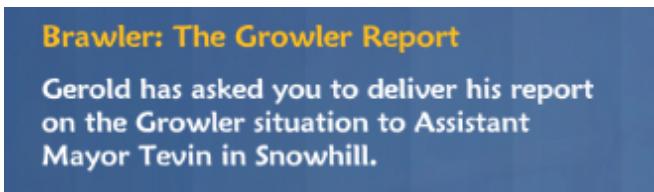


圖 3：來自 *Free Realms* 的標簽（Label）實例

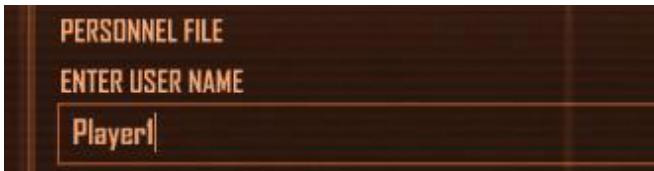


圖 4：來自 *Crysis Warhead* 的文本輸入框（Text input）實例

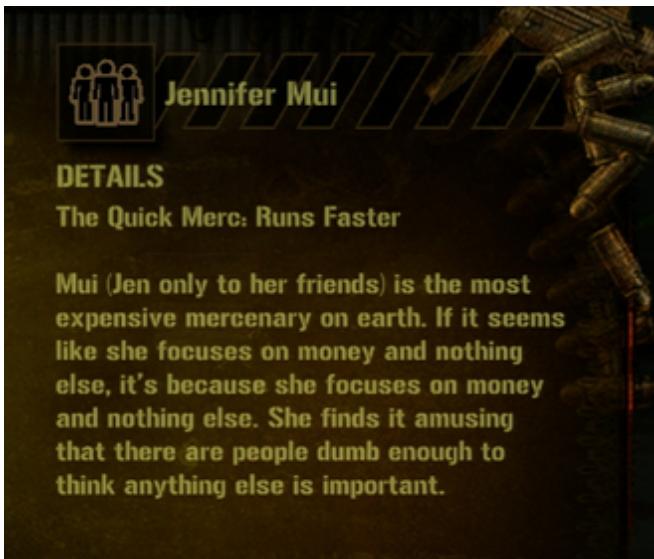


圖 5：來自 *Mercenaries 2* 的文本區域（Text area）實例

2.1.1 Button



圖 6：無皮膚按鈕

按鈕 Button 為 CLIK 框剪的基本元件，可以在任何地方使用需要一個能發出滴答聲的介面控制。默認的 Button 類（scaleform.clik.controls.Button）支援一個 textField 來顯示一個標簽，並制定視覺化用戶互動。按鈕可以單獨使用，也可以作為合成元件的一部分，如作為 ScrollBar 的方向按鈕或者 Slider 翻頁按鈕。大多數交互元件可以回應點擊動作或為擴展按鈕。

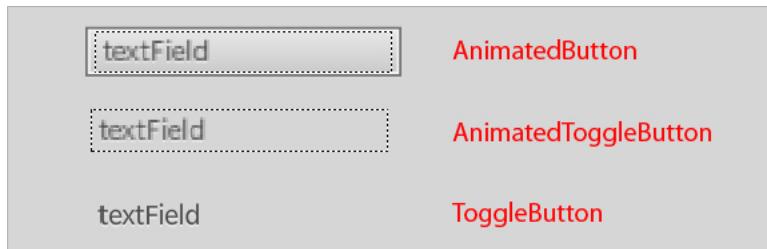


圖 7：AnimatedButton、AnimatedToggleButton 和 ToggleButton

CLIK Button 為一個通用按鈕元件，支援滑鼠交互，鍵盤交互，狀態和其他功能，可以在多種用戶介面中使用。同時也支援選中功能以及動畫狀態。ToggleButton、AnimatedButton 和 AnimatedToggleButton 位於 Button.fla 元件原始檔案中，使用相同的基本元件類。

2.1.1.1 用戶交互

按鈕元件可以用滑鼠或任何類似控制器點擊。當獲得焦點時也可以通過鍵盤按鈕控制。

2.1.1.2 元件設置

一個使用 CLIK Button 類的 MovieClip 必須具備下面所列的子單元。也列出了可選元素。

- **textField:** (可選) TextField 類型。按鈕標簽
- **focusIndicator:** (可選) MovieClip 類型。一個單獨的 MovieClip 用來顯示焦點狀態。如果被使用，必須有兩個命名幀：“show” 和 “hide”。默認情況下，over 狀態用來表示一個獲得焦點的

Button 元件。但是在有些類中，這個行爲限制了使用，設計師可能要將焦點狀態和滑鼠 over 狀態分開使用。

2.1.1.3 狀態

CLIK 按鈕元件支援基於用戶交互的不同的狀態。這些狀態包括：

- **up** 或默認狀態；
- 當滑鼠箭頭在元件上方或者獲得焦點時 **over** 狀態；
- 當按鈕按下時 **down** 狀態；
- **disabled** 狀態；

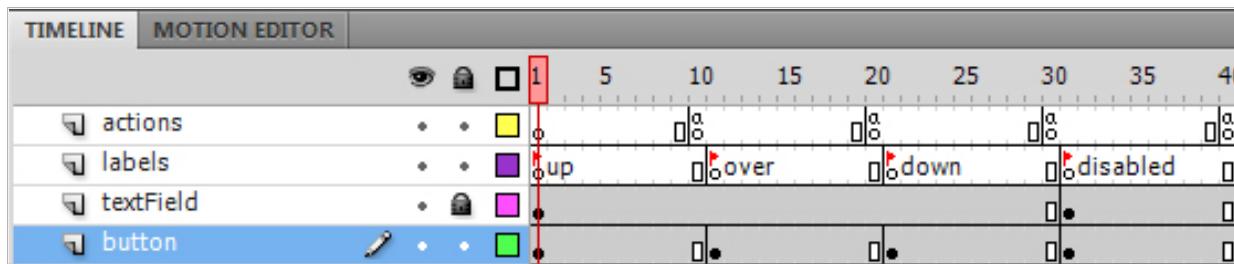


圖 8：Button 時間軸

這些狀態在 Flash 時間軸中用關鍵幀來表示，為按鈕元件所需要的最少關鍵幀設置的正確操作。同時還有其餘狀態擴展元件功能以支援複雜用戶交互和動畫轉換，這些資訊在文檔 [CLIK 按鈕入門](#) 中有所描述。

2.1.1.4 屬性檢查

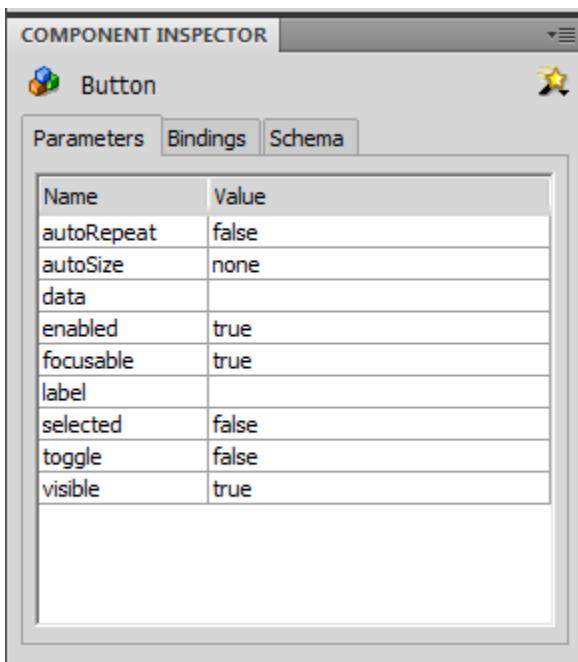


Figure 1: Button component inspectable properties in the CS4 component inspector.

按鈕元件的檢查屬性為：

autoRepeat	確定在按下並保持住按鈕時該按鈕是否發出 "click"(點擊)事件。
autoSize	確定按鈕是否進行縮放以適應其所包含的文本以及已調整大小的按鈕將向哪個方向對齊。將 autoSize 屬性設置為 TextFieldAutoSize.NONE 將會使大小保持不變。
Data	與該按鈕相關的資料。在 ButtonGroup 中使用按鈕時此屬性尤其有用。
enabled	如果設置為 false (假),就禁用該按鈕。已禁用的元件將不再收到使用者輸入。
focusable	啟用/禁用對於元件的焦點管理。將 focusable (可聚焦)屬性設置為 false 將會取消對 tab 鍵、方向鍵以及基於滑鼠的焦點更改的支援。
Label	設置該按鈕的標籤。
selected	設置該按鈕的選定狀態。按鈕可以有兩組滑鼠狀態:已選定狀態和未選定狀態。當一個按鈕的 toggle (切換)屬性為 true 時,點擊該按鈕時就會更改已選定的狀態,不過,即使 toggle 屬性為 false ,也可以使用 ActionScript 設置已選定狀態。
Toggle	設置按鈕套索模式。如果設置為 true ,按鈕將作為一個套索按鈕使用。
Visible	如果設置為 false 則隱藏按鈕。

2.1.1.5 事件

多陣列件產生進行使用者互動、狀態更改和焦點管理的事件。這些事件對於使用 **CLIK** 元件創建功能性使用者介面來說非常重要。

所有事件回檔均會收到單個 **Event**(事件)或 **Event** 子類參數,該參數包含有相應事件的相關資訊。下列屬性是所有事件所共有的。

- **type** : 事件類型。
- **target** : 事件目標。
- **currentTarget** : 通過一個事件偵聽器積極處理 **Event** 物件的物件。
- **eventPhase** : 事件流中的當前階段。 (**EventPhase.CAPTURING_PHASE** 、
EventPhase.AT_TARGET 、**EventPhase.BUBBLING_PHASE**) 。
- **bubbles** : 指明事件是不是起泡事件。
- **cancelable** : 指明與事件關聯的行為是否可以避免

按鈕元件產生事件列表如下。事件旁邊的屬性列表為通用屬性的補充內容。

ComponentEvent.SHOW	運行時可視屬性已設置為 true
ComponentEvent.HIDE	運行時可視屬性已設置為 false
ComponentEvent.STATE_CHANGE	元件的狀態已更改。
FocusHandlerEvent.FOCUS_IN	元件已收到焦點。
FocusHandlerEvent.FOCUS_OUT	元件已失去焦點。
Event.SELECT	選定的屬性已發生變化。
MouseEvent.ROLL_OVER	滑鼠游標滑過該按鈕。 mouseldx :用來生成該事件的滑鼠游標的索引(僅適用於多滑鼠游標環境)。uint 類型。僅限於 Scaleform,需要將該事件歸於 MouseEventEx 。 buttonIdx :指明為哪個按鈕生成該事件(基於零的索引)。僅限於 Scaleform,需要將該事件歸於 MouseEventEx 。
MouseEvent.ROLL_OUT	The mouse cursor has rolled out of the button. mouseldx :用來生成該事件的滑鼠游標的索引(僅適用於多滑鼠游標環境)。uint 類型。僅限於 Scaleform,需要將該事件歸於 MouseEventEx 。 buttonIdx :指明為哪個按鈕生成該事件(基於零的索引)。僅限於 Scaleform,需要將該事件歸於 MouseEventEx 。
ButtonEvent.PRESS	已按 DropdownMenu(下來功能表)。 controllerIdx :用來生成該事件的控制器的索引(僅適用於多滑鼠游標環境)。uint 類型。 isKeyboard :為 true - 假如事件是由電腦鍵盤 (Keyboard)/遊戲鍵盤 (Gamepad) 生成的;為 false - 假如事件是由滑鼠生成的。 isRepeat :為 true - 假如事件是由一個被按住的 autoRepeating 按鈕生成的;為 false - 該按鈕當前不重複。
MouseEvent.DOUBLE_CLICK	已按兩下該元件。僅在 doubleClickEnabled 屬性為 true 時激發。 mouseldx :用來生成該事件的滑鼠游標的索引(僅適用於多滑鼠游標環境)。uint 類型。僅限於 Scaleform,需要將該事件歸於 MouseEventEx 。 buttonIdx :指明為哪個按鈕生成該事件(基於零的索引)。僅限於 Scaleform,需要將該事件歸於 MouseEventEx 。
ButtonEvent.CLICK	按鈕被點擊。 controllerIdx :用來生成該事件的控制器的索引(僅適用於多滑鼠游標環境)。uint 類型。 isKeyboard :為 true - 假如事件是由電腦鍵盤 (Keyboard)/遊戲

	鍵盤 (Gamepad) 生成的;為 <code>false</code> ad 假如事件是由滑鼠生成的。 <code>isRepeat</code> :為 <code>true</code> ea 假如事件是由一個被按住的 <code>autoRepeating</code> 按鈕生成的;為 <code>false</code> pe 該按鈕當前不重複。
ButtonEvent.DRAG_OVER	滑鼠游標拖動到按鈕上方 (滑鼠左鍵被按下) <code>controllerIdx</code> :用來生成該事件的控制器的索引(僅適用於多滑鼠游標環境)。 <code>uint</code> 類型。
ButtonEvent.DRAG_OUT	滑鼠箭頭從按鈕拖開 (滑鼠左鍵被按下)。 <code>controllerIdx</code> :用來生成該事件的控制器的索引(僅適用於多滑鼠游標環境)。 <code>uint</code> 類型。
ButtonEvent.RELEASE_OUTSIDE	滑鼠箭頭從按鈕拖開滑鼠左鍵鬆開。 <code>controllerIdx</code> :用來生成該事件的控制器的索引(僅適用於多滑鼠游標環境)。 <code>uint</code> 類型。

ActionScript 代碼片段用來捕獲或處理這些事件。下例中展示了如何處理按鈕點擊事件。

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

第一行代碼為 “`ButtonEvent.PRESS`” 事件安裝事件監聽器，按鈕名稱為 ‘`myButton`’，當事件觸發時使其調用 `onButtonPress` 函數。向事件處理常式(示例中名為 ‘`e`’)提供的 `ButtonEvent` 參數包含該事件的相關資訊。該參數中的事件的類型必須是同一個類,或者是在添加偵聽器時使用的該類型的一個祖先。

2.1.2 CheckBox



圖 10：無皮膚核取方塊 **CheckBox**

核取方塊 `CheckBox` (`scaleform.clik.controls.CheckBox`) 為一個按鈕元件當被點擊時設置為選中狀態。核取方塊用來顯示 `true/false` (布林值) 的變化。與 `ToggleButton` 功能類似，但是隱藏設置 `Toggle` 屬性。

2.1.2.1 用戶交互

使用滑鼠或者任何相關鍵盤控制器點擊 CheckBox 元件可以使其為選中或未選中。在其他方面，核取方塊行為與按鈕相同。

2.1.2.2 元件設置

使用 CLIK CheckBox 類的動畫剪輯 MovieClip 必須具備下面所列的子單元。也列出了可選元素：

- **textField:** (可選) TextField 類型，按鈕標簽。
- **focusIndicator:** (可選) MovieClip 類型，一個獨立的 MovieClip 用來顯示焦點狀態。如果被使用，該 MovieClip 必須有兩個名字為“show”和“hide”的幀。

2.1.2.3 狀態

根據 Toggle 屬性，核取方塊 CheckBox 需要另外的關鍵幀集來表示選擇狀態。這些狀態包括：

- **up** 或默認狀態；
- 當滑鼠箭頭在元件上方或者獲得焦點時為 **over** 狀態；
- 當按鈕被點擊時候為 **down** 狀態；
- **disabled** 狀態；
- **selected_up** 或者默認狀態；
- 當滑鼠箭頭位於元件上方或獲得焦點時為 **selected_over** 狀態；
- 當按鈕被按下時為 **selected_down** 狀態；
- **selected_disabled** 狀態；

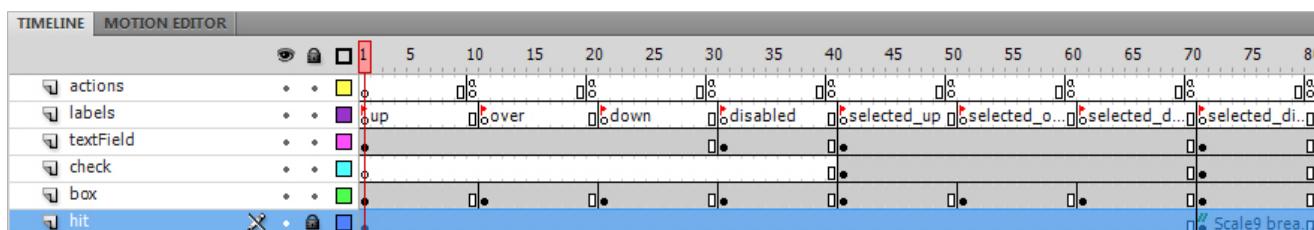
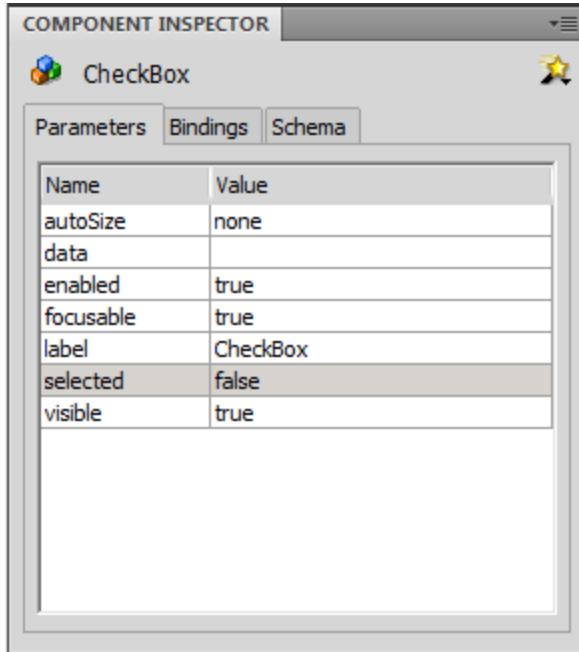


圖 11: CheckBox 時間軸

這裏為核取方塊 CheckBox 所需要的最少關鍵幀設置。按鈕 Button 元件支援狀態和關鍵幀的擴展設置，與核取方塊元件相同，在 [CLIK 按鈕入門](#) 文檔中有詳細描述。

2.1.2.4 屬性檢查



由於從控制按鈕繼承而來，核取方塊 **CheckBox** 包含了與按鈕相同的檢查屬性，具有 **autoRepeat** 屬性和 **Toggle** 屬性。

autoSize	確定按鈕是否進行縮放以適應其所包含的文本以及已調整大小的按鈕 將向哪個方向對齊。將 autoSize 屬性設置為 autoSize=TextFieldAutoSize.NONE 將會使其當前大小保持不變。
Data	與該按鈕相關的資料。在 ButtonGroup 中使用按鈕時此屬性尤其有用。
enabled	如果設置為 false ,就禁用該按鈕。已禁用的元件將不再收到使用者輸入。
focusable	啟用/禁用對於元件的焦點管理。將 focusable (可聚焦)屬性設置為 false 將會取消對 tab 鍵、方向鍵以及基於滑鼠的焦點更改的支援。
Label	設置按鈕標簽
selected	設置該按鈕的選定狀態。按鈕可以有兩組滑鼠狀態:已選定狀態和未選定狀態。當一個按鈕的 toggle (切換)屬性為 true 時,點擊該按鈕時就會更改已選定的狀態,不過,即使 toggle 屬性為 false ,也可以使用 ActionScript 設置已選定狀態。
Visible	如果設置為 false 隱藏按鈕。

2.1.2.5 事件

所有事件回檔均會收到單個 Event(事件)或 Event 子類參數,該參數包含有相應事件的相關資訊。下列屬性是所有事件所共有的。

- **type** : 事件類型。
- **target** : 事件目標。
- **currentTarget** : 通過一個事件偵聽器積極處理 Event 物件的物件。
- **eventPhase** : 事件流中的當前階段。（EventPhase.CAPTURING_PHASE、EventPhase.AT_TARGET、EventPhase.BUBBLING_PHASE）。
- **bubbles** : 指明事件是不是起泡事件。
- **cancelable** : 指明與事件關聯的行為是否可以避免

核取方塊 CheckBox 元件產生的事件列表如下。事件旁邊的屬性列表為通用屬性的補充內容。

ComponentEvent.SHOW	運行時可視屬性已設置為 true
ComponentEvent.HIDE	運行時可視屬性已設置為 false
FocusHandlerEvent.FOCUS_IN	該按鈕收到了焦點。
FocusHandlerEvent.FOCUS_OUT	該按鈕失去了焦點。
Event.SELECT	選定的屬性已發生變化。
ComponentEvent.STATE_CHANGE	按鈕狀態已改變。
MouseEvent.ROLL_OVER	滑鼠箭頭在按鈕上方滾動。 mouseldx :用來生成該事件的滑鼠游標的索引(僅適用於多滑鼠游標環境)。uint 類型。僅限於 Scaleform,需要將該事件歸於 MouseEventEx。 buttonIdx :指明為哪個按鈕生成該事件(基於零的索引)。僅限於 Scaleform,需要將該事件歸於 MouseEventEx。
MouseEvent.ROLL_OUT	滑鼠箭頭從按鈕移開。 mouseldx :用來生成該事件的滑鼠游標的索引(僅適用於多滑鼠游標環境)。uint 類型。僅限於 Scaleform,需要將該事件歸於 MouseEventEx。 buttonIdx :指明為哪個按鈕生成該事件(基於零的索引)。僅限於 Scaleform,需要將該事件歸於 MouseEventEx。
ButtonEvent.PRESS	按鈕被點擊。 controllerIdx :用來生成該事件的控制器的索引(僅適用於多滑鼠游標環境)。uint 類型。 isKeyboard :為 true - 假如事件是由電腦鍵盤 (Keyboard)/遊戲鍵盤 (Gamepad) 生成的;為 false - 假如事件是由滑鼠生成的。 isRepeat :為 true - 假如事件是由一個被按住的 autoRepeating 按鈕生成的;為 false - 該按鈕當前不重複。

MouseEvent.DOUBLE_CLICK	按鈕被雙擊。只在 <i>doubleClickEnabled</i> 屬性為 <i>true</i> 時被觸發。 <i>mouseldx</i> :用來生成該事件的滑鼠游標的索引(僅適用于多滑鼠游標環境)。 <i>uint</i> 類型。僅限於 Scaleform ,需要將該事件歸於 MouseEventEx 。 <i>buttonIdx</i> :指明為哪個按鈕生成該事件(基於零的索引)。僅限於 Scaleform ,需要將該事件歸於 MouseEventEx 。
ButtonEvent.CLICK	按鈕被點擊。 <i>controllerIdx</i> :用來生成該事件的控制器的索引(僅適用于多滑鼠游標環境)。 <i>uint</i> 類型。 <i>isKeyboard</i> :為 <i>true</i> - 假如事件是由電腦鍵盤 (Keyboard)/遊戲鍵盤 (Gamepad) 生成的;為 <i>false</i> - 假如事件是由滑鼠生成的。 <i>isRepeat</i> :為 <i>true</i> - 假如事件是由一個被按住的 <i>autoRepeating</i> 按鈕生成的;為 <i>false</i> - 該按鈕當前不重複。
ButtonEvent.DRAG_OVER	滑鼠箭頭拖動到按鈕上方 (滑鼠左鍵被按下) <i>controllerIdx</i> :用來生成該事件的控制器的索引(僅適用于多滑鼠游標環境)。 <i>uint</i> 類型。
ButtonEvent.DRAG_OUT	滑鼠箭頭從按鈕拖開 (滑鼠左鍵被按下)。 <i>controllerIdx</i> :用來生成該事件的控制器的索引(僅適用于多滑鼠游標環境)。 <i>uint</i> 類型。
ButtonEvent.RELEASE_OUTSIDE	滑鼠箭頭從按鈕拖開滑鼠左鍵鬆開。 <i>controllerIdx</i> :用來生成該事件的控制器的索引(僅適用于多滑鼠游標環境)。 <i>uint</i> 類型。

下例中代碼展示了如何處理核取方塊 CheckBox 的 Toggle 動作：

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

2.1.3 Label



圖 12: 未繪製皮膚的標簽

CLIK 標簽 Label 元件(`scaleform.clik.controls.Label`)為一個不可編輯的標準 `textField` 元件，由 MovieClip 符號進行圍繞，具有一些附加的便利特性。在本質上，標簽 Label 支援與標準 `textField` 相同的屬性和行為，但是，有一些常用的特性為該元件本身所特有。如果用戶需要直接改變其屬性，支援訪問標簽實際上的 `textField` 區域。在某些情況下，如一些下面內容將會描述的內容，開發者可能會使用 `textField` 替代標簽元件。

由於標簽 Label 為一個 MovieClip 符號，可以用圖形元素進行修飾，而這在標準的 `textField` 無法做到。作為一個符號，不需要如 `textField` 實例一樣對每一個進行配置。標簽 Label 還提供了 `disabled` 狀態在時間軸可以被定義。然而，用標準的 `textField` 來類比這些功能需要很多複雜的 AS2 代碼。

標簽 Label 元件默認強制使用，這意味著在運行時在場景中改變一個標簽 Label 實例大小不會顯示出效果，開發者應該在大多數情況下使用 `textField` 來提到 Label 標簽。通常，文本元素不需要經常被重用，則 `textField` 比起標簽 Label 使用更加簡單。

2.1.3.1 用戶交互

在標簽 Label 內無用戶交互。

2.1.3.2 元件設置

使用 CLIK Label 類的 MovieClip 必須擁有下列命名的子元素。標注了對應的可選元素：

- **`textField`**: TextField 類型，Label 標簽文本

2.1.3.3 狀態

CLIK Label 元件支援基於標準屬性的兩種狀態：

- **default** 或者 enabled 狀態；
- **disabled** 狀態

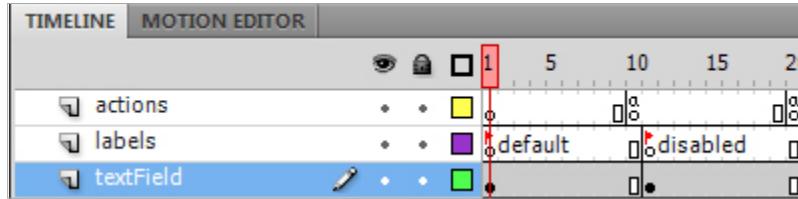
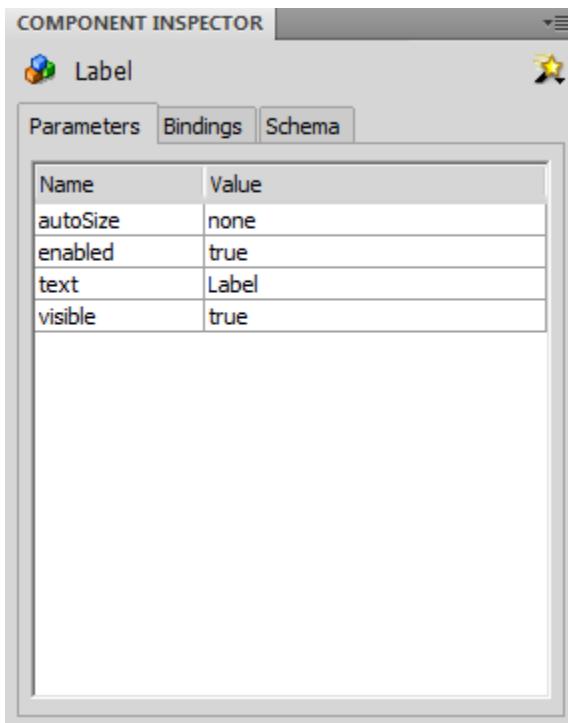


圖 13: Label 時間軸

2.1.3.4 檢查屬性



標簽 Label 的檢查屬性如下：

Text	設置標簽文本
Visible	如果設置為 false,就隱藏元件。
Enabled	如果設置為 false,就禁用元件。
autoSize	確定標籤 (Label) 是否會縮放以適應其所包含的文本以及已調整大小的按鈕將向哪個方向對齊。將 autoSize 屬性設置為 autoSize=TextFieldAutoSize.NONE 將會使其當前大小保持不變。

2.1.3.5 事件

所有事件回檔均會收到單個 Event(事件)或 Event 子類參數,該參數包含有相應事件的相關資訊。下列屬性是所有事件所共有的。

- **type** : 事件類型。
- **target** : 事件目標。
- **currentTarget** : 通過一個事件偵聽器積極處理 Event 物件的物件。
- **eventPhase** : 事件流中的當前階段。（EventPhase.CAPTURING_PHASE、EventPhase.AT_TARGET、EventPhase.BUBBLING_PHASE）。
- **bubbles** : 指明事件是不是起泡事件。
- **cancelable** : 指明與事件關聯的行為是否可以避免

由標簽 Label 元件產生的事件列表如下。事件旁邊的屬性列表為通用屬性的補充內容。

ComponentEvent.SHOW	visible(可見)屬性已在運行時設置為 true。
ComponentEvent.HIDE	visible 屬性已在運行時設置為 false。
ComponentEvent.STATE_CHANGE	元件的狀態已更改。

2.1.4 TextInput

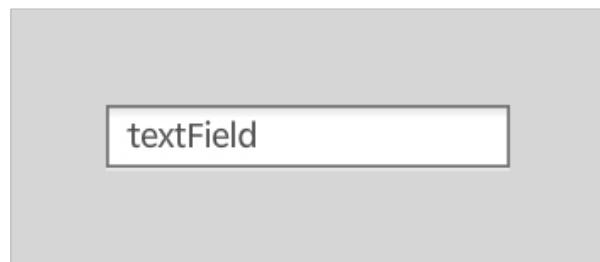


圖 14: 無皮膚 TextInput.

文本輸入 TextInput (scaleform.clik.controls.TextInput)為一個可編輯的 textField 元件，用來捕獲用戶文本輸入。與標簽 Label 類似，該元件僅僅為一個標準 textField 的外框，因此支援 textField 的功能，如密碼模式、最大字元數和 HTML 文本。只有一部分屬性為該元件本身所有，其他的屬性可以直接進入 TextInput 的 textField 實例進行更改。

textField 元件應該用於輸入，因為無需編輯文本可以使用 Label 標簽來顯示。與 Label 標簽類似，開發者可能會根據自身需求用標準的 textFields 代替 TextInput 元件使用。但是，當開發複雜 UI 介面時，特別是在 PC 應用中，TextInput 元件提供了基於標準 textField 的有價值的功能擴展。

作為特殊屬性，**TextInput** 支援焦點和 **disabled** 狀態，這在標準 **textField** 中很難實現。根據獨立的焦點狀態，**TextInput** 支援自定義焦點指示器，這在標準 **TextInput** 也不包含。複雜的 AS2 代碼需要改變標準 **TextInput** 的外觀類型，而 **TextInput** 的外觀類型可以在時間軸上簡單得進行配置。**TextInput** 檢查屬性為不熟悉 Flash Studio 的設計師和編程人員提供了一種簡單的工作流程。開發者可以簡便得監聽 **TextInput** 觸發的事件以創建自定義行為。

TextInput 同時也支援 **textField** 提供的標準選擇和剪切、拷貝和粘貼功能，包括了多行 HTML 格式文本。默認情況下，快捷鍵命令為選擇 (Shift+Arrows)、剪切 (Shift+Delete)、拷貝 (Ctrl+Insert)、和粘貼 (Shift+Insert)。

“文本輸入”可支援滑鼠滾動和滑出事件。可通過對這一特殊的 **actAsButton** 屬性進行設置，來提供能夠執行兩種滑鼠事件的兩個額外關鍵幀。這些幀被命名為“over”和“out”，分別代表滾動和滑出狀態。如果設置了 **actAsButton** 模式，並且“over”/“out”幀不存在，那麼“文本輸入”將會按照“預設值”關鍵幀執行這兩種事件。請注意，這些幀不會通過預設的“文本輸入”元件而出現。開發商會根據具體要求對他們進行添加。

當用戶未設定或輸入值時，該元件還支援默認顯示的文本。可將默認文本屬性設置為任何字串。默認文本的主題（顏色和樣式）為淺灰(0xAAAAAA)，斜體。可通過向“默認文本格式”屬性分配一個新的“文本格式”物件的方法對樣式進行自定義。

2.1.4.1 用戶交互

點擊 **TextInput** 使其獲得焦點，則在 **textField** 中出現一個箭頭。當箭頭顯現時，用戶能夠通過鍵盤或類似控制設備輸入字元。按下坐右方向鍵可以移動箭頭。當箭頭已經位於 **textField** 的左邊緣，使用左方向鍵則焦點將轉移到左邊的控制部件。使用右方向鍵也如此。

2.1.4.2 元件設置

使用 CLIK **TextInput** 類的 **MovieClip** 必須具備下面所列的子單元。也列出了可選元素：

- **textField**: **textField** 類型

2.1.4.3 States 狀態

CLIK **TextInput** 元件支援三種狀態，均基於焦點和 **disabled** 屬性：

- **default** 或 **enabled** 狀態；

- **focused** 狀態，通常為 **textField** 周圍突出顯示的邊框；
- **disabled** 狀態。

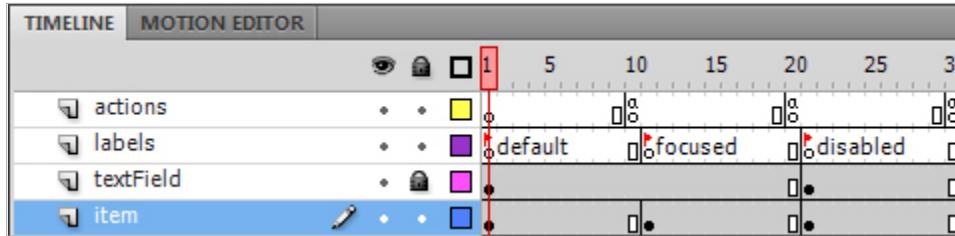
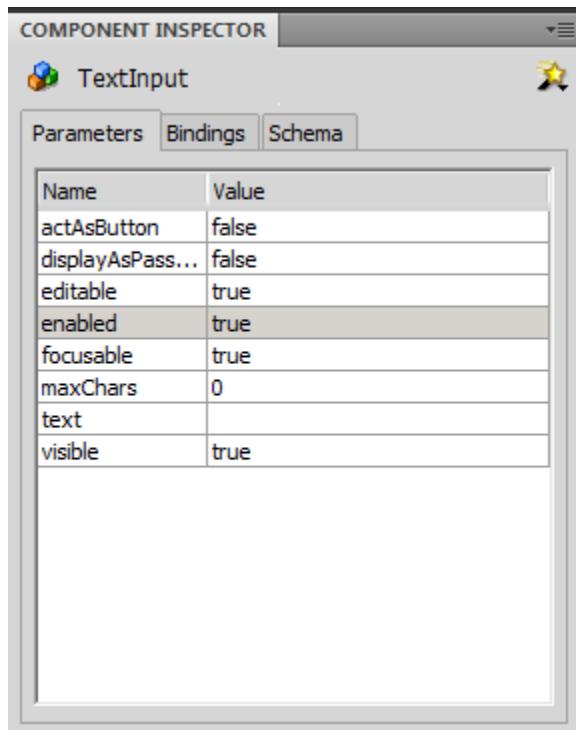


圖 15: TextInput 時間軸

2.1.4.4 檢查屬性



文本輸入 TextInput 元件的檢查屬性如下所示:

actAsButton	如果為“真”，則“文本輸入”的行為在未選中狀態下類似一個按鈕，並支援滾動和滑出狀態。一旦按下滑鼠或 tab 鍵，“文本輸入”將會轉為正常模式，直至退出選中狀態。
displayAsPassword	如果是 true,就把 textField 設置為顯示 '*' 字元,而不是真正字元。textField 的值將會是使用者輸入的真正的字元,由文字屬性返回。
Editable	如果設置為 false,就使 TextInput 變為不可編輯。
Enabled	Disables the component if set to false.
Focusable	啟用/禁用對於元件的焦點管理。將 focusable(可聚焦)屬性

maxChars	設置為 <code>false</code> 將會取消對 tab 鍵、方向鍵以及基於滑鼠的焦點更改的支援。
Text	一個大於零的數位限制可以輸入 <code>textField</code> 的字元的數量。
Visible	設置 <code>textField</code> 的初始文本。

2.1.4.5 事件

所有事件回檔均會收到單個 `Event`(事件)或 `Event` 子類參數,該參數包含有相應事件的相關資訊。下列屬性是所有事件所共有的。

- **`type`** : 事件類型。
- **`target`** : 事件目標。
- **`currentTarget`** : 通過一個事件偵聽器積極處理 `Event` 物件的物件。
- **`eventPhase`** : 事件流中的當前階段。（`EventPhase.CAPTURING_PHASE`、`EventPhase.AT_TARGET`、`EventPhase.BUBBLING_PHASE`）。
- **`bubbles`** : 指明事件是不是起泡事件。
- **`cancelable`** : 指明與事件關聯的行為是否可以避免

文本輸入 `TextInput` 元件產生的事件列表如下。事件旁邊的屬性列表為通用屬性的補充內容。

ComponentEvent.SHOW	<code>visible(可見)</code> 屬性已在運行時設置為 <code>true</code> 。
ComponentEvent.HIDE	<code>visible</code> 屬性已在運行時設置為 <code>false</code> 。
FocusHandlerEvent.FOCUS_IN	元件已收到焦點。
FocusHandlerEvent.FOCUS_OUT	元件已失去焦點。
Event.CHANGE	<code>textField</code> 內容發生改變
ComponentEvent.STATE_CHANGE	元件的狀態已更改。
MouseEvent.ROLL_OVER	滑鼠箭頭在按鈕上方滾動。 <code>mouseldx</code> :用來生成該事件的滑鼠游標的索引(僅適用於多滑鼠游標環境)。uint 類型。僅限於 <code>Scaleform</code> ,需要將該事件歸於 <code>MouseEventEx</code> 。 <code>buttonIdx</code> :指明為哪個按鈕生成該事件(基於零的索引)。僅限於 <code>Scaleform</code> ,需要將該事件歸於 <code>MouseEventEx</code> 。
MouseEvent.ROLL_OUT	滑鼠箭頭從按鈕移開。 <code>mouseldx</code> :用來生成該事件的滑鼠游標的索引(僅適用於多滑鼠游標環境)。uint 類型。僅限於 <code>Scaleform</code> ,需要

將該事件歸於 `MouseEventEx`。

`buttonIdx`:指明為哪個按鈕生成該事件(基於零的索引)。僅限於 `Scaleform`,需要將該事件歸於 `MouseEventEx`。

以下代碼為揭示如何監聽 `textField` 內容變化的實例：

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

2.1.5 TextArea

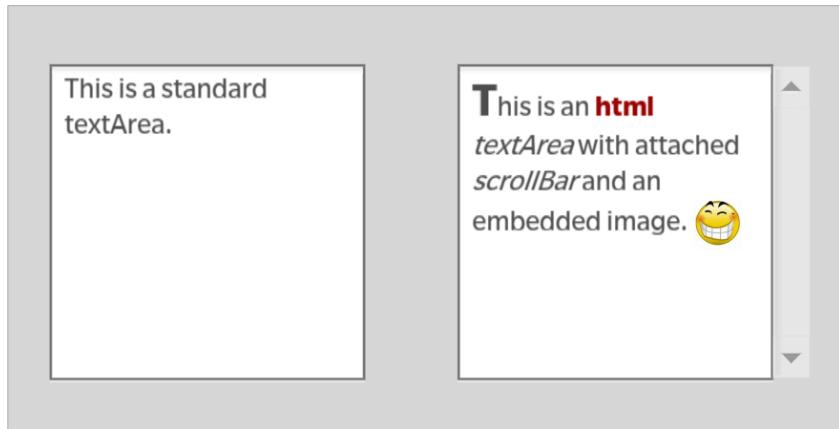


圖 16: 無皮膚 TextArea。

文本區域 `TextArea` (`scaleform.clik.controls.TextArea`) 從 `CLIK TextInput` 繼承而來，共用相同功能、屬性和狀態，但是具有一個可選捲軸 `ScrollBar` 用於多行可編輯滾動文本輸入。請參考“文本輸入”描述，學習更多的有關“文本輸入”和“文本區域”共用的特殊功能。

類似於 `Label` 和 `TextInput`，`TextArea` 也為一個標準的多行 `textField` 的外框，因此支援 `textField` 的屬性和行為，如 `HTML` 文本、文字邊框、選擇、剪切、拷貝、粘貼。開發者可以簡單得用一個標準的 `textField` 替代 `TextArea`，但是，強烈建議使用該元件，因為其具有擴展功能、狀態、檢查屬性和事件。

儘管標準 `textField` 能夠用於 `ScrollIndicator` 或者 `ScrollBar`，`TextArea` 提供了與這些元件緊湊的組合功能。與標準的 `textField` 不同，`TextArea` 能夠在使用鍵盤或類似控制器使其獲得焦點時進行滾動，甚至在不可編輯時也如此。由於滾動元件不能獲得焦點，`TextArea` 能夠展現更多有沒的焦點圖型狀態，能夠在獲得焦點的時候裝飾自身和滾動元件。

2.1.5.1 用戶交互

點擊 **TextArea** 使其獲得焦點，則在 **textField** 中出現一個箭頭。當箭頭顯現時，用戶能夠通過鍵盤或類似控制設備輸入字元。按下左右方向鍵可以移動箭頭。當箭頭已經位於 **textField** 的邊緣，使用方向鍵則焦點將轉移到相鄰的控制部件。

2.1.5.2 元件設置

一個使用 **CLIK TextArea** 類的 **MovieClip** 必須具備下面所列的子單元。也列出了可選元素：

- **textField**: **TextField** 類型

2.1.5.3 狀態

與上級元件 **TextInput** 類似，**TextArea** 元件支援三種狀態，以焦點和 **disabled** 屬性為基礎。

- **default** 或 **enabled** 狀態；
- **focused** 狀態，通常為 **textField** 周圍突出顯示的邊框；
- **disabled** 狀態；

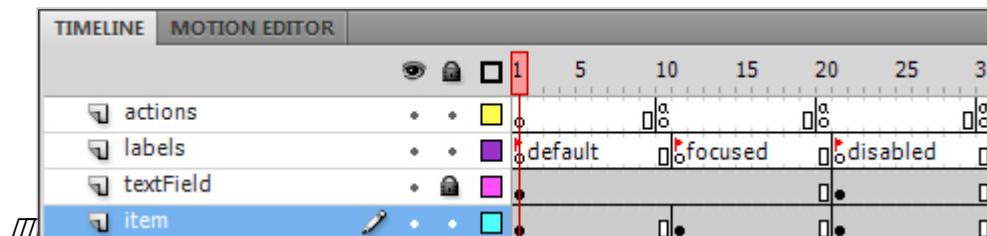
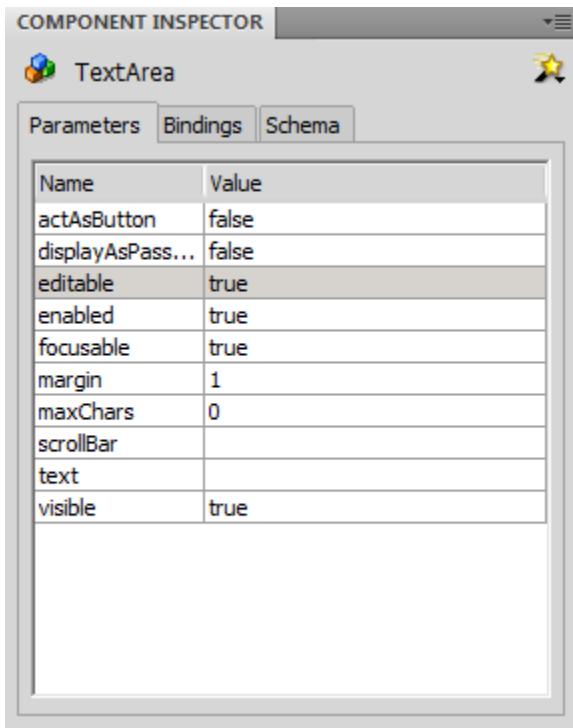


圖 17: **TextArea** 時間軸

2.1.5.4 檢查屬性



TextArea 元件檢查屬性與 TextInput 類似，具有一對輔助和冗長的密碼特性。輔助特性與 CLIK ScrollBar 元件有關，將在 2.4 節描述：

actAsButton	如果為 true, 則 TextInput 的行為將會與一個未聚焦的按鈕相似, 並支援 rollOver 和 rollOut 狀態。一旦通過按滑鼠或 tab 鍵進行了聚焦, TextInput 就恢復到其正常模式, 直到失去焦點。
displayAsPassword	如果是 true, 就把 textField 設置為顯示 '*' 字元, 而不是真正字元。textField 的值將會是使用者輸入的真正的字元, 由文字屬性返回。
Editable	如果設置為 false 則使 TextInput 不可編輯
Enabled	如果設置為 false, 就禁用元件。
focusable	啟用/禁用對於元件的焦點管理。將 focusable(可聚焦)屬性設置為 false 將會取消對 tab 鍵、方向鍵以及基於滑鼠的焦點更改的支援。
maxChars	一個大於零的數位, 作為 textField 中可以輸入的最多字元數。
scrollBar	CLIK ScrollBar 元件使用的實例名, 或者一個到 ScrollBar 符號的鏈結 ID (本例中由 TextArea 創建一個實例)。
scrollPolicy	當設置為 "auto" 時, 滾軸將只顯示是否有足夠的文本可以需要滾動。如果設置為 "on" 則 ScrollBar 將長期顯示, 如果設置為 "off" 則不顯示, 該屬性只影響分配一個 ScrollBar 的元件 (參考 ScrollBar 特性)
Text	設置 textField 的初始文本。

Visible

如果設置為 `false` 則隱藏元件

2.1.5.5 事件

所有事件回檔均會收到單個 `Event`(事件)或 `Event` 子類參數,該參數包含有相應事件的相關資訊。下列屬性是所有事件所共有的。

- **`type`** : 事件類型。
- **`target`** : 事件目標。
- **`currentTarget`** : 通過一個事件偵聽器積極處理 `Event` 物件的物件。
- **`eventPhase`** : 事件流中的當前階段。 (`EventPhase.CAPTURING_PHASE`、
`EventPhase.AT_TARGET`、`EventPhase.BUBBLING_PHASE`) 。
- **`bubbles`** : 指明事件是不是起泡事件。
- **`cancelable`** : 指明與事件關聯的行為是否可以避免

文本區域 `TextArea` 元件產生的事件列表如下所示。事件旁列出的屬性為通用屬性的補充。

ComponentEvent.SHOW	<code>visible</code> (可見)屬性已在運行時設置為 <code>true</code> 。
ComponentEvent.HIDE	<code>visible</code> 屬性已在運行時設置為 <code>false</code> 。
FocusHandlerEvent.FOCUS_IN	元件已收到焦點。
FocusHandlerEvent.FOCUS_OUT	元件已失去焦點。
Event.CHANGE	<code>textField</code> 內容發生改變
Event.SCROLL	在文本區域滾動
ComponentEvent.STATE_CHANGE	元件的狀態已更改。
MouseEvent.ROLL_OVER	<code>mouseldx</code> :用來生成該事件的滑鼠游標的索引(僅適用於多滑鼠游標環境)。 <code>uint</code> 類型。僅限於 <code>Scaleform</code> ,需要將該事件歸於 <code>MouseEventEx</code> 。 <code>buttonIdx</code> :指明為哪個按鈕生成該事件(基於零的索引)。僅限於 <code>Scaleform</code> ,需要將該事件歸於 <code>MouseEventEx</code> 。
MouseEvent.ROLL_OUT	<code>mouseldx</code> :用來生成該事件的滑鼠游標的索引(僅適用於多滑鼠游標環境)。 <code>uint</code> 類型。僅限於 <code>Scaleform</code> ,需要將該事件歸於 <code>MouseEventEx</code> 。 <code>buttonIdx</code> :指明為哪個按鈕生成該事件(基於零的索引)。僅限於 <code>Scaleform</code> ,需要將該事件歸於 <code>MouseEventEx</code> 。

以下例子展示了如何監聽 TextArea 滾動事件：

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

2.2 分組類型

分組類型包括 RadioButton、ButtonGroup 和 ButtonBar 元件。ButtonGroup 為一個管理類型具有特別的邏輯來維護按鈕 Button 的分組。不具有可視外觀不存在於場景中。但是，ButtonBar 存在於場景中也用來維護按鈕分組。RadioButton 為一個特殊的按鈕可以自動與同類元件分組到 ButtonGroup 中。

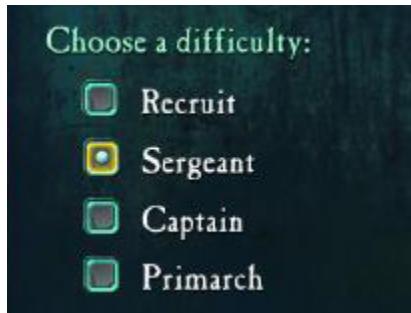


圖 18:來自 *Dawn of War II* 的選擇按鈕組

2.2.1 RadioButton



圖 19:無皮膚的選擇按鈕 RadioButton

選擇按鈕 RadioButton (scaleform.clik.controls.RadioButton)為一個按鈕元件，通常屬於一個集用來顯示和改變一個值。在這個集中只能選擇一個選擇按鈕，點擊集中另外一個選擇按鈕將選中一個新的元件，之前被選中的元件將失去被選擇狀態。

CLIK 選擇按鈕與核取方塊 CheckBox 元件非常類似，共用功能、狀態和行為。主要的區別為選擇按鈕支援分組屬性，可以指派一個自定義按鈕組 ButtonGroup（見下節）。選擇按鈕不需要固定設置為選中屬性，因為選中屬性由按鈕組實例進行管理。

2.2.1.1 用戶交互

使用滑鼠或類似控制器點擊未選中的選擇按鈕元件將其選中。如果選擇按鈕為被選中狀態，另外一個同屬一個按鈕組的選擇按鈕被點擊，則先前選中的選擇按鈕將失去選中狀態。其他方面，選擇按鈕行為與按鈕相同。

2.2.1.2 元件設置

使用 Clik RadioButton 類的 MovieClip 必須具備下面所列的子單元。也列出了可選元素：

- **textField**：(可選) TextField 類型，按鈕標簽。
- **focusIndicator**：(可選) MovieClip 類型，一個獨立的 MovieClip 用來顯示焦點狀態。如果被使用，該 MovieClip 必須具有兩個幀名為：“show” 和 “hide”。

2.2.1.3 狀態

由於選擇按鈕可以在選中和未選中狀態間轉換，與核取方塊 CheckBox 類似，需要至少以下幾種狀態：

- **up** 或默認狀態；
- 當滑鼠箭頭在元件上方或者獲得焦點時為 **over** 狀態；
- 當按鈕被點擊時候為 **down** 狀態；
- **disabled** 狀態；
- **selected_up** 或者默認狀態；
- 當滑鼠箭頭位於元件上方或獲得焦點時為 **selected_over** 狀態；
- 當按鈕被按下時為 **selected_down** 狀態；
- **selected_disabled** 狀態；

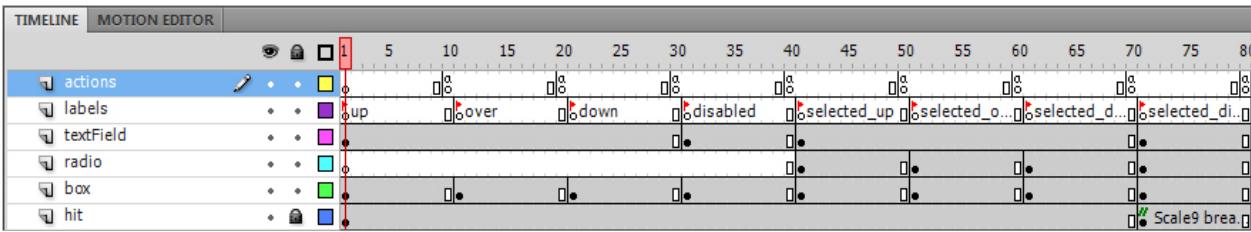
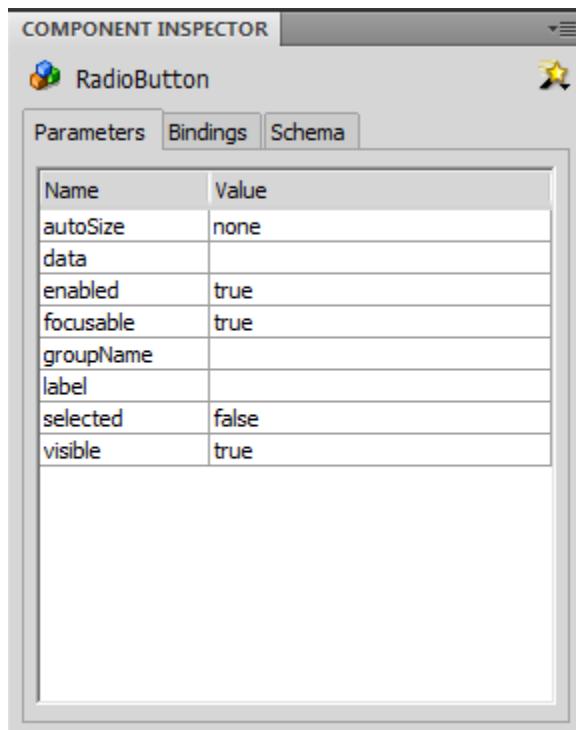


圖 20:選擇按鈕 RadioButton 時間軸

這裏為選擇按鈕 RadioButton 所需要的最少關鍵幀設置。按鈕 Button 元件支援狀態和關鍵幀的擴展設置，與選擇按鈕 RadioButton 元件相同，在 [CLIK 按鈕入門](#) 文檔中有詳細描述。

2.2.1.4 檢查屬性



由於從按鈕 Button 控制按鈕繼承而來，選擇按鈕 RadioButton 包含了與按鈕相同的檢查屬相，具有 disableFocus 屬性和 Toggle 屬性。

autoSize	確定按鈕是否進行縮放以適應其所包含的文本以及已調整大小的按鈕將向哪個方向對齊。將 autoSize 屬性設置為 autoSize=TextFieldAutoSize.NONE 將會使其當前大小保持不變。
Data	與該按鈕相關的資料。在 ButtonGroup 中使用按鈕時此屬性尤其有用。

enabled	如果設置為 <code>false</code> ,就禁用該按鈕。禁用的元件將不再收到使用者輸入。
focusable	啟用/禁用對於元件的焦點管理。將 <code>focusable</code> (可聚焦)屬性設置為 <code>false</code> 將會取消對 <code>tab</code> 鍵、方向鍵以及基於滑鼠的焦點更改的支援。
groupName	<code>ButtonGroup</code> 實例的一個名稱,該實例本身存在,或者應由 <code>RadioButton</code> 自動創建。如果是由 <code>RadioButton</code> 創建,新的 <code>ButtonGroup</code> 將會存在於 <code>RadioButton</code> 的容器內部。例如,如果 <code>RadioButton</code> 存在於 <code>_root</code> 之內,則會在 <code>_root</code> 中創建其 <code>ButtonGroup</code> 物件。使用同一個組的所有 <code>RadioButton</code> 都將屬於一個集。
Label	設置該按鈕的標籤。
selected	設置該按鈕的選定狀態。按鈕可以有兩組滑鼠狀態:已選定狀態和未選定狀態。當一個按鈕的 <code>toggle</code> (切換)屬性為 <code>true</code> 時,點擊該按鈕時就會更改已選定的狀態,不過,即使 <code>toggle</code> 屬性為 <code>false</code> ,也可以使用 ActionScript 設置已選定狀態。

2.2.1.5 事件

所有事件回檔均會收到單個 `Event`(事件)或 `Event` 子類參數,該參數包含有相應事件的相關資訊。下列屬性是所有事件所共有的。

- **`type`** : 事件類型。
- **`target`** : 事件目標。
- **`currentTarget`** : 通過一個事件偵聽器積極處理 `Event` 物件的物件。
- **`eventPhase`** : 事件流中的當前階段。 (`EventPhase.CAPTURING_PHASE`、
`EventPhase.AT_TARGET`、`EventPhase.BUBBLING_PHASE`) 。
- **`bubbles`** : 指明事件是不是起泡事件。
- **`cancelable`** : 指明與事件關聯的行為是否可以避免

選擇按鈕 `RadioButton` 元件產生的事件列表如下所示。事件旁列出的屬性為通用屬性的補充。

ComponentEvent.SHOW	<code>visible</code> (可見)屬性已在運行時設置為 <code>true</code> 。
ComponentEvent.HIDE	<code>visible</code> 屬性已在運行時設置為 <code>false</code> 。
FocusHandlerEvent.FOCUS_IN	元件已收到焦點。
FocusHandlerEvent.FOCUS_OUT	元件已失去焦點。
Event.SELECT	選定的屬性已發生變化。
ComponentEvent.STATE_CHANGE	元件的狀態已更改。
MouseEvent.ROLL_OVER	滑鼠箭頭在按鈕上方滾動。

	mouseldx: 用來生成該事件的滑鼠游標的索引(僅適用于多滑鼠游標環境)。uint 類型。僅限於 Scaleform,需要將該事件歸於 MouseEventEx。
	buttonIdx: 指明為哪個按鈕生成該事件(基於零的索引)。僅限於 Scaleform,需要將該事件歸於 MouseEventEx。
MouseEvent.ROLL_OUT	滑鼠箭頭從按鈕移開。
	mouseldx: 用來生成該事件的滑鼠游標的索引(僅適用于多滑鼠游標環境)。uint 類型。僅限於 Scaleform,需要將該事件歸於 MouseEventEx。
	buttonIdx: 指明為哪個按鈕生成該事件(基於零的索引)。僅限於 Scaleform,需要將該事件歸於 MouseEventEx。
ButtonEvent.PRESS	按鈕被點擊。
	controllerIdx: 用來生成該事件的控制器的索引(僅適用于多滑鼠游標環境)。uint 類型。
	isKeyboard: 為 true - 假如事件是由電腦鍵盤 (Keyboard)/遊戲鍵盤 (Gamepad) 生成的;為 false – 假如事件是由滑鼠生成的。
	isRepeat: 為 true – 假如事件是由一個被按住的 autoRepeating 按鈕生成的;為 false – 該按鈕當前不重複。
MouseEvent.DOUBLE_CLICK	按鈕被雙擊。只在 <i>doubleClickEnabled</i> 屬性為 true 時被觸發。
	mouseldx: 用來生成該事件的滑鼠游標的索引(僅適用于多滑鼠游標環境)。uint 類型。僅限於 Scaleform,需要將該事件歸於 MouseEventEx。
	buttonIdx: 指明為哪個按鈕生成該事件(基於零的索引)。僅限於 Scaleform,需要將該事件歸於 MouseEventEx。
ButtonEvent.CLICK	按鈕被點擊。
	controllerIdx: 用來生成該事件的控制器的索引(僅適用于多滑鼠游標環境)。uint 類型。
	isKeyboard: 為 true - 假如事件是由電腦鍵盤 (Keyboard)/遊戲鍵盤 (Gamepad) 生成的;為 false – 假如事件是由滑鼠生成的。
	isRepeat: 為 true – 假如事件是由一個被按住的 autoRepeating 按鈕生成的;為 false – 該按鈕當前不重複。
ButtonEvent.DRAG_OVER	滑鼠箭頭拖動到按鈕上方 (滑鼠左鍵被按下)

	controllerIdx :用來生成該事件的控制器的索引(僅適用于多滑鼠游標環境)。uint 類型。
ButtonEvent.DRAG_OUT	滑鼠箭頭從按鈕拖開 (滑鼠左鍵被按下) 。
	controllerIdx :用來生成該事件的控制器的索引(僅適用于多滑鼠游標環境)。uint 類型。
ButtonEvent.RELEASE_OUTSIDE	滑鼠箭頭從按鈕拖開滑鼠左鍵鬆開。
	controllerIdx :用來生成該事件的控制器的索引(僅適用于多滑鼠游標環境)。uint 類型。

以下例子顯示如何處理選擇按鈕 RadioButton 的選中狀態：

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

2.2.2 ButtonGroup

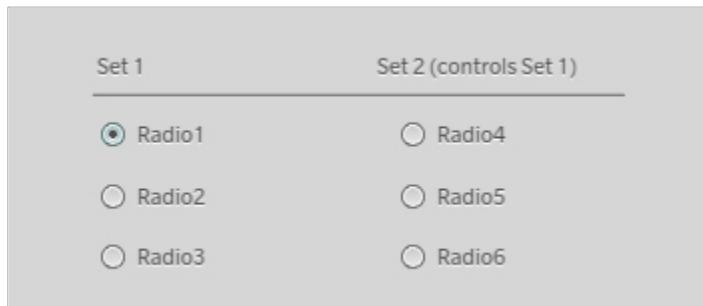


圖 21: 無皮膚按鈕分組 **ButtonGroup**.

CLIK ButtonGroup (scaleform.clik.controls.ButtonGroup)本身不是一個元件，但是有重要作用用於管理按鈕集。可以使在集中的一個按鈕被選中，確保其餘的未選中。如果用戶選擇集中的另一個按鈕，則當前選中按鈕將變為未選中。任何從 CLIK 按鈕元件繼承的元件（如核取方塊 CheckBox 和選擇按鈕 RadioButton）能夠使用按鈕分組 ButtonGroup 實例。

2.2.2.1 用戶交互

按鈕組不具有用戶交互功能，因為不是可視元件。但是，在下屬的選擇按鈕 RadioButton 被點擊時起到間接的作用。

2.2.2.2 元件設置

使用 CLIK 按鈕組 ButtonGroup 類的 MovieClip 不需要任何子單元，因為不具有視覺化外觀。

2.2.2.3 狀態

按鈕組 ButtonGroup 在場景中不具有可視外觀。因此無關聯狀態。

2.2.2.4 檢查屬性

按鈕組 ButtonGroup 在場景中不具有可視外觀。因此無需檢查屬性。

2.2.2.5 事件

所有事件回檔均會收到單個 Event(事件)或 Event 子類參數，該參數包含有相應事件的相關資訊。下列屬性是所有事件所共有的。

- **type**：事件類型。
- **target**：事件目標。
- **currentTarget**：通過一個事件偵聽器積極處理 Event 物件的物件。
- **eventPhase**：事件流中的當前階段。（EventPhase.CAPTURING_PHASE、EventPhase.AT_TARGET、EventPhase.BUBBLING_PHASE）。
- **bubbles**：指明事件是不是起泡事件。
- **cancelable**：指明與事件關聯的行為是否可以避免

ButtonGroup 元件產生的事件列表如下所示。事件旁列出的屬性為通用屬性的補充。

Event.CHANGE 在組中選擇一個新的按鈕

ButtonEvent.CLICK 組中的按鈕被點擊

target: 選擇按鈕，CLIK 按鈕 Button 類型

下例展示了如何判斷按鈕組 ButtonGroup 中哪個按鈕被選中：

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

2.2.3 ButtonBar

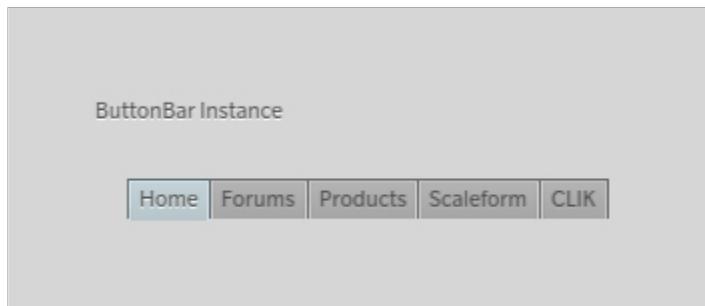


圖 22: 無皮膚按鈕欄 **ButtonBar**.

按鈕欄 **ButtonBar** (`scaleform.clik.controls.ButtonBar`) 與按鈕組 **ButtonGroup** 類似，儘管具有視覺化外觀。也可以基於資料源 **dataProvider**（參見[編程細節](#)描述 **dataProvider** 章節）動態創建按鈕實例。按鈕欄可用與創建動態 **tab** 欄 UI 元素。

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

2.2.3.1 用戶交互

按鈕欄與按鈕組 **ButtonGroup** 具有相同的行為，也不能與用戶直接交互，按鈕欄也由按鈕點擊事件簡介影響。

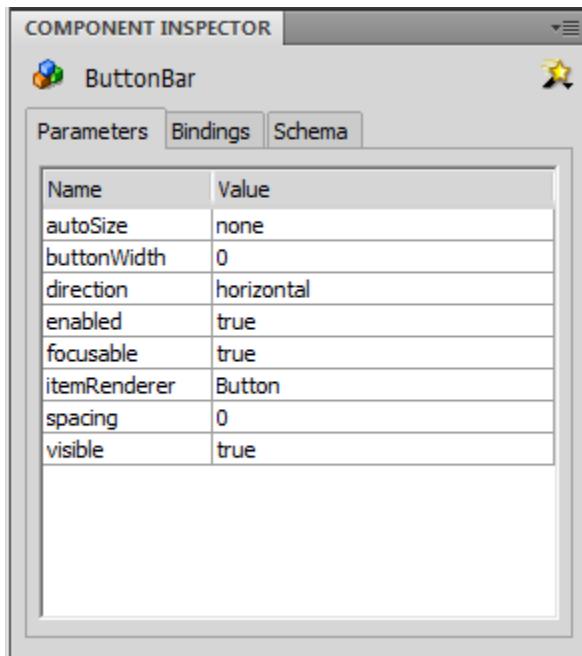
2.2.3.2 元件設置

使用 CLIK 按鈕欄 **ButtonBar** 類的視頻剪輯 **MovieClip** 不需要任何子單元，因為不具有視覺化外觀。

2.2.3.3 狀態

CLIK 按鈕欄不具有任何可視狀態，因為其管理按鈕元件只用來顯示組的狀態。

2.2.3.4 檢查屬性



儘管按鈕欄元件沒有內容（只為 Flash IDE 場景中的一個簡單小圓圈），但包含幾個檢查屬性。主要由按鈕欄 ButtonBar 創建的按鈕實例分佈設置決定。

autoSize	如果設置為 <code>true</code> , 則重新調整 Button(按鈕)實例的大小以適應顯示的標籤。
Direction	按鈕放置。“橫向”(Horizontal) 將把 Button 實例並排放置,而“縱向”(Vertical) 則把這些實例依次堆疊在上面。
buttonWidth	給所有 Button 實例設置一個通用的寬度。如果將 <code>autoSize</code> 設為 <code>true</code> , 則忽略此屬性。
Enabled	如果設置為 <code>false</code> , 就禁用該按鈕。禁用的元件將不再收到使用者輸入。
Focusable	啟用/禁用對於元件的焦點管理。將 <code>focusable</code> (可聚焦)屬性設置為 <code>false</code> 將會取消對 <code>tab</code> 鍵、方向鍵以及基於滑鼠的焦點更改的支援。
itemRenderer	按鈕元件符號的鏈結 ID，該符號將根據按鈕欄的資料分配需求進行實例化
Spacing	按鈕實例間隔，只影響當前方位（見 <code>direction</code> 屬性）
Visible	如果設置為 <code>false</code> , 就隱藏 ButtonBar。

2.2.3.5 事件

所有事件回檔均會收到單個 Event(事件)或 Event 子類參數,該參數包含有相應事件的相關資訊。下列屬性是所有事件所共有的。

- **type** : 事件類型。

- **target** : 事件目標。
- **currentTarget** : 通過一個事件偵聽器積極處理 Event 物件的物件。
- **eventPhase** : 事件流中的當前階段。（EventPhase.CAPTURING_PHASE、EventPhase.AT_TARGET、EventPhase.BUBBLING_PHASE）。
- **bubbles** : 指明事件是不是起泡事件。
- **cancelable** : 指明與事件關聯的行為是否可以避免

ButtonBar 元件產生的事件列表如下所示。事件旁列出的屬性為通用屬性的補充。

ComponentEvent.SHOW	visible(可見)屬性已在運行時設置為 true。
ComponentEvent.HIDE	visible 屬性已在運行時設置為 false。
FocusHandlerEvent.FOCUS_IN	元件已收到焦點。
FocusHandlerEvent.FOCUS_OUT	元件已失去焦點。
IndexEvent.INDEX_CHANGE	已經選定該組中的一個新按鈕。 index:ButtonBar 的選定索引。int 類型。值 -1(如果當前未選定任何專案)到按鈕數量減去 1。 lastIndex:ButtonBar 的以前選定的索引。int 類型。值 -1(如果以前未選定任何專案)到按鈕數量減去 1。 data:選定的dataProvider 的資料值。AS3 Object 類型。
ButtonBarEvent.BUTTON_SELECT	已經選定該組中的一個新按鈕。 index:ButtonBar 的選定索引。Int 類型。值 -1(如果當前未選定任何專案)到按鈕數量減去 1。 renderer:ButotnBar 內現在選定的 Button 實例。Button 類型。

以下例子展示了當在按鈕欄中的按鈕實例被點擊後如何進行事件監聽。

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

2.3 滾動類型

滾動類型包括 ScrollIndicator、ScrollBar 和 Slider。滾動指示器 ScrollIndicator 無需交互用來顯示目標物件元件的滾動位置，而捲軸 ScrollBar 支援用戶交互來改變滾動位置。捲軸由四個按鈕組成：翻頁按鈕、軌道、向上方向箭和向下方向鍵。滑動條 Slider 與捲軸類似，但是只包含一個交互的翻頁按鈕和軌道，不根據目標元件的單元數來改變翻頁按鈕大小。

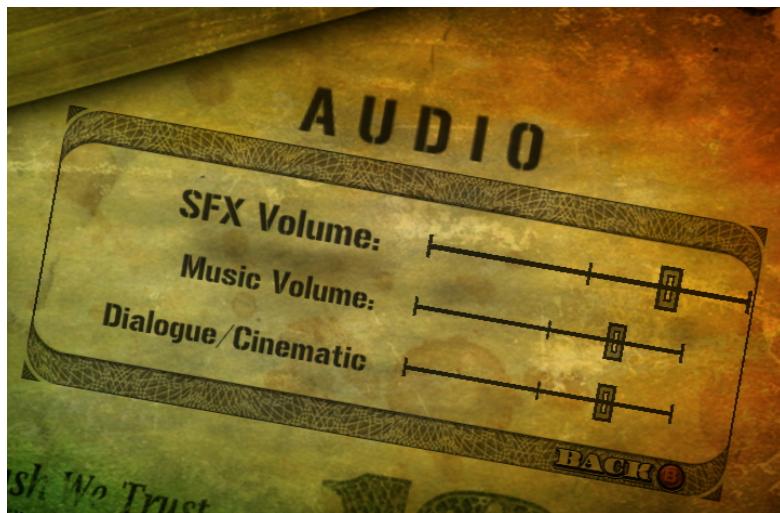


圖 23: 來自 **Mercenaries 2** 的音頻功能表滑動條實例

2.3.1 ScrollIndicator



圖 24: 無皮膚 ScrollIndicator.

CLIK 滾動指示器 ScrollIndicator (`scaleform.clik.controls.ScrollIndicator`) 顯示了其他元件的滾動位置，如多行文本區域 `textField`。可以在文本區域用來自動顯示滾動位置。所有基於列表的元件，包括文本區 `TextArea`，具有捲軸屬性可以由滾動指示器或捲軸（見下節）實例或鏈結 ID 進行顯示。

2.3.1.1 用戶交互

滾動指示器與具有用戶交互功能，因為只用來顯示。

2.3.1.2 元件設置

一個使用 CLIK ScrollIndicator 類的 MovieClip 必須具備下面所列的子單元。也列出了可選元素：

- **thumb**: CLIK 按鈕 Button 類型，滾動指示塊。
- **track** : 動畫剪輯 MovieClip 類型，滾動指示軌道，軌道的邊界決定了滾動塊可以移動的位置。

2.3.1.3 狀態

滾動指示器沒有顯性狀態，使用子單元的狀態：滾動塊和軌道按鈕元件。

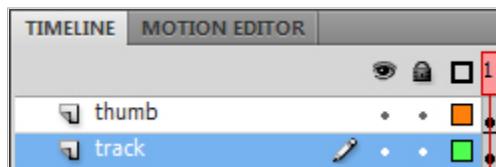
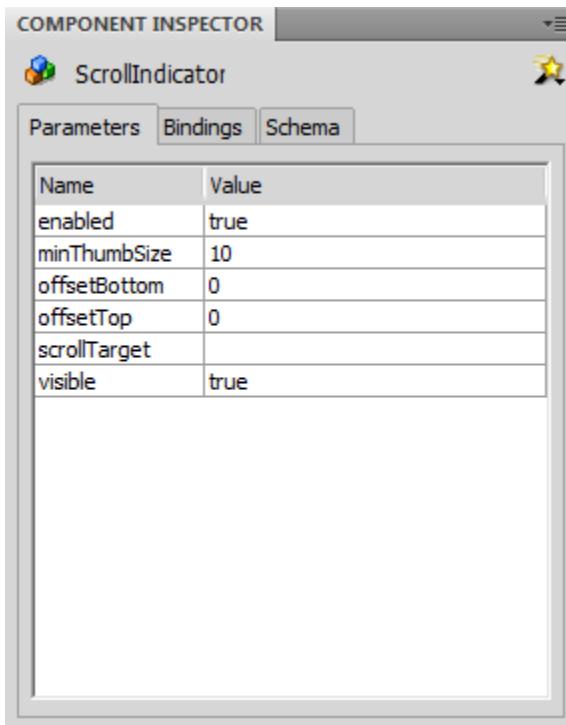


圖 25:滾動指示器 ScrollIndicator 時間軸

2.3.1.4 檢查屬性



滾動指示器的檢查屬性如下所示：

Enabled	如果設置為 <code>false</code> ,就禁用該按鈕。禁用的元件將不再收到使用者輸入。
offsetTop	拖動點頂部偏移。正值將使拖動點向頂部位置的更高處移動。
offsetBottom	拖動點底部偏移。正值將使拖動點向底部位置的更低處移動。
scrollTarget	設置一個文本區域 <code>TextArea</code> 或者常用多行文本域 <code>textField</code> 作為滾動目標自動回應滾動事件。非文本域 <code>textField</code> 類型需要手動更新滾動指示器 <code>ScrollIndicator</code> 屬性。
Visible	如果設置為 <code>false</code> 則隱藏元件

2.3.1.5 事件

所有事件回檔均會收到單個 `Event`(事件)或 `Event` 子類參數,該參數包含有相應事件的相關資訊。下列屬性是所有事件所共有的。

- **`type`** : 事件類型。
-
- **`target`** : 事件目標。
- **`currentTarget`** : 通過一個事件偵聽器積極處理 `Event` 物件的物件。
- **`eventPhase`** : 事件流中的當前階段。(`EventPhase.CAPTURING_PHASE` 、
`EventPhase.AT_TARGET` 、`EventPhase.BUBBLING_PHASE`) 。
- **`bubbles`** : 指明事件是不是起泡事件。
- **`cancelable`** : 指明與事件關聯的行為是否可以避免

`ScrollIndicator` 元件產生的事件列表如下所示。事件旁列出的屬性為通用屬性的補充。

ComponentEvent.SHOW	visible(可見)屬性已在運行時設置為 <code>true</code> 。
ComponentEvent.HIDE	<code>visible</code> 屬性已在運行時設置為 <code>false</code> 。
Event.SCROLL	已經滾動 <code>ScrollIndicator</code> 。

下例顯示如何監聽滾動事件：

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

2.3.2 ScrollBar



圖 26: 無皮膚捲軸 ScrollBar.

CLIK 捲軸 ScrollBar(`scaleform.clik.controls.ScrollBar`)顯示和控制其他元件的滾動位置。通過拖動滾動塊按鈕增加交互功能到滾動指示器，同時還包括向上和向下方向鍵和一個可以點擊的軌道。

2.3.2.1 用戶交互

捲軸 ScrollBar 上的滾動塊可以由滑鼠或類似控制器進行控制，可以在捲軸軌迹邊界之內拖動。當滑鼠游標位於捲軸上方時移動滑鼠滾輪可以進行滾動操作。點擊向上方向鍵可以向上滾動，點擊向下方向鍵可以向下滾動。點擊軌道有兩種行爲：滾動塊繼續沿點擊點位置滾動，或者立即跳轉到點擊點位置並設為拖動狀態。軌道模式由 `trackMode` 檢查屬性決定（見檢查屬性小節）。忽略軌道模式 `trackMode` 設置，按下(`Shift`)鍵並點擊軌道將立即將滾動快移動到滑鼠游標處並設為拖動狀。

2.3.2.2 元件設置

一個使用 Clik ScrollBar 類的 MovieClip 必須具備下面所列的子單元。也列出了可選元素：

- **upArrow**: Clik 按鈕類型，向上滾動按鈕；通常位於捲軸頂部。
- **downArrow**: Clik 按鈕類型，向下滾動按鈕；通常位於捲軸底部。
- **thumb**: Clik 按鈕類型，捲軸上的滾動塊。
- **track**: Clik 按鈕類型，捲軸軌道，其邊界決定了滾動塊的活動方位。

2.3.2.3 狀態

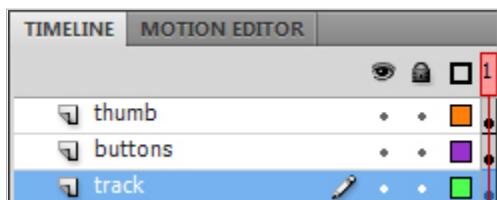
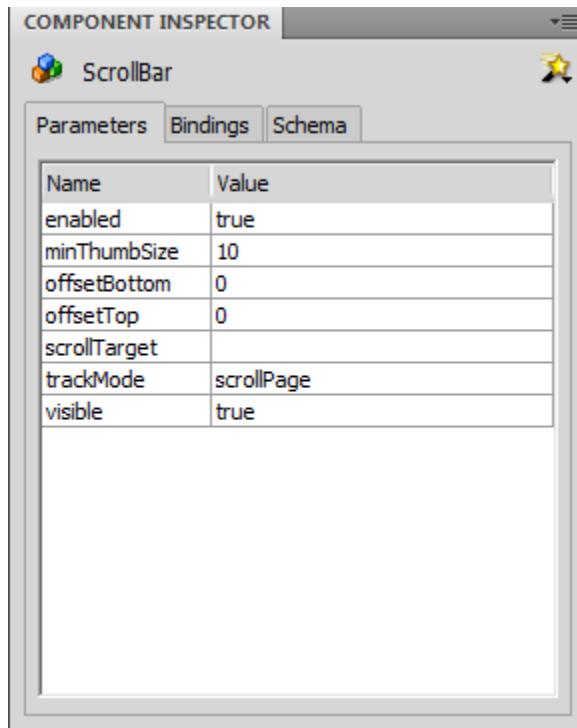


圖 27: 捲軸 ScrollBar 時間軸

捲軸 ScrollBar，與滾動指示器 ScrollIndicator 類似，不需要顯性狀態。使用子單元的狀態包括：滾動塊、向上按鈕、向下按鈕和軌道按鈕元件。

2.3.2.4 檢查屬性



捲軸檢查屬性和滾動指示器類似，只增加一條, **trackMode**:

Enabled	如果設置為 <code>false</code> ,就禁用元件。禁用的元件將不再收到使用者輸入。
offsetTop	拖動點頂部偏移。正值將使拖動點向頂部位置的更高處移動。
offsetBottom	拖動點底部偏移。正值將使拖動點向底部位置的更低處移動。
scrollTarget	回應滾動事件時設置文本區 <code>TextArea</code> 或常用多汗文本域 <code>textField</code> 的滾動目標位置
trackMode	當用戶用滑鼠點擊捲軸，滾動頁 <code>scrollPage</code> 設置將導致滾動塊翻頁知道釋放師表。 <code>scrollToCursor</code> 設置可以使滾動塊立即跳轉到滑鼠游標所在位置並轉入拖動模式直到釋放滑鼠。
Visible	如果設置為 <code>false</code> 則隱藏元件

2.3.2.5 事件

所有事件回檔均會收到單個 `Event`(事件)或 `Event` 子類參數,該參數包含有相應事件的相關資訊。下列屬性是所有事件所共有的。

- **type** : 事件類型。
- **target** : 事件目標。
- **currentTarget** : 通過一個事件偵聽器積極處理 `Event` 物件的物件。

- **eventPhase**：事件流中的當前階段。（EventPhase.CAPTURING_PHASE、EventPhase.AT_TARGET、EventPhase.BUBBLING_PHASE）。
- **bubbles**：指明事件是不是起泡事件。
- **cancelable**：指明與事件關聯的行為是否可以避免

ScrollBar 元件產生的事件列表如下所示。事件旁列出的屬性為通用屬性的補充。

ComponentEvent.SHOW	visible(可見)屬性已在運行時設置為 true。
ComponentEvent.HIDE	visible 屬性已在運行時設置為 false。
Event.SCROLL	已經滾動 ScrollBar。

下列代碼指示如何監聽滾動事件：

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

2.3.3 Slider



圖 28: 無皮膚滑動條 Slider.

滑動條 Slider(scaleform.clik.controls.Slider)顯示了一個範圍內的數值，用滾動塊來表示值，通過拖動來改變值。

2.3.3.1 用戶交互

滑動塊可以被滑鼠或類似控制器在滑動軌道邊界內拖動。在軌道上點擊可以立即移動滾動塊到滑鼠游標所在位置並設置為拖動狀。當獲得焦點時，左右方向鍵在對應的方向移動滾動塊，而 home 和 end 鍵可以將滾動塊移動到捲軸的開始和末尾處。

2.3.3.2 元件設置

使用 CLIK Slider 類的 MovieClip 必須具備下面所列的子單元。也列出了可選元素：

- **thumb** : CLIK 按鈕類型，滑動快。
- **Track** : CLIK 按鈕類型，滑動軌道邊界決定了滑動塊可以移動位置。

2.3.3.3 狀態

與滾動指示器和捲軸類似，滑動條沒有顯性狀態，使用子單元的狀態包括：滾動塊和軌道按鈕元件。

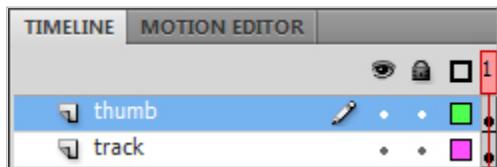
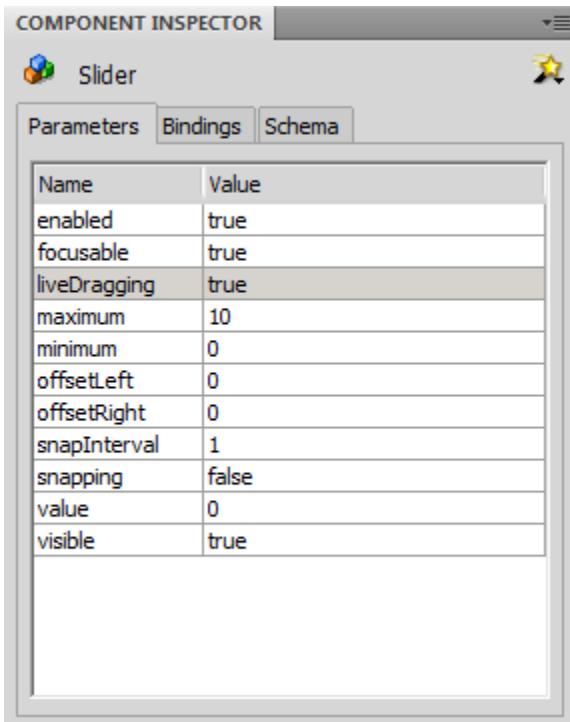


图 29:滑动条 Slider 时间轴

2.3.3.4 檢查屬性



滑動條 Slider 元件的檢查屬性如下所示：

Enabled

如果設置為 `false`,就禁用元件。禁用的元件將不再收到使用者輸入。

Focusable	啟用/禁用對於元件的焦點管理。將 focusable(可聚焦)屬性設置為 false 將會取消對 tab 鍵、方向鍵以及基於滑鼠的焦點更改的支援。
Value	滑動條 Slider 顯示的數值
Minimum	滑動條範圍內的最小值
Maximum	滑動條範圍內的最大值
Snapping	如果設置為 true，滾動快將按照多被間隔進行移動
snapInterval	滾動間隔決定滾動塊跳轉間隔函數，設置為 false 時該值無效
liveDragging	如果設置為 true，拖動滾動塊時則滑動條 Slider 將產生一個改變事件，滑動條只在拖動結束後產生事件改變
offsetLeft	拖動點向左偏移。正值將會向內擠壓拖動點。
offsetRight	拖動點向右偏移。正值將會向內擠壓拖動點。
Visible	如果設置為 false 則隱藏元件

2.3.3.5 事件

所有事件回檔均會收到單個 Event(事件)或 Event 子類參數,該參數包含有相應事件的相關資訊。下列屬性是所有事件所共有的。

- **type** : 事件類型。
- **target** : 事件目標。
- **currentTarget** : 通過一個事件偵聽器積極處理 Event 物件的物件。
- **eventPhase** : 事件流中的當前階段。（EventPhase.CAPTURING_PHASE、EventPhase.AT_TARGET、EventPhase.BUBBLING_PHASE）。
- **bubbles** : 指明事件是不是起泡事件。
- **cancelable** : 指明與事件關聯的行為是否可以避免

Slider 元件產生的事件列表如下所示。事件旁列出的屬性為通用屬性的補充。

ComponentEvent.SHOW	visible(可見)屬性已在運行時設置為 true。
ComponentEvent.HIDE	visible 屬性已在運行時設置為 false。
ComponentEvent.STATE_CHANGE	元件的狀態已更改。
FocusHandlerEvent.FOCUS_IN	該按鈕收到了焦點。
FocusHandlerEvent.FOCUS_OUT	該按鈕失去了焦點。
SliderEvent.VALUE_CHANGE	改變滑動條 Slider's 值

下例顯示了如何監聽滑動條 Slider 值的改變：

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

}

2.4 列表類型

CLIK 列表類型包括 NumericStepper、OptionStepper、ScrollingList、TileList 和 DropdownMenu 元件。所有元件，除了 NumericStepper，都用 DataProvider 來管理資訊並顯示。ListItemRenderer 元件也包括在此目錄中，因為 ScrollingList 和 TileList 元件需要用來顯示列表專案。

NumericStepper 和 OptionStepper 元件只用一次顯示一個單元，而 ScrollingList 和 TileList 能顯示多個單元。最後兩個元件可以支援 ScrollIndicator 或 ScrollBar 元件。DropdownMenu 元件在靜止狀態顯示一個單元，但是在打開時使用 ScrollingList 或 TileList 可以顯示多個單元。



圖 30:來自 *Mercenaries 2* 的滾動指示器滾動列表實例

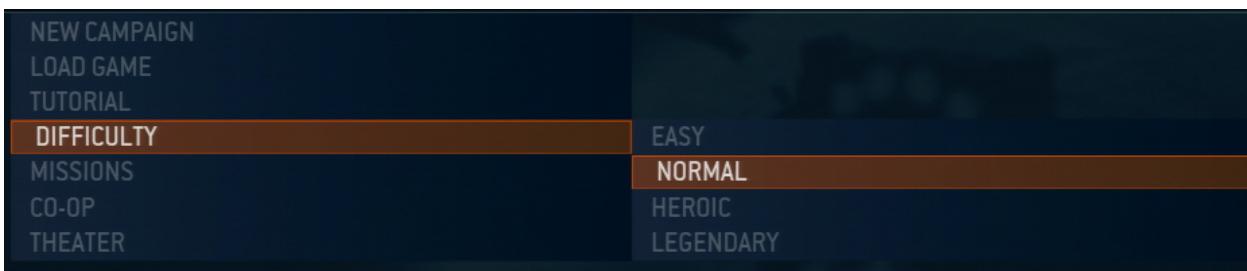


圖 31:來自 *Halo Wars* 的'Difficulty Settings'下拉功能表實例

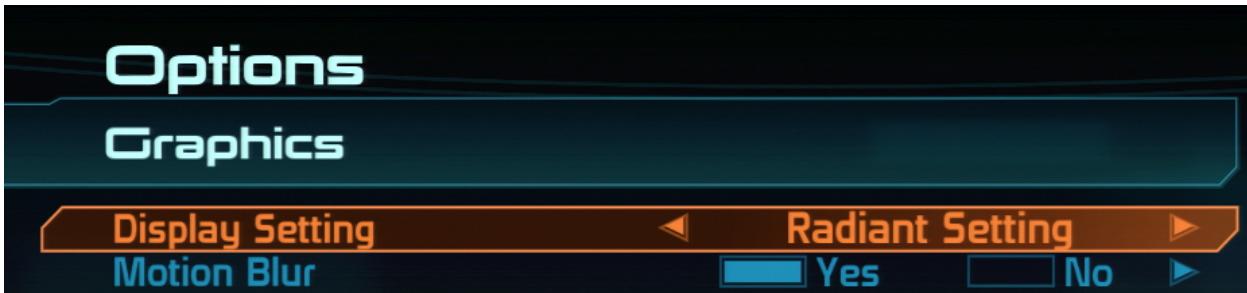


圖 32:來自 *Mass Effect* 的'Display Setting'選項實例

2.4.1 NumericStepper



圖 33:無皮膚 Numeric Stepper.

NumericStepper (scaleform.clik.controls.NumericStepper)顯示在分配範圍內的一個數位，支援任意步長值的增加和減少。

2.4.1.1 用戶交互

NumericStepper 包括兩個方向鍵可以通過滑鼠或類似控制器點擊以改變數位值。當獲得焦點時，數位值可以通過鍵盤左右方向鍵或類似控制器改變。這些鍵可以根據步長增加和減少當前值。按下(Home) 和 (End)鍵或類似控制器將在對應的最大值和最小值之間改變數位值。

2.4.1.2 元件設置

使用 CLIK NumericStepper 類的 MovieClip 必須具備下面所列的子單元。也列出了可選元素：

- **textField:** TextField 類型，用來顯示當前值。
- **nextBtn:** CLIK 按鈕 Button 類型，點擊可以根據步長增加當前值。
- **prevBtn:** CLIK 按鈕 Button 類型，點擊可以根據步長減少當前值。

2.4.1.3 狀態

NumericStepper 元件支援三種狀態，基於焦點和禁止屬性：

- **default** 或 enabled 狀態；
- **focused** 狀態，突出顯示 textField 區域；
- **disabled** 狀態；

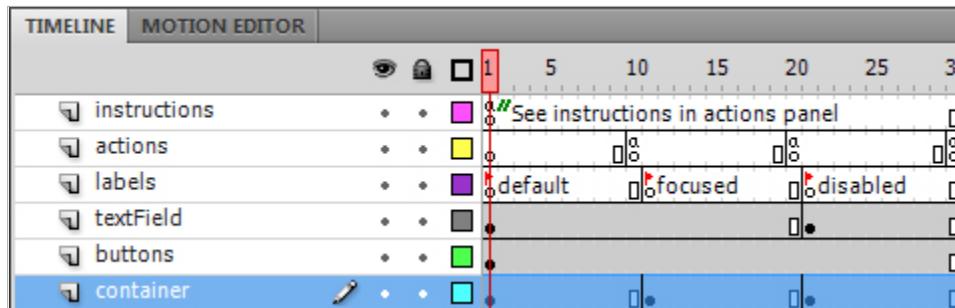
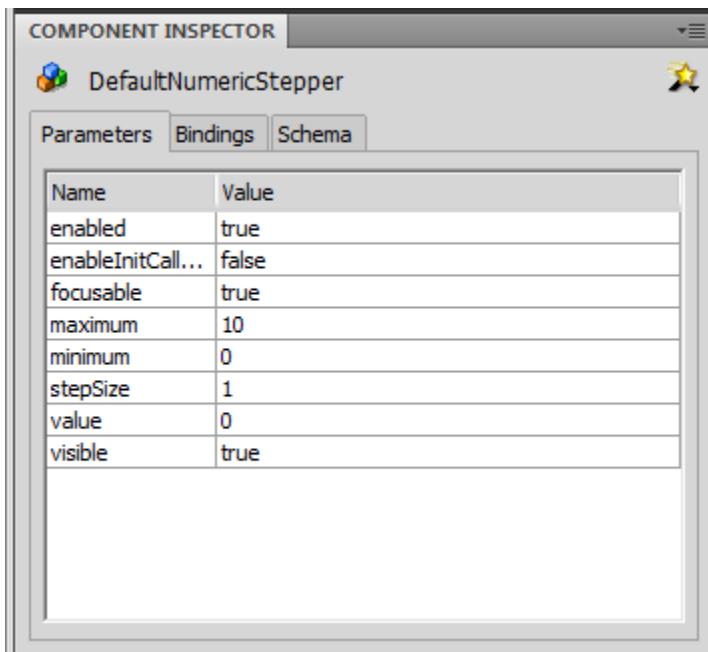


圖 34: NumericStepper 時間軸

2.4.1.4 檢查屬性



從數位步長 NumericStepper 元件繼承來的動畫剪輯 MovieClip 將有以下檢查屬性：

Enabled	如果設置為 <code>false</code> ,就禁用元件。禁用的元件將不再收到使用者輸入。
Focusable	啟用/禁用對於元件的焦點管理。將 <code>focusable</code> (可聚焦)屬性設置為 <code>false</code> 將會取消對 <code>tab</code> 鍵、方向鍵以及基於滑鼠的焦點更改的支援。
minimum	NumericStepper 數值範圍內的最小值
maximum	NumericStepper 數值範圍內的最大值
stepSize	每次點擊 NumericStepper 的一個按鈕,就應遞增/遞減多少 NumericStepper 的值。
Value	滑動條 Slider 顯示的數值
Visible	如果設置為 <code>false</code> 則隱藏元件

2.4.1.5 事件

所有事件回檔均會收到單個 `Event`(事件)或 `Event` 子類參數,該參數包含有相應事件的相關資訊。下列屬性是所有事件所共有的。

- **type** : 事件類型。
- **target** : 事件目標。
- **currentTarget** : 通過一個事件偵聽器積極處理 `Event` 物件的物件。
- **eventPhase** : 事件流中的當前階段。（`EventPhase.CAPTURING_PHASE`、`EventPhase.AT_TARGET`、`EventPhase.BUBBLING_PHASE`）。

- **bubbles**：指明事件是不是起泡事件。
- **cancelable**：指明與事件關聯的行為是否可以避免

NumericStepper 元件產生的事件列表如下所示。事件旁列出的屬性為通用屬性的補充。

ComponentEvent.SHOW	visible(可見)屬性已在運行時設置為 true。
ComponentEvent.HIDE	visible 屬性已在運行時設置為 false。
ComponentEvent.STATE_CHANGE	元件的狀態已更改。
FocusHandlerEvent.FOCUS_IN	元件已收到焦點。
FocusHandlerEvent.FOCUS_OUT	元件已失去焦點。
IndexEvent.INDEX_CHANGE	<p>NumericStepper 的值已發生變化。</p> <p>index:NumericStepper 的選定的索引。int 類型。值 -1(如果當前未選定任何專案)到按鈕數量減去 1。</p> <p>lastIndex:NumericStepper 的以前選定的索引。int 類型。值 -1(如果以前未選定任何專案)到按鈕數量減去 1。</p> <p>data:選定的dataProvider 的資料值。AS3 Object 類型。</p>

下例顯示了對 NumericStepper 值改變的監聽：

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

2.4.2 OptionStepper



圖 35: 無皮膚 OptionStepper.

OptionStepper (scaleform.clik.controls.OptionStepper),與 NumericStepper 類似，用來顯示一個值，但可以顯示更多資訊。使用資料源dataProvider 實例查詢當前值；因此支援不同類型元素的任意數值。應

用 OptionStepper 的選擇項索引屬性通過代碼對顯示值進行設置，該索引是附在資料提供者中的一個從零開始的索引。資料源 `dataProvider` 通過代碼進行指派，如下例所示：

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

2.4.2.1 用戶交互

與 NumericStepper 類似，OptionStepper 包括兩個方向鍵，可以通過滑鼠或類似控制器點擊改變當前值。當獲得焦點時，當前值可以通過左右方向鍵或類似控制器進行改變。這些鍵按照順序向前或者向後改變當前值。按下(Home)和(End)鍵或類似控制器可以將當前值改變為資料源 `dataProvider` 的第一個和最後一個單元。

2.4.2.2 元件設置

使用 CLIK NumericStepper 類的動畫剪輯 MovieClip 必須具備下面所列的子單元。也列出了可選元素：

- **textField** : TextField 類型，用來顯示當前值。
- **nextBtn** : CLIK 按鈕 Button 類型，改變當前值為資料源 `dataProvider` 中的下一個值。
- **prevBtn** : CLIK 按鈕 Button 類型，當前值為資料源 `dataProvider` 中的上一個值。

2.4.2.3 狀態

OptionStepper 元件支援三種狀態，基於焦點和禁止屬性：

- **default** 或 **enabled** 狀態；
- **focused** 狀態，突出顯示 `textField` 區域；
- **disabled** 狀態；

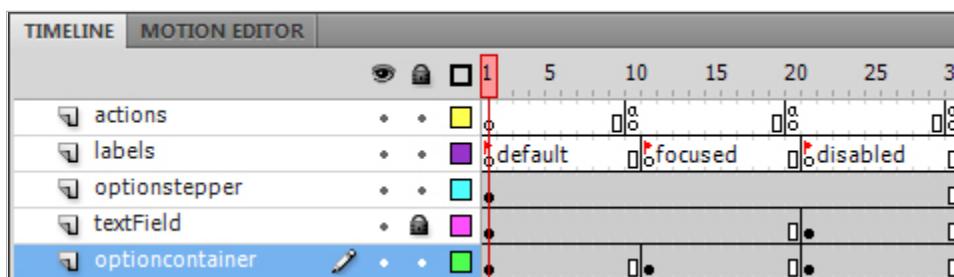
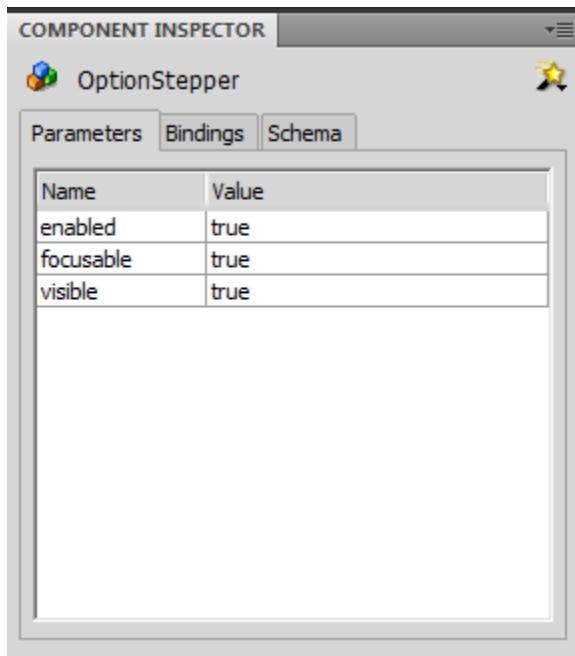


圖 36: OptionStepper 時間軸

2.4.2.4 檢查屬性



來自選項步長 OptionStepper 元件的視頻剪輯 MovieClip 具有以下檢查屬性。

Enabled	如果設置為 <code>false</code> ,就禁用元件。禁用的元件將不再收到使用者輸入。
Focusable	啟用/禁用對於元件的焦點管理。將 <code>focusable</code> (可聚焦)屬性設置為 <code>false</code> 將會取消對 <code>tab</code> 鍵、方向鍵以及基於滑鼠的焦點更改的支援。
Visible	如果設置為 <code>false</code> 則隱藏元件

2.4.2.5 事件

所有事件回檔均會收到單個 `Event`(事件)或 `Event` 子類參數,該參數包含有相應事件的相關資訊。下列屬性是所有事件所共有的。

- **`type`** : 事件類型。
- **`target`** : 事件目標。
- **`currentTarget`** : 通過一個事件偵聽器積極處理 `Event` 物件的物件。
- **`eventPhase`** : 事件流中的當前階段。 (`EventPhase.CAPTURING_PHASE`、
`EventPhase.AT_TARGET`、`EventPhase.BUBBLING_PHASE`) 。
- **`bubbles`** : 指明事件是不是起泡事件。
- **`cancelable`** : 指明與事件關聯的行為是否可以避免

OptionStepper 元件產生的事件列表如下所示。事件旁列出的屬性為通用屬性的補充。

ComponentEvent.SHOW	<code>visible</code> (可見)屬性已在運行時設置為 <code>true</code> 。
ComponentEvent.HIDE	<code>visible</code> 屬性已在運行時設置為 <code>false</code> 。

ComponentEvent.STATE_CHANGE	元件的狀態已更改。
FocusHandlerEvent.FOCUS_IN	元件已收到焦點。
FocusHandlerEvent.FOCUS_OUT	元件已失去焦點。
IndexEvent.INDEX_CHANGE	<p>OptionStepper 的值已發生變化。</p> <p>index:NumericStepper 的選定的索引。int 類型。值 -1(如果當前未選定任何專案)到按鈕數量減去 1。</p> <p>lastIndex:NumericStepper 的以前選定的索引。int 類型。值 -1(如果以前未選定任何專案)到按鈕數量減去 1。</p> <p>data:選定的dataProvider 的資料值。AS3 Object 類型。</p>

下列顯示了 OptionStepper 值改變的監聽：

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

2.4.3 ListItemRenderer



图 2:无皮肤 **ListItemRenderer**.

列表項渲染器 **ListItemRenderer** (scaleform.clik.controls.ListItemRenderer)從 CLIK 按鈕 **Button** 類繼承而來，擴展了列表相關特性，在容器元件中需要用到。但是，不是作為一個單獨組建設計，只與滾動列表 **ScrollingList**、標題列表 **TitleList** 和下拉功能表 **DropdownMenu** 元件共同使用。

2.4.3.1 用戶交互

由於 `ListItemRenderer` 從按鈕 `Button` 元件繼承而來，與按鈕具有相同的用戶交互如使用滑鼠點擊。滾動滑鼠游標到 `ListItemRenderer` 或將游標移開都將對元件產生影響，拖動滑鼠游標效果也一樣。`ListItemRenderer` 的容器元件定義了鍵盤或類似控制器的交互。

2.4.3.2 元件設置

使用 CLIK `ListItemRenderer` 類的動畫剪輯 `MovieClip` 必須具備下面所列的子單元。也列出了可選元素：

- **`textField`**：(可選) `TextField` 類型，列表項標簽。
- **`focusIndicator`**: (可選) `MovieClip` 類型，一個獨立的動畫剪輯 `MovieClip` 用來顯示焦點狀態。如果被使用，該動畫剪輯必須擁有兩個幀分別命名為：“`show`” 和 “`hide`”。

2.4.3.3 狀態

由於可以在一個容器元件內部被選擇，列表項渲染器 `ListItemRenderer` 需要關鍵幀為 `selected` 設置來表示其選擇狀態。元件狀態包括：

- **`up`** 或默認狀態；
- 當滑鼠箭頭在元件上方或者獲得焦點時為 `over` 狀態；
- 當按鈕被點擊時候為 `down` 狀態；
- **`disabled`** 狀態；
- **`selected_up`** 或者默認狀態；
- 當滑鼠箭頭位於元件上方或獲得焦點時為 `selected_over` 狀態；
- 當按鈕被按下時為 `selected_down` 狀態；
- **`selected_disabled`** 狀態；

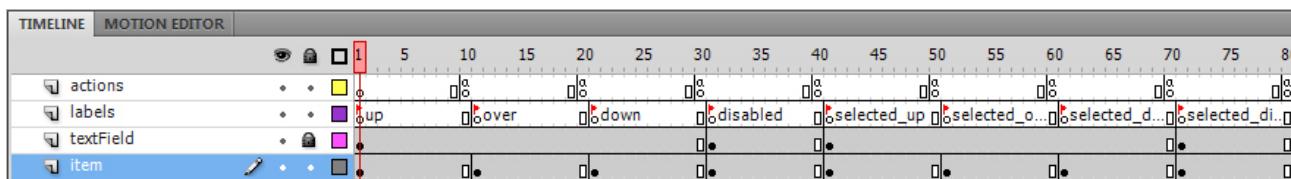
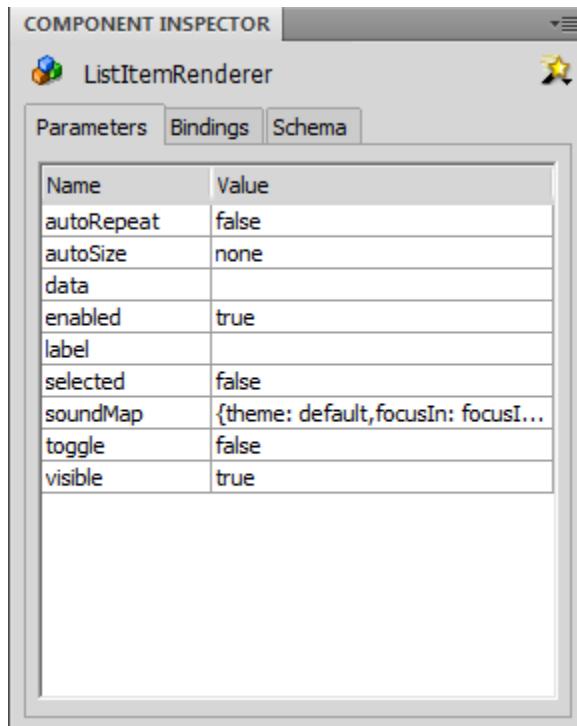


圖 38: `ListItemRenderer` 時間軸

這裏為選擇按鈕 `ListItemRenderer` 所需要的最少關鍵幀設置。按鈕 `Button` 元件支援狀態和關鍵幀的擴展設置，與 `ListItemRenderer` 元件相同，在 [CLIK 按鈕入門](#) 文檔中有詳細描述。

2.4.3.4 檢查屬性



由於列表項渲染器 **ListItemRenderer** 由一個容器元件按控制，用戶無需手動配置，值包含了按鈕的一小部分檢查屬性。

autoRepeat	確定在按下並保持住時 ListItemRender 是否發出 "click"(點擊)事件。
autoSize	確定 ListItemRender 是否會縮放以適應其所包含的文本以及已調整大小的 ListItemRender 將向哪個方向對齊。將 autoSize 屬性設置為 autoSize=TextFieldAutoSize.NONE 將會使其當前大小保持不變。
Data	與 ListItemRender 相關的資料。在一個 List(清單) 元件 (ScrollingList / TileList) 中使用 ListItemRender 時此屬性尤其有用。
Enabled	如果設置為 false ,就禁用元件。禁用的元件將不再收到使用者輸入。
Label	ListItemRender 標簽設置
selected	設置 ListItemRender 的選定狀態。 ListItemRender 可以有兩組滑鼠狀態:已選定狀態和未選定狀態。當一個 ListItemRender 的 toggle 屬性為 true 時,點擊該按鈕時就會更改已選定的狀態,不過,即使 toggle 屬性為 false ,也可以使用 ActionScript 設置已選定狀態。
Toggle	設置 ListItemRender 的 toggle 屬性。如果設置為 true ,則 ListItemRender 的行為將會與一個切換按鈕相同。
Visible	如果設置為 false 則隱藏元件

2.4.3.5 事件

所有事件回檔均會收到單個 Event(事件)或 Event 子類參數,該參數包含有相應事件的相關資訊。下列屬性是所有事件所共有的。

- **type** : 事件類型。
- **target** : 事件目標。
- **currentTarget** : 通過一個事件偵聽器積極處理 Event 物件的物件。
- **eventPhase** : 事件流中的當前階段。（EventPhase.CAPTURING_PHASE、EventPhase.AT_TARGET、EventPhase.BUBBLING_PHASE）。
- **bubbles** : 指明事件是不是起泡事件。
- **cancelable** : 指明與事件關聯的行為是否可以避免

下麵列出 ListItemRenderer 元件生成的事件。除常見屬性外,還提供了該事件旁邊列出的屬性。

一般情況下,使用者不要對 ListItemRenderer's 發出的事件進行偵聽,而應將一個事件偵聽器添加到 List(清單)本身。當與其子 ListItemRenderer 之一發生交互時,清單就會發出包含 ITEM_PRESS 和 ITEM_CLICK 的 ListEvents。這使使用者可以將一個 EventListener 添加到針對某個滑鼠點擊事件(ListEvent.ITEM_CLICK)的清單,而不是上述清單內每個 ListItemRenderer 上的一個 EventListener。

ComponentEvent.SHOW	visible(可見)屬性已在運行時設置為 true。
ComponentEvent.HIDE	visible 屬性已在運行時設置為 false。
ComponentEvent.STATE_CHANGE	元件的狀態已更改。
FocusHandlerEvent.FOCUS_IN	元件已收到焦點。
FocusHandlerEvent.FOCUS_OUT	元件已失去焦點。
Event.SELECT	選定的屬性已發生變化。
MouseEvent.ROLL_OVER	滑鼠箭頭在按鈕上方滾動。 mouseldx :用來生成該事件的滑鼠游標的索引(僅適用於多滑鼠游標環境)。uint 類型。僅限於 Scaleform,需要將該事件歸於 MouseEventEx。 buttonIdx :指明為哪個按鈕生成該事件(基於零的索引)。僅限於 Scaleform,需要將該事件歸於 MouseEventEx。
MouseEvent.ROLL_OUT	滑鼠箭頭從按鈕移開。 mouseldx :用來生成該事件的滑鼠游標的索引(僅適用於多滑鼠游標環境)。uint 類型。僅限於 Scaleform,需要將該事件歸於 MouseEventEx。 buttonIdx :指明為哪個按鈕生成該事件(基於零的索引)。僅限於 Scaleform,需要將該事件歸於 MouseEventEx。

ButtonEvent.PRESS	<p>按鈕被點擊。</p> <p>controllerIdx:用來生成該事件的控制器的索引(僅適用于多滑鼠游標環境)。uint 類型。</p> <p>isKeyboard:為 true - 假如事件是由電腦鍵盤 (Keyboard)/遊戲鍵盤 (Gamepad) 生成的;為 false - 假如事件是由滑鼠生成的。 isRepeat:為 true - 假如事件是由一個被按住的 autoRepeating 按鈕生成的;為 false - 該按鈕當前不重複。</p>
MouseEvent.DOUBLE_CLICK	<p>按鈕被雙擊。只在 <i>doubleClickEnabled</i> 屬性為 true 時被觸發。</p> <p>mouselidx:用來生成該事件的滑鼠游標的索引(僅適用于多滑鼠游標環境)。uint 類型。僅限於 Scaleform,需要將該事件歸於 MouseEventEx。</p> <p>buttonIdx:指明為哪個按鈕生成該事件(基於零的索引)。僅限於 Scaleform,需要將該事件歸於 MouseEventEx。</p>
ButtonEvent.CLICK	<p>按鈕被點擊。</p> <p>controllerIdx:用來生成該事件的控制器的索引(僅適用于多滑鼠游標環境)。uint 類型。</p> <p>isKeyboard:為 true - 假如事件是由電腦鍵盤 (Keyboard)/遊戲鍵盤 (Gamepad) 生成的;為 false - 假如事件是由滑鼠生成的。 isRepeat:為 true - 假如事件是由一個被按住的 autoRepeating 按鈕生成的;為 false - 該按鈕當前不重複。</p>
ButtonEvent.DRAG_OVER	<p>滑鼠箭頭拖動到按鈕上方 (滑鼠左鍵被按下)</p> <p>controllerIdx:用來生成該事件的控制器的索引(僅適用于多滑鼠游標環境)。uint 類型。</p>
ButtonEvent.DRAG_OUT	<p>滑鼠箭頭從按鈕拖開 (滑鼠左鍵被按下)。</p> <p>controllerIdx:用來生成該事件的控制器的索引(僅適用于多滑鼠游標環境)。uint 類型。</p>
ButtonEvent.RELEASE_OUTSIDE	<p>滑鼠箭頭從按鈕拖開滑鼠左鍵鬆開。</p> <p>controllerIdx:用來生成該事件的控制器的索引(僅適用于多滑鼠游標環境)。uint 類型。</p>

2.4.4 ScrollingList



圖 39: 無皮膚滾動列表 ScrollingList.

滾動列表 ScrollingList (scaleform.clik.controls.ScrollingList)為一個元件可以滾動其元素。可以自身展示列表項或者使用現存的場景中的列表項。可以附加滾動指示器 ScrollIndicator 或捲軸 ScrollBar 元件提供滾動反饋和控制功能。該元件可與資料源dataProvider 一起使用。資料源通過代碼指派，如下面實例所示：

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

默認情況下滾動列表 crollingList 使用列表項渲染器 ListItemRenderer 元件作為內容。因此列表項渲染器必須在 FLA 文件庫中存在，除非列表項渲染器檢查屬性改變成另外一個元件。見檢查屬性小節獲得更多資訊。

2.4.4.1 用戶交互

點擊一個列表項或者一個附屬的捲軸 ScrollBar 實例將焦點傳遞到滾動列表 ScrollingList 元件。當獲得焦點，按下向上向下方向鍵或類似的控制器控制列表選項的滾動選擇一個元素。如果沒有選中元素，則默認選中頂部元素。如果游標位於滾動列表 ScrollingList 頂部則滑鼠滾輪能夠在列表中滾動。

在邊界上的滾動行爲由滾動列表 ScrollingList 的 *wrapping* 屬性決定，無檢查項。如果 *wrapping* 設置為 “normal”，當選項達到列表開始處或者結束處則焦點將離開元件。如果 *wrapping* 設置為 “wrap”，則選項將圍繞開始處或結束處。如果 *wrapping* 設置為 “stick”，則選項在達到資料結尾時停止，焦點不轉移到相鄰元件。

按下鍵盤(Page Up) 和 (Page down)或類似控制器將按頁滾動選項，例如，列表中的可視選項數。按下 (Home) 和 (End)鍵，或者類似控制器，將滾動列表到相應元素的開始和末尾處。與附屬捲軸 ScrollBar 元件交互將按預期影響滾動列表 ScrollingList。見捲軸 ScrollBar 小節瞭解其用戶交互功能。

開發者可以簡單得將遊戲控制杆映射到鍵盤和滑鼠控制鍵。例如，鍵盤方向鍵通常引導控制器上的 D-Pad。這個對應關係可以使 CLIK 構建的 UI 介面工作在更加廣泛的平臺之上。

2.4.4.2 元件設置

滾動列表 ScrollingList 不需要任何子單元，但是，在場景上的滾動列表 ScrollingList 元件上放置一個元件或者調整元件大小時一個視覺化背景就能起到輔助作用。

2.4.4.3 狀態

滾動列表 ScrollingList 元件支援三種狀態，基於焦點和禁止屬性：

- **default** 或 enable 狀態；
- **focused** 狀態，通常突出顯示元件邊框區域；
- **disabled** 狀態。

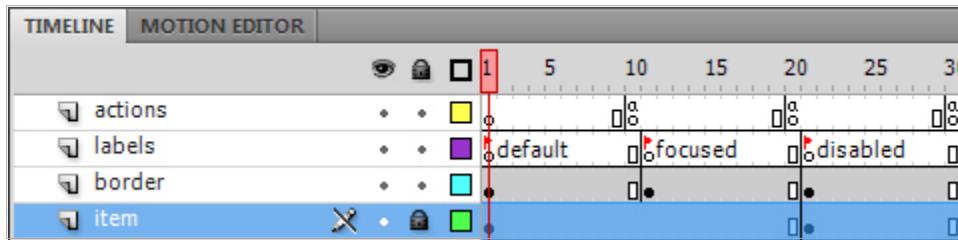
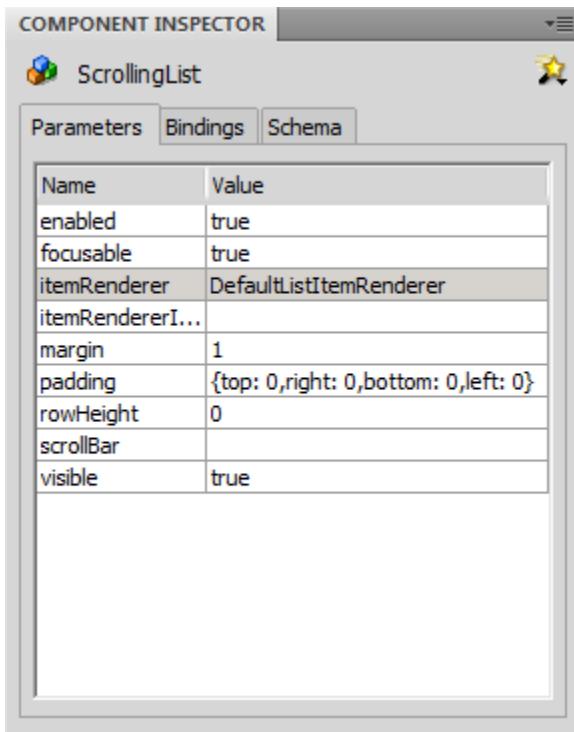


圖 40:滾動列表 ScrollingList 時間軸

2.4.4.4 檢查屬性



來自滾動列表 ScrollingList 元件的視頻剪輯 MovieClip 具有以下檢查屬性。

Enabled	如果設置為 <code>false</code> ,就禁用元件。禁用的元件將不再收到使用者輸入。
Focusable	啟用/禁用對於元件的焦點管理。將 <code>focusable</code> (可聚焦)屬性設置為 <code>false</code> 將會取消對 <code>tab</code> 鍵、方向鍵以及基於滑鼠的焦點更改的支援。
itemRenderer	<code>ListItemRenderer</code> 的符號名稱。用來在內部創建清單項實例。如果設置了 <code>itemRendererInstanceName</code> 屬性(即如果使用外部 <code>ListItemRenderers</code>),則沒有影響。
itemRendererInstanceName	列表項渲染器擴展字首，與滾動列表 <code>ScrollingList</code> 元件一起使用。場景中的列表項實例必須用該屬性值作為字首。如果屬性設置為 ‘r’，則所有本元件使用的列表項實例必須具有以下值： ‘r1’、‘r2’、‘r3’.....第一個項應該為數值 1。
Margin	內部創建的列表元件和列表項元件的頁面邊緣。如果設置了 <code>rendererInstanceName</code> 屬性該值不產生影響。
Padding	針對清單項的額外填充 (Padding)。如果設置了 <code>itemRendererInstanceName</code> 屬性(即如果使用外部 <code>ListItemRenderer</code>),則此值沒有影響。填充不會影響自動生成的

	ScrollBar(捲軸)。
rowHeight	內部創建的清單項實例的高度。如果設置了 itemRendererInstanceName 屬性(即如果使用外部 ListItemRenderer), 則此值沒有影響。
Visible	如果設置為 false , 就隱藏元件。這不會隱藏附加的 ScrollBar 或任何外部 ListItemRenderer 。

2.4.4.5 事件

所有事件回檔均會收到單個 Event(事件)或 Event 子類參數, 該參數包含有相應事件的相關資訊。下列屬性是所有事件所共有的。

- **type** : 事件類型。
- **target** : 事件目標。
- **currentTarget** : 通過一個事件偵聽器積極處理 Event 物件的物件。
- **eventPhase** : 事件流中的當前階段。 (EventPhase.CAPTURING_PHASE、EventPhase.AT_TARGET、EventPhase.BUBBLING_PHASE) 。
- **bubbles** : 指明事件是不是起泡事件。
- **cancelable** : 指明與事件關聯的行為是否可以避免

滾動列表 ScrollingList 元件產生的事件列表如下所示。事件旁列出的屬性為通用屬性的補充。

ComponentEvent.SHOW	visible(可見)屬性已在運行時設置為 true。
ComponentEvent.HIDE	visible 屬性已在運行時設置為 false。
ComponentEvent.STATE_CHANGE	元件的狀態已更改。
FocusHandlerEvent.FOCUS_IN	元件已收到焦點。
FocusHandlerEvent.FOCUS_OUT	元件已失去焦點。
ListEvent.INDEX_CHANGE	選定的索引已發生變化。 <ul style="list-style-type: none"> • itemRenderer: 按兩下過的清單項。 IListItemRenderer 類型。 • itemData: 與清單項關聯的資料。此值是從清單的 dataProvider 中檢索的。 ActionScript Object 類型。 • index: 清單的新的選定索引。int 類型。值 -1(未設置選定的索引)到清單項數量減去 1。 • controllerIdx: 用來生成該事件的控制器/滑鼠的

	索引(僅適用於多滑鼠游標環境)。
ListEvent.ITEM_PRESS	已按下一个清單項。 <ul style="list-style-type: none">• itemRenderer: 按兩下過的清單項。IListItemRenderer 類型。• itemData:與清單項關聯的資料。此值是從清單的dataProvider 中檢索的。ActionScript Object 類型。• index:清單的新的選定索引。int 類型。值 -1(未設置選定的索引)到清單項數量減去 1。• controllerIdx:用來生成該事件的控制器/滑鼠的索引(僅適用於多滑鼠游標環境)。
ListEvent.ITEM_CLICK	已點擊一個清單項。 <ul style="list-style-type: none">• itemRenderer: 按兩下過的清單項。IListItemRenderer 類型。• itemData:與清單項關聯的資料。此值是從清單的dataProvider 中檢索的。ActionScript Object 類型。• index:清單的新的選定索引。int 類型。值 -1(未設置選定的索引)到清單項數量減去 1。• controllerIdx:用來生成該事件的控制器/滑鼠的索引(僅適用於多滑鼠游標環境)。
ListEvent.ITEM_DOUBLE_CLICK	已按兩下一個清單項。 <ul style="list-style-type: none">• itemRenderer: 按兩下過的清單項。IListItemRenderer 類型。• itemData:與清單項關聯的資料。此值是從清單的dataProvider 中檢索的。ActionScript Object 類型。• index:清單的新的選定索引。int 類型。值 -1(未設置選定的索引)到清單項數量減去 1。• controllerIdx:用來生成該事件的控制器/滑鼠的索引(僅適用於多滑鼠游標環境)。
ListEvent.ITEM_ROLL_OVER	滑鼠游標滑過一個清單項。 <ul style="list-style-type: none">• itemRenderer: 按兩下過的清單項。IListItemRenderer 類型。• itemData:與清單項關聯的資料。此值是從清單

的 `dataProvider` 中檢索的。ActionScript Object 類型。

- `index`: 清單的新的選定索引。int 類型。值 -1(未設置選定的索引)到清單項數量減去 1。
- `controllerIdx`: 用來生成該事件的控制器/滑鼠的索引(僅適用於多滑鼠游標環境)。

ListEvent.ITEM_ROLL_OUT

滑鼠游標滑離一個清單項。

- `itemRenderer`: 按兩下過的清單項。IListItemRenderer 類型。
- `itemData`: 與清單項關聯的資料。此值是從清單的 `dataProvider` 中檢索的。ActionScript Object 類型。
- `index`: 清單的新的選定索引。int 類型。值 -1(未設置選定的索引)到清單項數量減去 1。
- `controllerIdx`: 用來生成該事件的控制器/滑鼠的索引(僅適用於多滑鼠游標環境)。

下例顯示了如何監聽列表項的點擊動作：

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

2.4.5 TileList



圖 41: 無皮膚標題列表 TileList.

標題列表 `TileList` (`scaleform.clik.controls.TileList`) 與滾動列表 `ScrollingList` 類似，為一個可以滾動其元素的元件。可以自身展示列表項或者使用現存的場景中的列表項。可以附加滾動指示器 `ScrollIndicator` 或捲軸 `ScrollBar` 元件提供滾動反饋和控制功能。標題列表 `TileList` 和滾動列表 `ScrollingList` 的區別為標題

列表同時支援多個行和列，列表項選擇可以向四個主要方向移動。該元件可與資料源 **dataProvider** 一起使用。資料源通過代碼指派，如下面實例所示：

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

默認情況下標題列表 **TileList** 使用列表項渲染器 **ListItemRenderer** 元件作為內容。因此列表項渲染器必須在 **FLA** 文件庫中存在，除非列表項渲染器檢查屬性改變成另外一個元件。見檢查屬性小節獲得更多資訊。

2.4.5.1 用戶交互

點擊一個列表項或者一個附屬的捲軸 **ScrollBar** 實例將焦點傳遞到標題列表 **TileList** 元件。當獲得焦點，按下向上向下方向鍵或者類似的控制器，如果包含多行逐個單元垂直滾動列表選項。也可以用向左向右方向鍵或類似控制器，如果包含多列逐個單元水平滾動列表選項。如果標題列表 **TileList** 包含多行和多列，四個方向鍵或類似控制器可以用來導航列表項。如果沒有選中元素，頂部元素自動作為默認選項。如果游標位於標題列表邊界上滑鼠滾輪可以滾動列表。

按下鍵盤(**Page Up**) 和 (**Page down**)或類似控制器將按頁滾動選項，例如，列表中的可視選項數。按下 (**Home**) 和 (**End**)鍵，或者類似控制器，將滾動列表到相應元素的開始和末尾處。與附屬捲軸 **ScrollBar** 元件交互將按預期影響標題列表 **TileList**。見捲軸 **ScrollBar** 小節瞭解其用戶交互功能。

2.4.5.2 元件設置

標題列表 **TileList** 不需要任何子單元，但是，在場景上的標題列表 **TileList** 元件上放置一個元件或者調整元件大小時一個視覺化背景就能起到幫助作用。

2.4.5.3 狀態

標題列表 **TileList** 元件支援三種狀態，基於焦點和禁止屬性：

- **default** 或 **enable** 狀態；
- **focused** 狀態，通常突出顯示元件邊框區域；
- **disabled** 狀態。

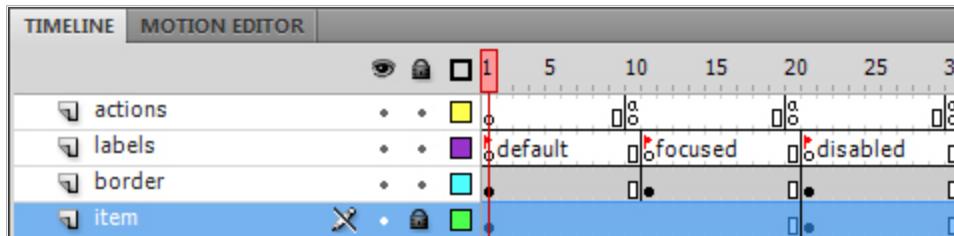
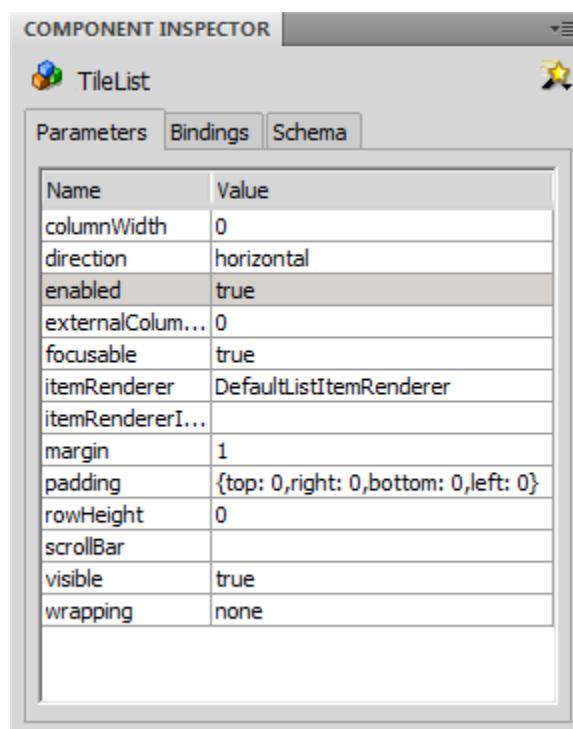


圖 42: 標題列表 **TileList** 時間軸

2.4.5.4 檢查屬性



來自標題列表 **TileList** 元件的視頻剪輯 MovieClip 具有以下檢查屬性：

columnWidth	內部創建的列表項實例寬度，如果設置了 <i>rendererInstanceName</i> 屬性該值不產生影響。
Direction	滾動方向。行和列的語義不會隨此值而變化。
Enabled	如果設置為 <i>false</i> ,就禁用元件。禁用的元件將不再收到使用者輸入。
externalColumnCount	當設置了 <i>rendererInstanceName</i> 屬性，該值用來指示外部渲染器使用的列數的標題列表 TileList
Focusable	啟用/禁用對於元件的焦點管理。將 <i>focusable</i> (可聚焦)屬性設置為 <i>false</i> 將會取消對 <i>tab</i> 鍵、方向鍵以及基於滑鼠的焦點更改的支援。

itemRenderer	<code>ListItemRenderer</code> 的符號名稱。用來在內部創建清單項實例。如果設置了 <code>itemRendererInstanceName</code> 屬性(即如果使用外部 <code>ListItemRenderer</code>), 則沒有影響。
itemRendererInstanceName	要與此 <code>ScrollingList</code> 元件一起使用的外部清單項渲染器的首碼。 <code>Stage</code> 上的清單項實例必須以此屬性值為首碼。如果將此屬性設置為值 ‘r’, 則要與此元件一起使用的所有清單項實例必須具有下列值:‘r1’, ‘r2’, ‘r3’,... 第一項應具有數位 1。
Margin	清單元件邊界與內部創建的清單項之間的空余空間。如果設置了 <code>itemRendererInstanceName</code> 屬性(即如果使用外部 <code>ListItemRenderer</code>), 則此值沒有影響。
Padding	針對清單項的額外填充 (Padding)。如果設置了 <code>itemRendererInstanceName</code> 屬性(即如果使用外部 <code>ListItemRenderer</code>), 則此值沒有效果。填充不會影響自動生成的 <code>ScrollBar</code> (捲軸)。
rowHeight	內部創建的清單項實例的高度。如果設置了 <code>itemRendererInstanceName</code> 屬性(即如果使用外部 <code>ListItemRenderer</code>), 則此值沒有影響。
Visible	如果設置為 <code>false</code> , 就隱藏元件。這不會隱藏附加的 <code>ScrollBar</code> 或任何外部 <code>ListItemRenderer</code> 。

2.4.5.5 事件

所有事件回檔均會收到單個 `Event`(事件)或 `Event` 子類參數, 該參數包含有相應事件的相關資訊。下列屬性是所有事件所共有的。

- **type** : 事件類型。
- **target** : 事件目標。
- **currentTarget** : 通過一個事件偵聽器積極處理 `Event` 物件的物件。
- **eventPhase** : 事件流中的當前階段。 (`EventPhase.CAPTURING_PHASE`、
`EventPhase.AT_TARGET`、`EventPhase.BUBBLING_PHASE`) 。
- **bubbles** : 指明事件是不是起泡事件。
- **cancelable** : 指明與事件關聯的行為是否可以避免

標題列表 `TileList` 元件產生的事件列表如下所示。事件旁列出的屬性為通用屬性的補充。

ComponentEvent.SHOW	<code>visible</code> (可見)屬性已在運行時設置為 <code>true</code> 。
ComponentEvent.HIDE	<code>visible</code> 屬性已在運行時設置為 <code>false</code> 。
ComponentEvent.STATE_CHANGE	元件的狀態已更改。
FocusHandlerEvent.FOCUS_IN	元件已收到焦點。

FocusHandlerEvent.FOCUS_OUT	元件已失去焦點。
ListEvent.INDEX_CHANGE	<p>選定的索引已發生變化。</p> <ul style="list-style-type: none"> • itemRenderer: 按兩下過的清單項。IListItemRenderer 類型。 • itemData: 與清單項關聯的資料。此值是從清單的dataProvider 中檢索的。ActionScript Object 類型。 • index: 清單的新的選定索引。int 類型。值 -1(未設置選定的索引)到清單項數量減去 1。 • controllerIdx: 用來生成該事件的控制器/滑鼠的索引(僅適用於多滑鼠游標環境)。
ListEvent.ITEM_PRESS	<p>已按下一個清單項。</p> <ul style="list-style-type: none"> • itemRenderer: 按兩下過的清單項。IListItemRenderer 類型。 • itemData: 與清單項關聯的資料。此值是從清單的dataProvider 中檢索的。ActionScript Object 類型。 • index: 清單的新的選定索引。int 類型。值 -1(未設置選定的索引)到清單項數量減去 1。 • controllerIdx: 用來生成該事件的控制器/滑鼠的索引(僅適用於多滑鼠游標環境)。
ListEvent.ITEM_CLICK	<p>已點擊一個清單項。</p> <ul style="list-style-type: none"> • itemRenderer: 按兩下過的清單項。IListItemRenderer 類型。 • itemData: 與清單項關聯的資料。此值是從清單的dataProvider 中檢索的。ActionScript Object 類型。 • index: 清單的新的選定索引。int 類型。值 -1(未設置選定的索引)到清單項數量減去 1。 • controllerIdx: 用來生成該事件的控制器/滑鼠的索引(僅適用於多滑鼠游標環境)。
ListEvent.ITEM_DOUBLE_CLICK	<p>已按兩下一個清單項。</p> <ul style="list-style-type: none"> • itemRenderer: 按兩下過的清單項。IListItemRenderer 類型。 • itemData: 與清單項關聯的資料。此值是從清單的

	<p><code>dataProvider</code> 中檢索的。ActionScript Object 類型。</p> <ul style="list-style-type: none"> • <code>index</code>:清單的新的選定索引。int 類型。值 -1(未設置選定的索引)到清單項數量減去 1。 • <code>controllerIdx</code>:用來生成該事件的控制器/滑鼠的索引(僅適用于多滑鼠游標環境)。
ListEvent.ITEM_ROLL_OVER	<p>滑鼠游標滑過一個清單項。</p> <ul style="list-style-type: none"> • <code>itemRenderer</code>: 按兩下過的清單項。IListItemRenderer 類型。 • <code>itemData</code>:與清單項關聯的資料。此值是從清單的 <code>dataProvider</code> 中檢索的。ActionScript Object 類型。 • <code>index</code>:清單的新的選定索引。int 類型。值 -1(未設置選定的索引)到清單項數量減去 1。 • <code>controllerIdx</code>:用來生成該事件的控制器/滑鼠的索引(僅適用于多滑鼠游標環境)。
ListEvent.ITEM_ROLL_OUT	<p>滑鼠游標滑離一個清單項。</p> <ul style="list-style-type: none"> • <code>itemRenderer</code>: 按兩下過的清單項。IListItemRenderer 類型。 • <code>itemData</code>:與清單項關聯的資料。此值是從清單的 <code>dataProvider</code> 中檢索的。ActionScript Object 類型。 • <code>index</code>:清單的新的選定索引。int 類型。值 -1(未設置選定的索引)到清單項數量減去 1。 • <code>controllerIdx</code>:用來生成該事件的控制器/滑鼠的索引(僅適用于多滑鼠游標環境)。

下例顯示如何判斷標題列表 TileList 接收焦點：

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

2.4.6 DropdownMenu

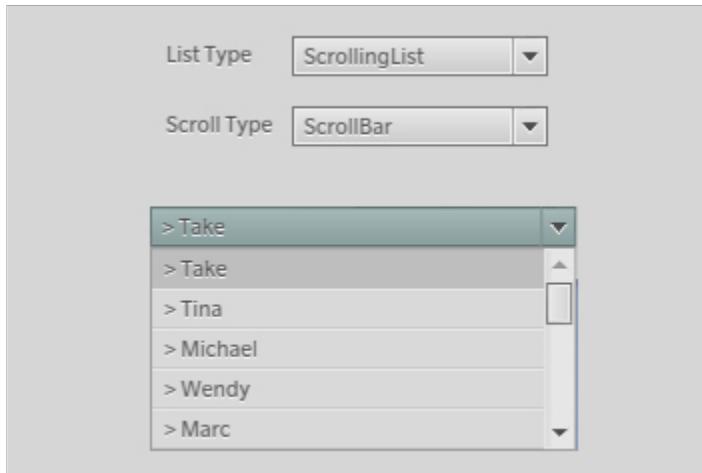


圖 43: 無皮膚下拉功能表 **DropdownMenu**.

下拉功能表 **DropdownMenu** (`scaleform.clik.controls.DropdownMenu`) 包含了按鈕和列表的行爲。點擊元件打開列表包含了選擇元素。下拉功能表 **DropdownMenu** 只在靜止狀態顯示選擇元素。可以配置成使用滾動列表 **ScrollingList** 或標題列表 **TileList**，並可以配上捲軸 **ScrollBar** 或滾動指示器 **ScrollIndicator**。列表與資料源 **dataProvider** 相關聯，下拉功能表 **DropdownMenu** 列表元素與資料源 **dataProvider** 進行連接。資料源 **dataProvider** 通過代碼指派，如下例所示：

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

默認情況下下拉功能表 **DropdownMenu** 使用列表項渲染器 **ListItemRenderer** 元件作為內容。因此下拉功能表 **DropdownMenu** 和列表項渲染器 **ListItemRenderer** 必須在 FLA 文件庫中存在，除非下拉屬性改變成另外一個元件。參考檢查屬性小節獲得更多資訊。

也需注意默認情況下下拉功能表 **DropdownMenu** 在列表元素中不附帶一個捲軸，捲軸 **ScrollBar** 或滾動指示器 **ScrollIndicator** 必須通過代碼附加到下拉功能表 **DropdownMenu** 列表元素。見提示和技巧小節獲得更多資訊。

2.4.6.1 用戶交互

點擊下拉功能表 **DropdownMenu** 實例或者按下(**Enter**)鍵或類似控制器將開打可選元素列表。當打開時焦點轉移到該列表，如在用戶交互小節所描述用戶能夠與列表中 **ScrollingList**、**TileList** 和 **ScrollBar** 元件進行交互。點擊一個列表項將其選中，關閉列表並在下拉功能表 **DropdownMenu** 元件中顯示選擇項。在列表邊界外用滑鼠點擊將自動關閉列表，焦點將傳遞回下拉功能表元件。

2.4.6.2 元件設置

下拉功能表 **DropdownMenu** 繼承了按鈕元件的絕大多數功能。從而視頻剪輯 **MovieClip** 使用下拉功能表類必須有以下名稱的子單元。列表和捲軸動態創建，注出了對應的可選單元：

- **textField**：(可選) **TextField** 類型，按鈕標簽。
- **focusIndicator**: (可選) **MovieClip** 類型，一個單獨的視頻剪輯用來顯示焦點狀態，如果被使用，則視頻剪輯必須擁有兩個幀名為：“**show**”和“**hide**”。

2.4.6.3 狀態

下拉功能表 **DropdownMenu** 開打時為選中狀態，因此需要與 **ToggleButton** 和 **CheckBox** 具有相同的狀態來指示選中狀態，這些狀態包括：

- **up** 或默認狀態；
- 當滑鼠箭頭在元件上方或者獲得焦點時為 **over** 狀態；
- 當按鈕被點擊時候為 **down** 狀態；
- **disabled** 狀態；
- **selected_up** 或者默認狀態；
- 當滑鼠箭頭位於元件上方或獲得焦點時為 **selected_over** 狀態；
- 當按鈕被按下時為 **selected_down** 狀態；
- **selected_disabled** 狀態；

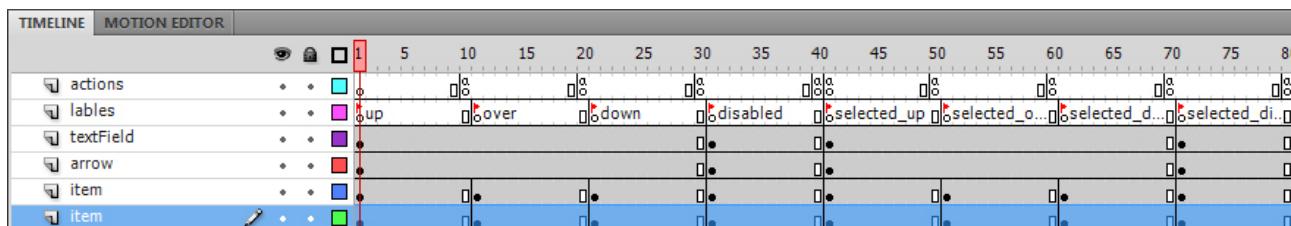
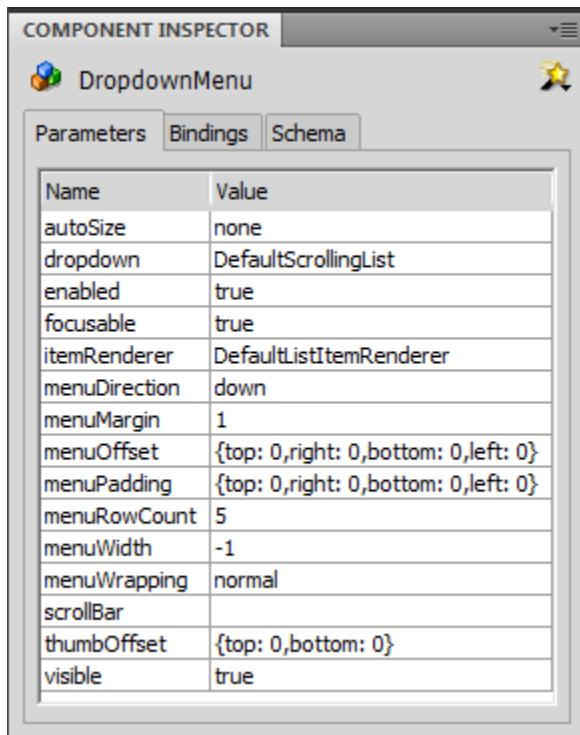


圖 44: 下拉功能表 **DropdownMenu** 時間軸

這裏為選擇按鈕下拉功能表 **DropdownMenu** 所需要的最少關鍵幀設置。按鈕 **Button** 元件支援狀態和關鍵幀的擴展設置，與 **DropdownMenu** 元件相同，在 [CLIK 按鈕入門](#)文檔中有詳細描述。

2.4.6.4 檢查屬性



下拉功能表 DropdownMenu 元件的檢查屬性為：

autoSize	確定已關閉的 DropdownMenu(下拉式功能表)是否會縮放以適應其所包含的文本以及已調整大小的按鈕將向哪個方向對齊。將 autoSize 屬性設置為 autoSize="none" 將會使其當前大小保持不變。
Dropdown	列表元件 (ScrollingList 或 TileList) 的符號名稱，與下拉功能表 DropdownMenu 元件一起使用
Enabled	如果設置為 false ,就禁用元件。
Focusable	預設情況下,元件可以收到用於使用者互動的焦點。將此屬性設置為 false 將會禁用焦點獲取。
menuDirection	下拉的清單將打開的方向。有效值為 "up"(向上)和 "down"(向下)。
menuMargin	列表部件與內部創建的列表項之間的邊距。該邊距還會對自動生成的捲軸產生影響。
menuOffset	下拉清單從下拉按鈕位置的水準和垂直偏移量。正水準值會將清單移動到下拉按鈕水準位置的右邊。正垂直值會將清單移離該按鈕。
menuPadding	針對清單項的頂部、底部、左側和右側的額外填充。不會影響自動生成的捲軸。
menuRowCount	清單應顯示的行數。
menuWidth	If set, this number will be enforced as the width of the drop down's list.

thumbOffset	捲軸拇指頂部和底部偏移量。如果該清單不自動創建一個捲軸實例,則此屬性沒有影響。
scrollbar	下拉清單的捲軸的符號名稱。由下拉清單實例創建。如果值為空,則下拉清單將沒有捲軸。
Visible	如果設置為 <code>false</code> ,就隱藏元件。

2.4.6.5 事件

所有事件回檔均會收到單個 Event(事件)或 Event 子類參數,該參數包含有相應事件的相關資訊。下列屬性是所有事件所共有的。

- **type** : 事件類型。
- **target** : 事件目標。
- **currentTarget** : 通過一個事件偵聽器積極處理 Event 物件的物件。
- **eventPhase** : 事件流中的當前階段。（EventPhase.CAPTURING_PHASE、EventPhase.AT_TARGET、EventPhase.BUBBLING_PHASE）。
- **bubbles** : 指明事件是不是起泡事件。
- **cancelable** : 指明與事件關聯的行為是否可以避免

DropdownMenu 元件產生的事件列表如下所示。除 `change` 事件意外,與 Button 元件類似,事件旁列出的屬性為通用屬性的補充。

ComponentEvent.SHOW	visible(可見)屬性已在運行時設置為 true。
ComponentEvent.HIDE	visible 屬性已在運行時設置為 false。
ComponentEvent.STATE_CHANGE	元件的狀態已更改。
Event.SELECT	選定的屬性已發生變化。
FocusHandlerEvent.FOCUS_IN	元件已收到焦點。
FocusHandlerEvent.FOCUS_OUT	元件已失去焦點。
ListEvent.INDEX_CHANGE	選定的索引已發生變化。 <ul style="list-style-type: none"> • itemRenderer: 按兩下過的清單項。IListItemRenderer 類型。 • itemData: 與清單項關聯的資料。此值是從清單的dataProvider 中檢索的。ActionScript Object 類型。 • index: 清單的新的選定索引。int 類型。值 -1(未設置選定的索引)到清單項數量減去 1。 • controllerIdx: 用來生成該事件的控制器/滑鼠的

	索引(僅適用於多滑鼠游標環境)。
MouseEvent.ROLL_OVER	<p>滑鼠箭頭在按鈕上方滾動。</p> <p>mouseldx:用來生成該事件的滑鼠游標的索引(僅適用於多滑鼠游標環境)。uint 類型。僅限於 Scaleform,需要將該事件歸於 MouseEventEx 。</p> <p>buttonIdx:指明為哪個按鈕生成該事件(基於零的索引)。僅限於 Scaleform,需要將該事件歸於 MouseEventEx 。</p>
MouseEvent.ROLL_OUT	<p>滑鼠箭頭從按鈕移開。</p> <p>mouseldx:用來生成該事件的滑鼠游標的索引(僅適用於多滑鼠游標環境)。uint 類型。僅限於 Scaleform,需要將該事件歸於 MouseEventEx 。</p> <p>buttonIdx:指明為哪個按鈕生成該事件(基於零的索引)。僅限於 Scaleform,需要將該事件歸於 MouseEventEx 。</p>
ButtonEvent.PRESS	<p>按鈕被點擊。</p> <p>controllerIdx:用來生成該事件的控制器的索引(僅適用於多滑鼠游標環境)。uint 類型。</p> <p>isKeyboard:為 true - 假如事件是由電腦鍵盤 (Keyboard)/遊戲鍵盤 (Gamepad) 生成的;為 false – 假如事件是由滑鼠生成的。 isRepeat:為 true – 假如事件是由一個被按住的 autoRepeating 按鈕生成的;為 false – 該按鈕當前不重複。</p>
MouseEvent.DOUBLE_CLICK	<p>按鈕被雙擊。只在 <i>doubleClickEnabled</i> 屬性為 true 時被觸發。</p> <p>mouseldx:用來生成該事件的滑鼠游標的索引(僅適用於多滑鼠游標環境)。uint 類型。僅限於 Scaleform,需要將該事件歸於 MouseEventEx 。</p> <p>buttonIdx:指明為哪個按鈕生成該事件(基於零的索引)。僅限於 Scaleform,需要將該事件歸於 MouseEventEx 。</p>
ButtonEvent.CLICK	<p>按鈕被點擊。</p> <p>controllerIdx:用來生成該事件的控制器的索引(僅適用於多滑鼠游標環境)。uint 類型。</p> <p>isKeyboard:為 true - 假如事件是由電腦鍵盤 (Keyboard)/遊戲鍵盤 (Gamepad) 生成的;為 false – 假如事件是由滑鼠生成的。 isRepeat:為 true – 假如事件是由一個被按住的 autoRepeating 按鈕生成的;為 false – 該按鈕當前不重複。</p>
ButtonEvent.DRAG_OVER	<p>滑鼠箭頭拖動到按鈕上方 (滑鼠左鍵被按下)</p> <p>controllerIdx:用來生成該事件的控制器的索引(僅適用於</p>

	多滑鼠游標環境)。uint 類型。
ButtonEvent.DRAG_OUT	滑鼠箭頭從按鈕拖開 (滑鼠左鍵被按下)。 controllerIdx: 用來生成該事件的控制器的索引(僅適用於多滑鼠游標環境)。uint 類型。
ButtonEvent.RELEASE_OUTSIDE	滑鼠箭頭從按鈕拖開滑鼠左鍵鬆開。 controllerIdx: 用來生成該事件的控制器的索引(僅適用於多滑鼠游標環境)。uint 類型。

2.5 進度類型

進度類型用來顯示狀態或事件或動作的進度。The Scaleform CLIK 框架包含了兩個元件屬於此分類，狀態指示器 StatusIndicator 和進度條 ProgressBar。狀態指示器 StatusIndicator 元件用來顯示事件或動作的狀態。而進度條元件 ProgressBar 與狀態指示器擁有相同的語義，但是包含了一些附加功能，可以監聽其他產生進度事件的元件或動作。

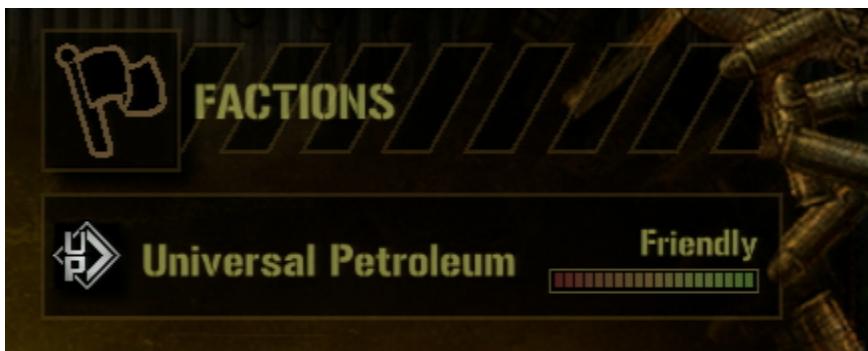


圖 45:來自 *Mercenaries 2* 的 Faction 狀態指示器實例

2.5.1 StatusIndicator

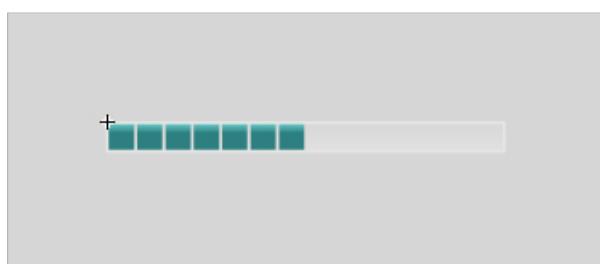


圖 46:無皮膚狀態指示器 StatusIndicator.

狀態指示器 StatusIndicator component (`scaleform.clik.controls.StatusIndicator`)顯示事件或動作狀態，使用時間軸作為視覺指示器。狀態指示器的值將為一個最大和最小值之間的值用來產生一個幀的序號，

以在元件時間軸上播放。由於元件時間軸用來顯示狀態，為創建創新的視覺指示器提供了絕對自由的發揮空間。

2.5.1.1 用戶交互

狀態指示器 StatusIndicator 無用戶交互。

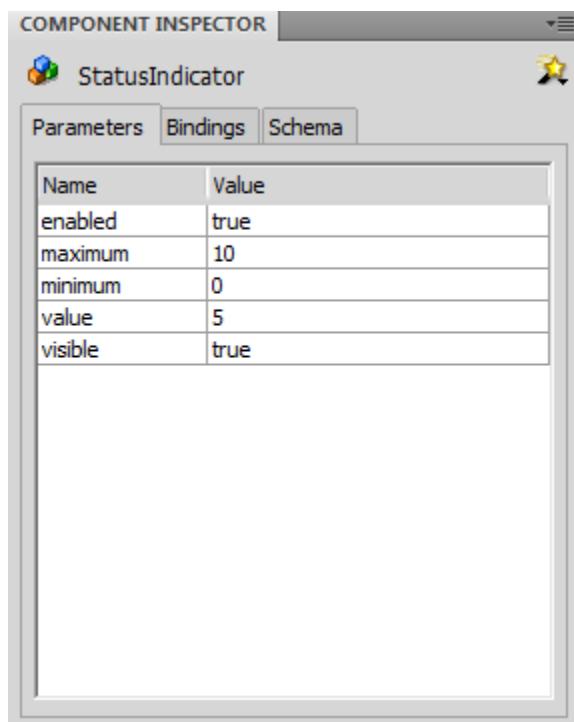
2.5.1.2 元件設置

使用 CLIK 狀態指示器 StatusIndicator 的視頻剪輯 MovieClip 不需要任何子單元。但是狀態指示器需要至少兩個幀以正確操作。確保在第一幀插入 stop()命令避免播放該幀。狀態指示器元件在值屬性產生的對應幀執行 gotoAndStop()命令。

2.5.1.3 狀態

狀態指示器 StatusIndicator 元件無狀態，元件幀用來顯示事件或動作

2.5.1.4 檢查屬性



來自狀態指示器 StatusIndicator 元件的視頻剪輯 MovieClip 具有以下檢查屬性：

Visible 如果設置為 false 則隱藏元件

Enabled 如果設置為 true 則禁止元件.

Value	一個事件或動作的狀態值，為一個最大值到最小值之間的播放的幀序號。
Minimum	插入目標幀的最小值
Maximum	插入目標幀的最大值

2.5.1.5 事件

所有事件回檔均會收到單個 Event(事件)或 Event 子類參數,該參數包含有相應事件的相關資訊。下列屬性是所有事件所共有的。

- **type** : 事件類型。
- **target** : 事件目標。
- **currentTarget** : 通過一個事件偵聽器積極處理 Event 物件的物件。
- **eventPhase** : 事件流中的當前階段。（EventPhase.CAPTURING_PHASE、EventPhase.AT_TARGET、EventPhase.BUBBLING_PHASE）。
- **bubbles** : 指明事件是不是起泡事件。
- **cancelable** : 指明與事件關聯的行為是否可以避免

StatusIndicator 元件產生的事件列表如下所示。事件旁列出的屬性為通用屬性的補充。

ComponentEvent.SHOW	visible 屬性已在運行時設置為 true。
ComponentEvent.HIDE	visible 屬性已在運行時設置為 false。

2.6 其他類型

Scaleform CLIK 框架頁包括若干個元件且不能簡單區分，但是為 UI 開發提供了寶貴的功能。They are the Window and DragSlot components.

Window(視窗)元件允許顯示可用來顯示內容或用作對話方塊的有模態和無模態視窗。

DragSlot 是一個 CLIK 元件的基礎實現,該 CLIK 元件設計為用於需要拖放功能的 UI。拖放是作為 CLIK 內的一個自訂框架實現的,而且必須安裝 CLIK DragManager(請參閱 scaleform.clik.managers.DragManager.as)。

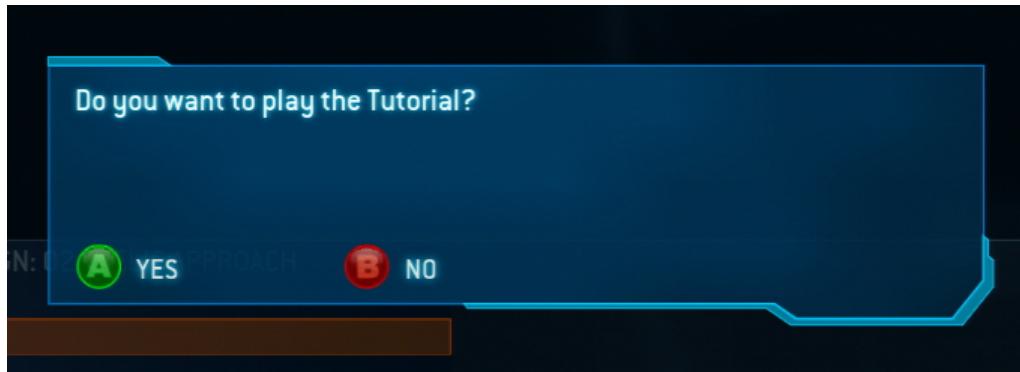


圖 48:來自 *Halo Wars* 的對話方塊 Window 實例

2.6.1 Window

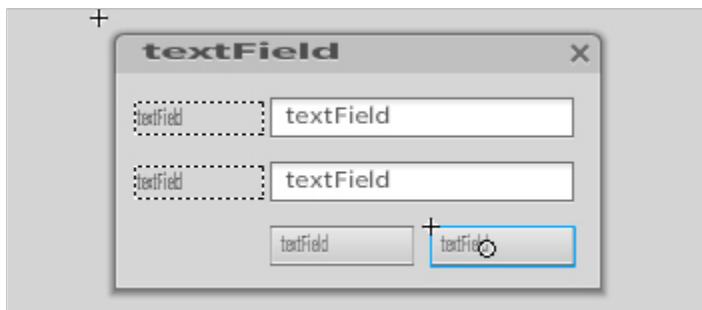


Figure 3: Sample unskinned Window.

CLIK Window 元件 (`scaleform.clik.controls.Window`) 顯示一個視窗或一個對話視圖,如一個 Alert(警告)對話方塊。注意內置元件沒有任何內容，因為設計成完全由用戶定義。但是，通過簡單的編輯元件符號可以添加內容。

PopUpManager 提供如下功能:作為對話方塊打開一個視窗(參閱 `PopUpManager.showModal()`)和同時管理多個對話方塊。

2.6.1.1 用戶交互

對話方塊 Window 的用戶交互在創建的對話方塊視圖裏定義。

2.6.1.2 元件設置

使用 CLIK Window 類的 MovieClip 必須具備下面所列的子單元。也列出了可選元素:

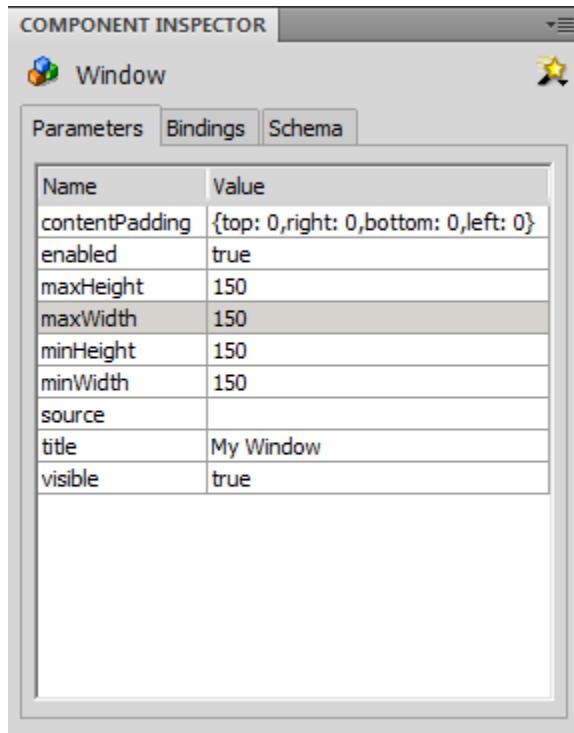
- **closeBtn** : (可選)CLIK Button 類型。一個關閉該視窗的按鈕。
- **titleBtn** : (可選)CLIK Button 類型。一個用於對話方塊的可拖曳標題列。

- **okBtn** : (可選)CLIK Button 類型。一個“Accept”(接受)或“OK”(確定)按鈕,按一下時會導致視窗關閉。
- **resizeBtn**: (可選)CLIK Button 類型。一個按鈕,當按住該按鈕然後拖動時,它使使用者可以重新調整視窗大小。
- **background** : (可選)MovieClip 類型。視窗的背景。如果重調視窗大小,該背景就會被縮放並變化大小。
- **hit** : (可選)MovieClip 類型。視窗的 hitArea。

2.6.1.3 狀態

對話方塊 Window 元件無狀態。視頻剪輯 MovieClip 作為對話方塊視圖顯示,或許有或許沒有自身狀態。

2.6.1.4 檢查屬性



來自對話方塊 Dialog 元件的視頻剪輯 MovieClip 具有以下檢查屬性：

contentPadding	頂部、底部、左側和右側填充,應該應用到載入到視窗中的內容。
Enabled	如果設置為 true 則禁止元件。
maxHeight	通過重新調整大小按鈕調整大小時視窗的最大高度。
maxWidth	通過重新調整大小按鈕調整大小時視窗的最大寬度。
minHeight	通過重新調整大小按鈕調整大小時視窗的最小高度。

minWidth	通過重新調整大小按鈕調整大小時視窗的最小寬度。
Source	應載入到視窗中的符號的匯出名稱。
Title	如果視窗有一個 titleBtn 子元素,此字串就會被用作其標籤。
Visible	如果設置為 false 則隱藏元件

2.6.1.5 事件

所有事件回檔均會收到單個 Event(事件)或 Event 子類參數,該參數包含有相應事件的相關資訊。下列屬性是所有事件所共有的。

- **type** : 事件類型。
- **target** : 事件目標。
- **currentTarget** : 通過一個事件偵聽器積極處理 Event 物件的物件。
- **eventPhase** : 事件流中的當前階段。（EventPhase.CAPTURING_PHASE、EventPhase.AT_TARGET、EventPhase.BUBBLING_PHASE）。
- **bubbles** : 指明事件是不是起泡事件。
- **cancelable** : 指明與事件關聯的行為是否可以避免

Window 元件產生的事件列表如下所示。事件旁列出的屬性為通用屬性的補充。

ComponentEvent.SHOW visible 屬性已在運行時設置為 true。

ComponentEvent.HIDE visible 屬性已在運行時設置為 false。

以下例子顯示如何處理對話方塊提交事件：

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

3 藝術細節

本章將幫助美工設計師開發 Scaleform CLIK 元件的皮膚，包括為小的元件實例繪製皮膚的詳細過程和最優方法，以及動畫和內置字體。

3.1 最優方法

本章包括了為 Scaleform CLIK 元件創建皮膚的最優方法。

3.1.1 圖元圖像

當開發基於 CLIK 元件繪製向量皮膚，建議所有資源都用圖元圖像。一個完美的圖元資源在 Flash 柵格化中完美排列。

為使得能夠改變柵格：

1. 從 Flash 頂部功能表選擇視圖；
2. 選擇柵格，點擊使柵格顯示；
3. 從 Flash 頂部功能表再次選擇視圖；
4. 選擇柵格然後點擊編輯柵格；
5. 在垂直和水平位置輸入 1px；
6. 點擊 OK。

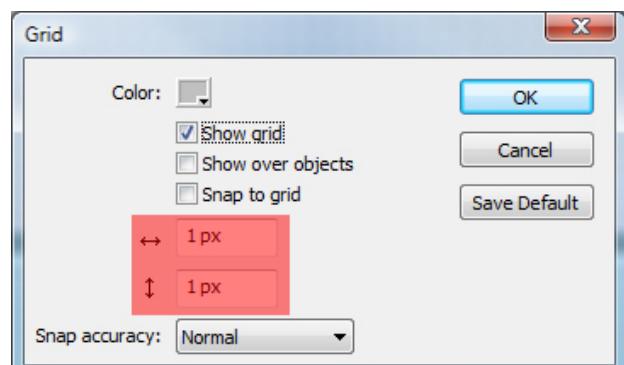


圖 51: 編輯柵格 Grid 視窗

現在應該可以看到覆蓋在場景上的一個柵格。每個柵格表示一個圖元。確保當創建美術資源時對齊柵格。這回確保最終的 SWF 中圖像不會模糊。注意：保持所有圖像尺寸為 2 的指數倍；否則可能會導致模糊，見下面的 2 的指數倍標題。

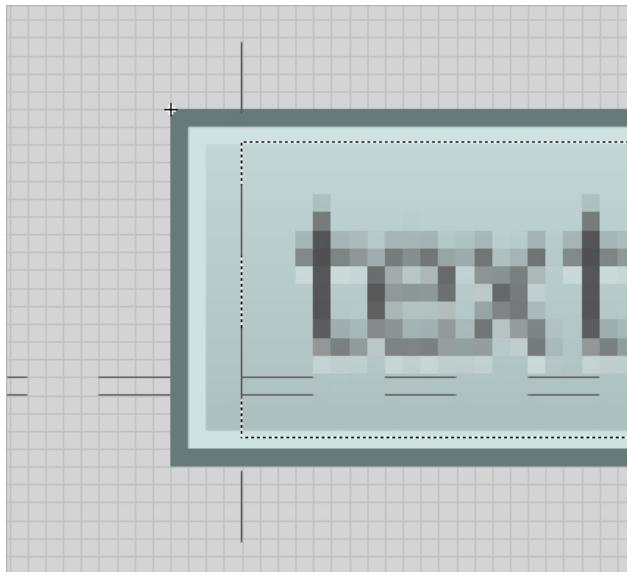


圖 52: 一個圖元的向量圖像

3.1.2 蒙版

在 Flash 中蒙板可以使設計師在運行時隱藏部分圖像。蒙板經常在動畫效果中使用，但是，蒙板消耗大量資源。Scaleform 建議儘量避免蒙板的使用。如果蒙板必須使用，保持在個位數之內並測試添加蒙板前後動畫性能。

在 Flash 中使用蒙板的一個可能選擇為在 Photoshop®中創建 PNG，用透明混合處理需要遮蓋的區域。但是，這只能用於非動畫圖像的透明區域。

3.1.3 動畫

最好避免動畫中向量圖形的轉換，如將一個正方形轉換成圓形。這些動畫類型開銷非常大，因為這些形狀在每一個幀都需要重新進行評估。

如果可以避免在向量圖形中縮放動畫，額外的柵格化影響記憶體和性能 – 將向量圖形轉換為三角元素圖像的過程。開銷最小的動畫為使用平移和旋轉，不需要額外的柵格化。

避免使用可編程映射動畫，選擇以映射動畫為基礎的時間軸選擇，因為時間軸映射動畫在性能上更加有優勢。使用時間軸映射動畫，保持在既定幀速率下實現一個平滑的動畫所需要的最少幀數量。

3.1.4 圖層和繪製元素

在 Flash 中創建皮膚使用盡可能少的圖層。每使用一個圖層至少增加一個繪製元素。繪製元素使用越多，記憶體需求就越大，性能也將受到影響。

3.1.5 複雜皮膚介面

在複雜皮膚中最好使用點陣圖；但是，在簡單皮膚中使用向量圖可以節省更多記憶體並可以在任何解析度下縮放而不失真（模糊）。

3.1.6 2 的指數倍

確保點陣圖在尺寸上為 2 的指數倍，2 的指數倍點陣圖尺寸如下所示：

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

3.2 已知問題和推薦工作流程

本節包括了已知藝術相關 Flash 問題列表和在 Scaleform CLIK 下的推薦工作流程。

3.2.1 複製元件

在 Flash 中複製元件存在幾個問題，複製元件導致原來元件的鏈結資訊和元件定義資訊不拷貝到新的元件中去。同樣的，有些元件沒有這些鏈結資訊將無法工作。本節描述了兩種方法來解決這個問題。

3.2.1.1 複製元件（方法 1）

從外部 FLA 文件複製一個無更改（無皮膚）CLIK 元件，如按鈕，到目標 FLA 文件最快的方如下：

1. 打開 *CLIK_Components.fla* 文件。
2. 從文件的庫中拷貝元件（例如，按鈕）到目標 FLA 文件，在原始檔案庫中點擊並選擇 *Copy*，然後點擊目標 FLA 中的庫並選擇 *Paste*。
3. 點擊目標 FLA 庫中的元件並選擇 *Rename* 對其重新命名，不要與 ‘Button’ 重名。
4. 再次點擊目標 FLA 庫中的元件，選擇 *Properties*。
5. 改變 *Identifier* 區域匹配步驟 3 中選擇元件的新名稱。
6. 點擊庫視窗中的空白區域並選擇 *Paste*，一個新的原始元件拷貝將粘貼進來包括所有的完整的鏈結資訊。

3.2.1.2 複製元件 (方法 2)

當在相同的庫中複製符號（元件），鏈結資訊將不會被拷貝到新的複製符號中去。元件功能執行需要這些資訊，實現的方法為：

1. 點擊 *library* 面板中需要複製的元件並選擇 *Properties*。
2. 在 *Properties* 視窗，雙擊文本（例如 `scaleform.clik.controlsButton`）突出顯示 *Class* 文本區域並按下(CTRL+C)鍵進行拷貝。
3. 按下 *Cancel*。
4. 再次點擊需要拷貝的元件，並選擇 *Duplicate*。
5. 在 *Duplicate Symbol* 視窗中，點擊 *Export* 使 *ActionScript* 核取方塊使能。
6. 點擊 *Class* 區域並按下(CTRL+V)來粘貼鏈結資訊到該文本區域 *textField*。
7. 按下 *OK*。
8. 點擊 *library* 面板中新拷貝的元件並選擇 *Component Definition*。
9. 點擊 *Class textField* 空白區域並按下(CTRL+V)來粘貼鏈結資訊到文本區域 *textField*。
10. 按下 *OK*。

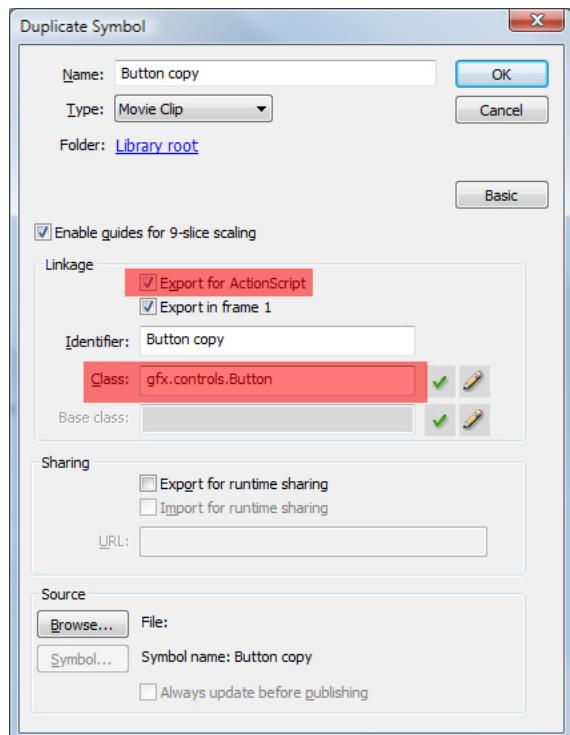


圖 53: 複製符號鏈結（必須填充紅色顯亮區域）

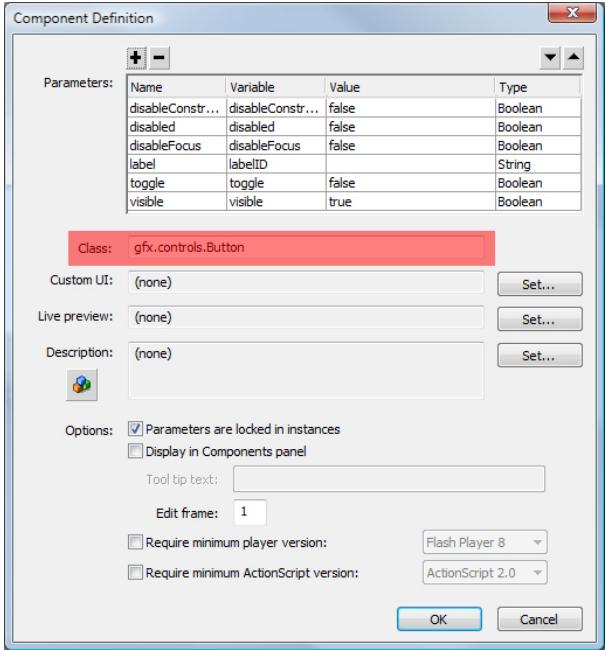


圖 54: 元件定義（必須填入類）

3.3 皮膚繪製實例

Scaleform CLIK 的主要優勢為視覺和函數層的分離。分離能夠使大部分部件中編程人員和藝術設計師可以各自獨立工作。輕鬆自定義外觀和風格為這種分離的主要優勢之一。

儘管內置 CLIK 元件具有一個標準的外觀和風格，但有時候需要自定義以滿足客戶自身需求。以下部分即描述了自定義內置元件的方法。

CLIK 元件設置皮膚與任何 Flash 標準符號相同，雙擊 FLA 文件庫中的一個元件獲得元件任意一個時間軸：

- 在 Flash 中修改默認皮膚；
- 在 Flash 中從頭開始創建自定義皮膚；或者
- 導入 Photoshop 或 Illustrator®中創建的藝術資源到 Flash。

請瀏覽文檔 [CLIK 入門](#)獲得關於皮膚繪製指南的詳細資訊。

3.3.1 StatusIndicator 皮膚繪製

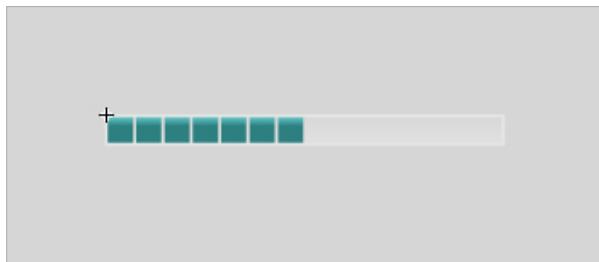


圖 55: 無皮膚 StatusIndicator。

狀態指示器 StatusIndicator 為一個唯一的元件在繪製皮膚時與其他大多數基於按鈕的元件需要略微不同的方法，打開狀態指示器元件，記錄時間軸，具有兩個圖層：*indicator* 和 *track*。*Track* 用來顯示背景圖像；無其他功能。*Indicator* 圖層使用若干關鍵幀包括點陣圖或向量圖來從最低到最高的逐步遞增狀態。

指南中使用 Photoshop 文件中創建的若干個圖層的點陣圖來繪製狀態指示器。這為狀態指示器皮膚繪製的方法之一，但是，還有其他的方法可以在場景中創建和裝配圖形。最終的結果應該都相同，能夠使狀態指示器正確工作。



圖 56: 狀態指示器 StatusIndicator PSD 文件

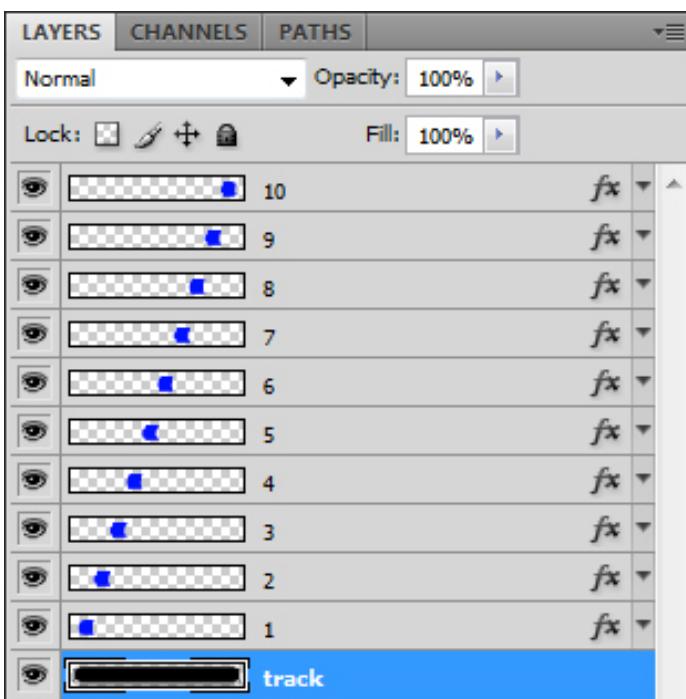


圖 57：在 Photoshop 中為 StatusIndicator PSD 文件設置的圖層

1. 在 Photoshop 中創建狀態指示器 StatusIndicator 點陣圖與上面描述類似。注意圖層順序和編號以及在圖層中每個圖像的位置。確保背景透明，為本使用手冊創建一個類似文件；
2. 保存文件為 PSD 格式；
3. 在 Flash 中，從功能表選擇 *File*，然後選擇 *Import -> Import to Stage*；
4. 瀏覽並選擇狀態指示器 StatusIndicator 皮膚 PSD 文件；
5. 在 *Import* 窗口，確保 *Convert layers to:* 下拉功能表設置為 ‘Layers’ ；
6. 按下 *OK*；
7. 一個新的 Flash 時間軸圖層應該在 PSD 文件中的每個層中創建，在上面圖像例子中，需要創建 10 個圖層，因為 PSD 文件內部擁有 10 個圖層（每個渲染為一個層）。每個圖層標簽分別從 1 到 10，1 為最底部圖層 10 為最頂部圖層，選擇 *layer 1*；
8. 在場景中的點陣圖圖像周圍繪製一個核取方塊；
9. 將圖像移動到老的無皮膚軌道位置，根據要求進行縮放；
10. 在 *layer 1* 上選擇第一個關鍵幀並拖動到第 6 幀；
11. 點擊幀選擇 *Insert* 關鍵幀添加新的幀到 *layer 1* 圖層的第 11 幀；
12. 在 *layer 2* 圖層選擇點陣圖並按下(Ctrl+X)進行剪切；
13. 選擇 *layer 1* 圖層中第 11 幀位置為新關鍵幀並按下(Ctrl+Shift+V)放置點陣圖到 *layer 1* 圖層；
14. 重複此過程直到 10 個點陣圖都在同一個圖層(*layer 1*)，位於正確的關鍵幀。每個並排的點陣圖應該拷貝到最後關鍵幀第 5 幀後的一個關鍵幀中。圖層 *layer 2* 中的點陣圖應該被拷貝到關鍵幀 11；圖層 *layer 3* 中的點陣圖應該被拷貝到第 16 幀；圖層 *layer 4* 中的點陣圖應該被拷貝到第 21 幀，等等。使用 10 個 PSD 圖像，最後關鍵幀應該位於第 51 幀；
15. 刪除空圖層 (2-10) 進行清理；
16. 如果在最後的點陣圖關鍵幀之後時間軸上有附加的幀，加上另外五個關鍵幀，然後選擇剩餘幀並刪除，刪除操作為首先選擇然後右鍵點擊並選擇 *Remove Frames* 即可。在本使用手冊中，最後幀應該位於第 55 幀；
17. 選擇原來的 *indicator layer* 並刪除；
18. 選擇原來的 *track* 圖層並刪除，確保不刪除新導入的 *track* 圖層；
19. 選擇 *layer 1* 並重命名為 ‘indicator’ ；
20. 拖動 *indicator* 圖層和 *track* 圖層到 *actions layer* 圖層下方，確保 *track* 圖層在 *indicator* 圖層下面；

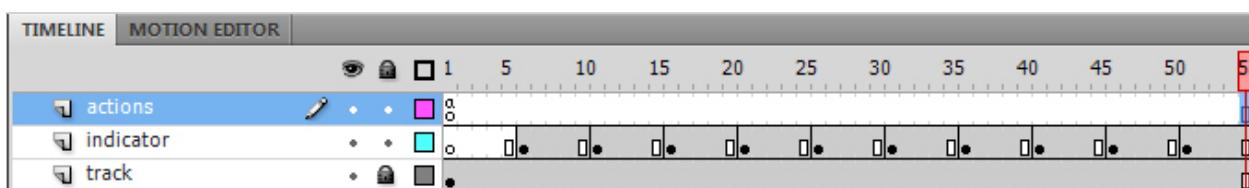


圖 58: 最終時間軸（注意關鍵幀位置和最終幀位置）

21. 推出狀態指示器 StatusIndicator 時間軸；

22. 設置檢查參數值為 1 到 10 之間的任何值；
23. 保存文件；
24. 發佈文件。查看新的皮膚的指示器。

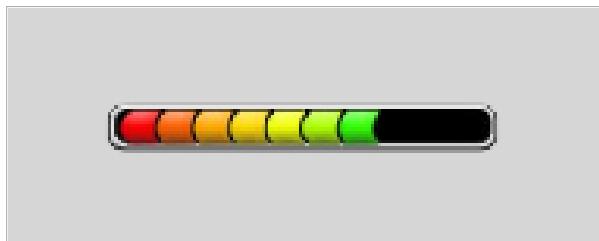


圖 59: StatusIndicator 皮膚繪製

3.4 字體和本地化

本節詳細描述了在 Scaleform CLIK 元件中字體使用。

3.4.1 概要

為了在 Scaleform 中字體能被正確渲染，所需字形（字元）必須內置到 SWF 或使用 Scaleform 本地化系統進行設置。以下為探究在 Flash UI 介面中管理字體的方法。

3.4.2 內置字體

和 Flash 播放器不同，Scaleform 需要內置字體以便顯示。Scaleform 播放器在未嵌入字體情況下將顯示空的進行矩形，甚至這些字體在系統中已存在也是如此。這個效果的優勢為方便判斷在 Scaleform 中字體是否正確嵌入。在內置元件設置中，在每個文本區域 `textField` 實例中都嵌入了 **Slate Mobile**。注意 **Slate Mobile** 不包含任何中文、日文或韓文（CJK）字元或字形，內置元件只包含了 ASCII 字形。這可以通過將 **Slate Mobile** 替換為包含 CJK 字形並設置適當的嵌入選項進行改變。

注意直接內置字體到文本區域 `textFields` 與 Scaleform 本地化系統不相容（在 3.4.4 小節有所描述）。Scaleform 實際上推薦使用 Scaleform 本地化系統來設置字體，比直接嵌入具有很多優勢。但是在某些情況下，如不需要本地化，內置字體最好為可選項。與 UI 介面管理類似，字體管理也需要幾點考慮如本地化和記憶體管理。

3.4.3 在 **textField** 嵌入字體

只需在動態或輸入文本區域 **textFields** 中嵌入字體，靜態文本在編譯時自動轉換為字形輪廓（原始向量圖形）。在動態文本區域 **textField** 中嵌入字體，在場景中選擇動態文本區域的屬性檢查(**Window > Property Inspector**)框中點擊 **Embed** 按鈕。選擇應該嵌入的字元或字元集並選擇 **OK**。一旦完成了這些步驟，在你的 **FLA** 文件中就可以使用該字體。在相同的 **FLA** 文件的多行文本區域中如果需要使用相同的字體，這些步驟只需要執行一次。同樣，多次嵌入相同字體不會增加記憶體使用量。注意字元集包含了大量的字形如中文在導入時佔用大量的記憶體。

3.4.4 本地化系統

Scaleform 本地化系統使用熱交換機制無論在當前位置是否發生變化均可導入和導出字體庫。這些字體庫本身為 **SWF** 文件包括內置字體字形可以被 **SWF** 文本使用。**Scaleform** 能夠使用來自字體庫的字形，為根據需求動態改變字體提供了強大的方法。

由於 **Scaleform** 本地化系統在文本區域 **textField** 直接嵌入字形時不能交換字體，文本區域必須使用一個導入的字形符號。通常字體符號在一個名為 **gxfontlib.fla** 的文件中創建並導入到 **swf** 文本中。這個導入的字體設置到所有支援字體交換的文本區域中。**Scaleform** 當前可以截留該字體符號鏈結並基於當前位置導入不同的字體。

字體庫不需要導出任何字體，只需要插入適當的字體（見前面小節的如何插入字體）。**Scaleform** 本地化系統使用 **fontconfig.txt** 文件來定義每個位置的字體庫，以及文本區域使用的字體符號之間的字體映射關係和插入到字體庫中的實體字體。

fontconfig.txt 文件也包含轉換對應關係。**Scaleform** 本地化系統可以自行文本的快速轉換，這些文本顯示在每個場景中的動態或輸入文本區域。例如，如果一個文本區域包含文本 ‘\$TITLE’，轉換映射具有一個對照表如 \$TITLE=Scaleform，然後文本區域將自動顯示 ‘Scaleform’來代替。

4 編程詳述

本章描述了子系統框架和亮點的具體細節，提供了 Scaleform CLIK 元件架構的上層理解。

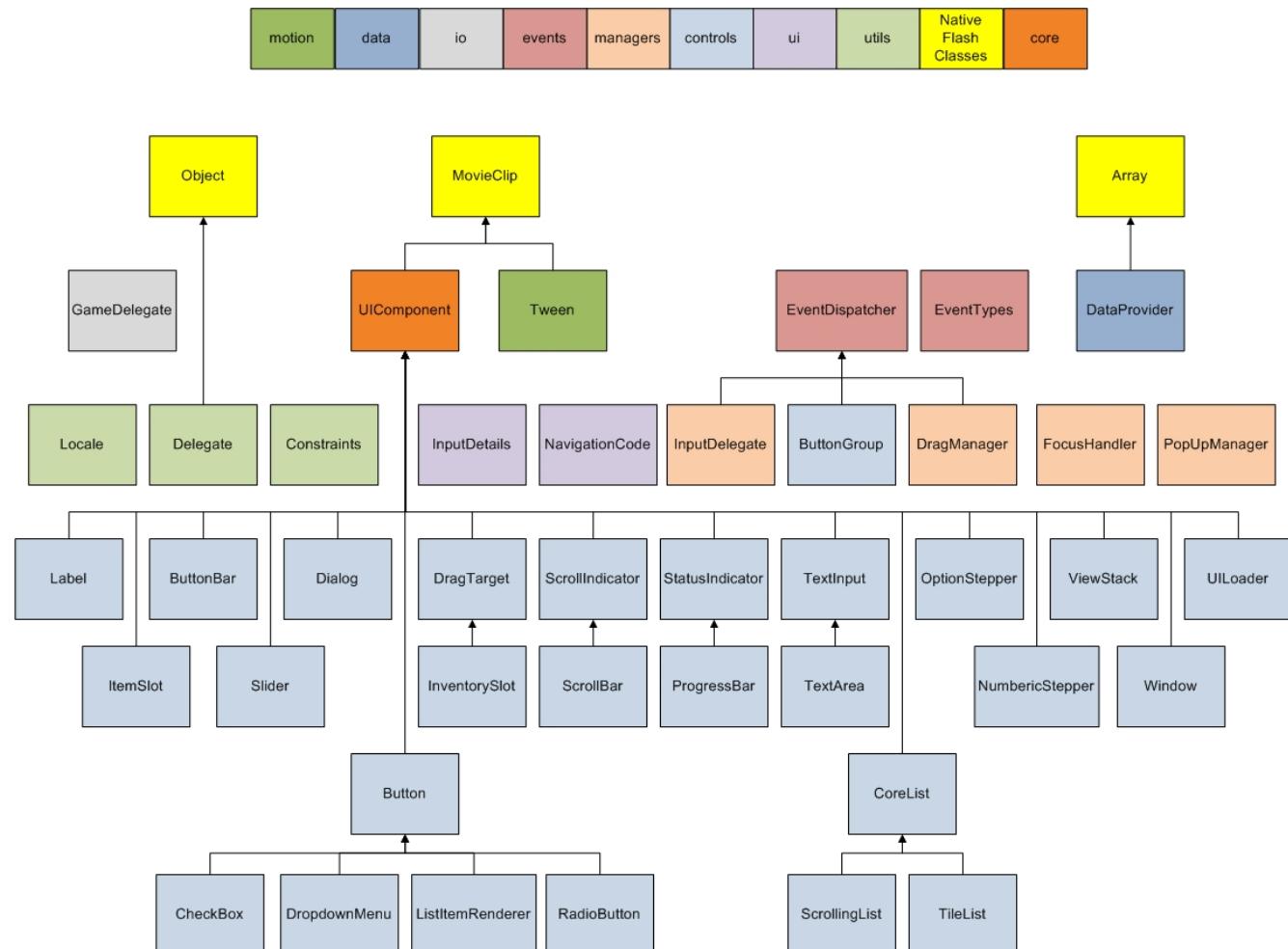


圖 60: Scaleform CLIK 類層次結構

4.1 UI 元件 *UIComponent*

內置元件章描述了與 Scaleform CLIK 捆綁的元件使用的類。所有這些元件都繼承了 **UIComponent** 類 (`scaleform.clik.core.UIComponent`) 的核心功能。這些類為所有 CLIK 元件的基礎，Scaleform 推薦自定義元件為 **UIComponent** 的子類。

UIComponent 類本身從 Flash 8 視頻剪輯 **MovieClip** 類繼承而來，因此繼承了標準 Flash 視頻剪輯 **MovieClip** 的所有屬性和方法。一些自定義讀/寫屬性也被 **UIComponent** 類所支援。

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

- ***focused;***
- ***focusable;***
- ***width;***
- ***height;***
- ***scaleX;***
- ***scaleY;***
- ***displayFocus:*** 如果元件應該顯示焦點狀態則設置為 true，更多資訊請參考[焦點處理](#) 小節。
- ***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 擁有幾個需要子類實現的空方法。這些方法包括：

- ***configUI:*** 元件配置調用；
- ***draw:*** 但元件無效時候調用，更多資訊，請參考[失效](#) 小節；
- ***changeFocus:*** 當元件接收或失去焦點時被調用；
- ***scrollWheel:*** 當滑鼠游標在元件上方滾動滾輪時被調用。

UIComponent 混合到 EventDispatcher 類用來支援事件預訂和指派，因此，所有的子類都支援事件預訂和指派。更多資訊，請參考[事件模型](#) 小節。

4.1.1 初始化

改類在 `onLoad()` 事件控制碼內執行下列初始化步驟：

- 設置 MovieClip 默認尺寸大小；
- 執行元件配置，改步驟調用 `configUI()` 方法；

繪製內容，該步驟調用 `validateNow()` 方法，立即執行畫面刷新，例如調用 `draw()` 函數。

4.2 元件狀態

幾乎所有的 Scaleform CLIK 元件支援視覺狀態，狀態通過元件時間軸上的一個特殊關鍵幀的導航來設置，或者傳遞狀態資訊到子單元。具有三個常用的狀態設置，詳細內容見下面內容。

4.2.1 按鈕元件

任何行為類似按鈕的元件，回應滑鼠動作，可以被選到此分類。例如使用此類的 CLIK 元件為 Button 及其變數、ListItemRenderer、RadioButton 和 CheckBox。複雜元件的子元素如捲軸 ScrollBar 也屬於按鈕 Button 元件。歸到此類的 CLIK 元件可以直接使用按鈕 Button 類(`scaleform.clik.controls.Button`)或者使用從按鈕 Button 類繼承而來的類。

按鈕元件支援的基本狀態為：

- **up** 或默認狀態；
- 當滑鼠箭頭在元件上方或者獲得焦點時為 **over** 狀態；
- 當按鈕被點擊時候為 **down** 狀態；
- **disabled** 狀態；

按鈕 **Button** 元件也支援字首狀態，可以根據其他屬性的值進行設置。默認情況下，核心 **CLIK** 按鈕 **Button** 元件只支援一個 “**selected_**”字首，但元件處於選中狀態時追加到幀的標簽。

按鈕元件支援的基本狀態，包括選中狀態，為以下所示：

- **up** 或默認狀態；
- 當滑鼠箭頭在元件上方或者獲得焦點時為 **over** 狀態；
- 當按鈕被點擊時候為 **down** 狀態；
- **disabled** 狀態；
- **selected_up** 或者默認狀態；
- 當滑鼠箭頭位於元件上方或獲得焦點時為 **selected_over** 狀態；
- 當按鈕被按下時為 **selected_down** 狀態；
- **selected_disabled** 狀態；

注釋：本節中涉及的狀態只是 **CLIK** 按鈕元件支援的狀態類型裏的一部分。詳細的狀態列表請參考 [CLIK 按鈕入門](#) 文檔。

CLIK 按鈕類提供了一個 **getStatePrefixes()**方法，使開發者能夠根據元件屬性改變列表字首。該方法定義如下：

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

如之前提到的，**CLIK** 按鈕默認情況下只支援 “**selected_**”字首。**getStatePrefixes()**方法根據選擇屬性返回不同字首序列。該字首序列將於適當的狀態標簽協同使用來確定幀的播放。

當狀態在內部進行設置，例如滑鼠滾動，出現一個幀標簽列表的查找表。按鈕類中的 **stateMap** 屬性定義了幀標簽映射的狀態。以下為 **CLIK** 按鈕類中定義的狀態映射關係：

```
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" ]
    }
}

```

每個狀態可能有多個目標標簽，從狀態表返回的值與 `getStatePrefixes()` 返回的字首結合產生一個播放的目標幀列表。下圖描述了用來決定正確幀播放的完整過程：

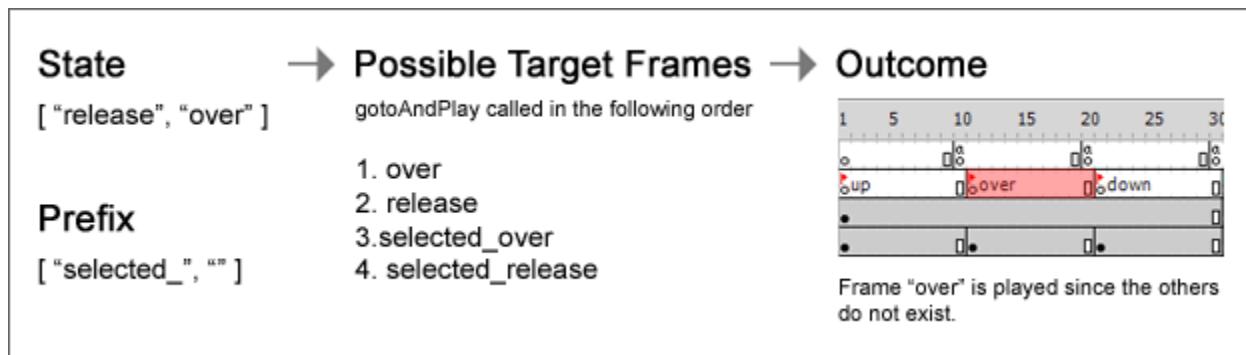


圖 61:決定正確幀播放的過程

如果不能找到特定字首的幀，播放位置總是跳轉到最後的幀，元件為之前請求幀的默認狀態。開發者可以忽略狀態映射而創建自定義行為。

4.2.2 非按鈕交互元件

這些涉及任何元件具有交互和接收焦點功能，但不回應滑鼠事件。例如使用這類方法的 **CLIK** 元件為 **ScrollingList**、**OptionStepper**、**Slider** 和 **TextArea**。這些元件可能包含子單元可以回應滑鼠事件。無按鈕交互元件支援的狀態為：

- **default** 狀態；
- **focused** 狀態；
- **disabled** 狀態。

4.2.3 非交互元件

這些指向任何元件為非交互性質，但是可以被禁止。標簽元件為狀態所支援的默認設置元件中唯一的非交互元件。狀態支援的非交互元件為：

- **default** 狀態；
- **disabled** 狀態。

4.2.4 特殊案例

有幾個元件不遵守上面描述的狀態規則，狀態指示器 **StatusIndicator** 及其子類進度條 **ProgressBar** 使用時間軸顯示元件值。播放位置將設置到幀的位置表示元件值的比例。例如，狀態指示器中的一個 50 幀時間軸最小值為 0，最大值為 10，當前值設置為 5 (50%) 將 **gotoAndStop()** 到第 25 幀 (50 幀的 50% 處)。擴展這些元件管理這些顯示程式非常容易。**updateValue()** 方法可以被修改或忽略以改變其行為。

在某些情況下，某些元件除默認行為外還具有特殊模式，支援額外元件狀態。“文本輸入”和“文本區域”元件在設置了“動作按鈕”的情況下，能夠支援“滾動”和“滑出”狀態，以支援滑鼠滾動和滑出事件。

4.3 事件模型

Scaleform CLIK 元件框架使用一個通信範例稱之為事件模型 **event model**。元件在改變或交互時“分派”事件，容器元件可以訪問不同的事件。這就可以向多個物件通知更改。ActionScript 3 實際上包括一個由 CLIK 利用的強大的事件系統。

有關 ActionScript 3 事件系統的詳細資訊，請訪

問：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 預訂一個事件

預訂一個事件，可以使用 **addEventListener()** 方法，包括一個類型參數指定監聽交互事件的類型。反過來使用 **removeEventListener()** 方法將取消事件預訂。如果多個監聽器包含相同的參數，只能觸發一個。上述方法的每個方法還需要類型函數的一個偵聽器參數，該類型函數與一個以前定義的函數或函數變數相關聯。

CLIK 使用多種自訂事件，每個事件都有其自己的獨特屬性，這些屬性用來提供有關此事件的更詳細的資訊。從 **Event(事件)** 類派生的所有標準 CLIK 事件均可在 **scaleform.clik.events** 中找到。

請注意，**addEventListener()** 還接受另外三個參數，並為其提供預設值：**useCapture**、**priority** 和 **useWeakReference**。一般說來，使用者將會希望所有其事件偵聽器使用弱引用，以避免維護對物件的強引

用,如不維護,這些物件就會成為收集來的垃圾。要確保您的偵聽器將會使用一個弱引用,您的 `addEventListener()` 語法應如下所示：

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

有關 `addEventListener()` 的參數的更多資訊,請訪問：

[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,此類正在創建一個偵聽器,該偵聽器針對從 `buttonInstance` 發出的 `ButtonEvent.CLICK` 事件進行偵聽。無論何時收到此事件,都將會執行 `onButtonClick` 函數。然後此偵聽器函數取消事件偵聽器。

事件的起因和類型不同,針對該偵聽器的 `Event`(事件)參數的屬性有所不同。`CLIK` 元件產生的詳細事件物件(及其屬性)列表,請參考[內置元件](#)相關章節。

4.3.1.2 指派一個事件

`CLIK` 元件希望使用 `dispatchEvent()` 方法通知預訂監聽器發生的變化或交互動作。該方法需要一個參數：一個包含相關資料的物件,包括一個強制類型屬性指定指派事件類型。元件的框架自動添加一個目標屬性,作為事件指派物件的索引,但是能夠手動設置為用自定義目標進行替換。

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

4.4 運行時創建元件

在 ActionScript 3 中,要在運行時動態創建一個元件,您應使用以下語法：

```
import flash.utils.getDefinitionByName;  
import scaleform.clik.controls.Button;  
  
public var myButton:Button;  
  
// 檢索對 Class (類) 定義的引用。
```

```

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 庫中存在一個符號,其 Class(類)設置為 “ButtonLinkageID”,而其 Base Class(基類)設置為 scaleform.clik.controls.Button 。

如果 .FLA 的庫中存在一個有類但無基類的符號,只需通過使用該類的構造函數,就可以創建該符號的一個實例,因為該符號和該類會被聯接在一起。

4.5 焦點處理

Scaleform CLIK 元件使用一個自定義焦點處理框架,在多數元件中執行,並且應該在無框架元件和符號中正常工作。一創建單個元件類,就會產生實體 CLIK FocusHandler 管理器 (scaleform.clik.managers.FocusHandler)。沒有必要直接產生實體 FocusHandler 。

所有焦點變化都是在 Scaleform 播放機層發生的,要麼通過滑鼠或鍵盤(遊戲鍵盤)焦點變化,要麼通過 ActionScript 變化。設置焦點的 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; // 只要帶有此逻辑的 MovieClip 在 Stage 上。
- FocusHandler.getInstance().setFocus(myMovieClip); // FocusHandler 是一个静态类。

4.5.1 最佳使用方法

焦點必須在一個 InteractiveDisplayObject 上設置,否則焦點就會預設到播放機。如果未進行設置,焦點管理系統將不能正常工作(如 Tab 鍵不切換焦點等)。可以通過在任何元件上設置焦點屬性為 true 使得焦點得以應用。另外一個應用焦點的方法,也是應用在非元件元素中,如下所示：

將 tabEnabled 和 mouseEnabled 設置為 false 的 MovieClips 不能通過 Scaleform 或 Flash 播放機聚焦,而且不會生成焦點更改事件。預設情況下,多數 CLIK 元件為其本身及其子元件設置 tabEnabled 和 mouseEnabled 屬性。

某些情況下,設計師可能希望某些元件同時具有 tabEnabled 和 mouseEnabled 屬性,但卻無法收到焦點;UIComponent.focusable 屬性解決了這一問題,因為 AS3 中沒有 MovieClip.focusEnabled 。如果將此屬性設置為 false,UIComponent 就無法通過 CLIK 焦點框架收到焦點。請注意,更改 focusable 屬性將會在內部影響該元件的 tabEnabled 和 mouseEnabled 屬性,因而無法維護以前應用到該元件的 tabEnabled 和 mouseEnabled 配置。

具有可獲得焦點的子元素元件，如捲軸 ScrollBar，使用焦點目標屬性傳遞焦點到它們自身元件。當元件焦點發生變化時，焦點控制碼 FocusHandler 將遞迴搜索焦點目標鏈，將焦點傳給上一個焦點不返回一個焦點目標。

有時當一個元件不是播放器或引用程式的實際焦點時需要出現一個焦點。`displayFocus` 屬性可以設置為 `true` 使元件表現為獲得焦點的樣子。例如，當滑動條 Slider 獲得焦點，滑動條軌道也需要獲得焦點。注意當焦點屬性發生改變時元件調用 `UIComponent.changeFocus()` 方法。

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

反過來，有時一個元件需要是可點擊的，但它卻不該通過 tab 接受焦點，例如，一個可拖曳的面板，或者將會受益于只滑鼠控制的任何其它元件。在此情況下，使用者可以將 `tabEnabled` 屬性設置為 `false`。

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

4.5.2 在複合附件捕獲焦點

複合元件為那些由其他元件組成，如 `ScrollingList`、`OptionStepper` 或 `ButtonBar`。元件本身可能擁有子元件的滑鼠控制碼，但是不具有自身滑鼠控制碼。意思為在 Flash 和 Scaleform 中的 Selection 引擎不支援項和內置導航焦點將無法識別元件，而錯誤得去檢查其子元件。

使元件行為不受合成的約束作為一個獨立的入口，需要以下步驟：

1. 在所有具有滑鼠控制碼的子元件上設置 `tabEnabled = mouseEnabled = focusable = false`（如 `OptionStepper` 中的箭頭按鈕）；
2. 在所有具有滑鼠控制碼的子元件上設置焦點目標屬性。確保在容器元件中設置 `focusable = true`；
3. 如果有必要設置子元件中的 `displayFocus` 屬性為 `true`，該屬性位於容器元件 `changeFocus` 方法之內；

現在當子元件獲得焦點，焦點將轉遞給容器元件。

4.6 輸入處理

由於 Scaleform CLIK 元件從 Flash 8 MovieClip 類繼承而來，當接收到用戶輸入時與任何其他 MovieClip 實例具有相同的行為。如果安裝了滑鼠控制碼元件可以捕獲滑鼠事件。然而，在 CLIK 上相關鍵盤或類似空孩子氣事件處理具有概念上的區別。

4.6.1 最佳使用方法

所有的非滑鼠輸入都可以被 `InputDelegate` 管理類 class (`scaleform.clik.managers.InputDelegate`)截取，`InputDelegate` 將輸入命令轉換為內部的任意一個 `InputDetails` 物件(`scaleform.clik.ui.InputDetails`)，或者從遊戲引擎請求輸入命令的值。後者包括了修改 `InputDelegate.readInput()`方法以支援目標應用程式。一旦 `InputDetails` 被創建，一個輸入事件就從 `InputDelegate` 得到指派。

`InputDetails` 包含了以下屬性：

- 類型，如 “key”
- 編碼，如按下鍵的鍵值
- 值，按鈕向量等輸入資訊，或者鍵輸入類型。這是一個數值，它提供了諸如按鈕向量或按鍵類型等輸入的附加資訊。關鍵事件是由鍵的按下與彈起動作而產生的，相應的輸入細節的數值參數也分別為 “`InputValue.KEY_UP (“keyUp”)` 或 `InputValue.KEY_DOWN (“keyDown”)`”；
- `navEquivalent(navigation equivalent)`，定義了可讀的導航方向如“`as NavigationCode.UP (“up”)`, `NavigationCode.DOWN (“down”)`, `NavigationCode.LEFT (“left”)`, 或者 `NavigationCode.RIGHT (“right”)`，只要可以進行映射。`NavigationCode` 類 (`scaleform.clik.ui.NavigationCode`)提供了一個手動通用導航枚舉列表。

`FocusHandler` 對從 `InputDelegate` 發出的 `InputEvent` 進行偵聽,然後從當前聚焦的元件重新發出同一個 `InputEvent`,以便由該元件或該事件的 `bubble` 鏈中的另一個元件進行處理。

獲得焦點的元件用來決定焦點路徑，為一個自上而下的元件列表，位於實現 `the handleInput()`方法的現實列表層次當中。

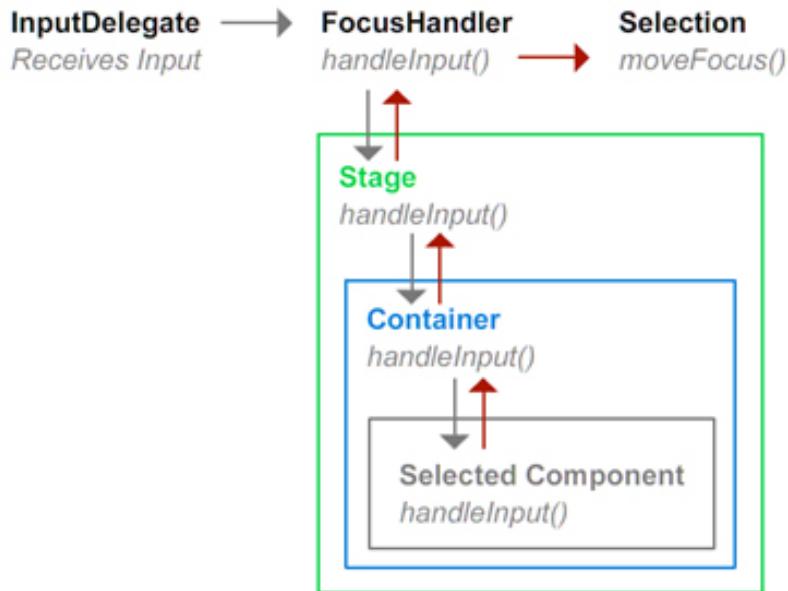


圖 62: handleInput() 函數鏈

當 InputEvent 完成處理後,handled 屬性應設置為 true。這使接收該事件的其它元件可以檢查 event.isDefaultPrevented(),以識別該事件是否已被處理。

```

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 將由 ActionScript 3 事件系統自動發泡 (bubble),假定其 .bubble 屬性未被修改。在某些情況下,如 ScrollingList,元件可能需要將 InputEvent 傳遞給其子元件,然後好嘗試處理該事件本身。例如,假如按了 Enter 鍵,ListItemRenderer 應生成一個 ButtonEvent.CLICK 事件,並且發出該事件的清單將不會做任何事情。

```

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; }
    }
    . .
}

```

也可以添加一個事件監聽器到 `InputDelegate.instance` 中手動監聽輸入事件，並按照這種方法處理輸入資訊。注意這種情況下輸入資訊仍然可以被 `FocusHandler` 控制碼捕獲並傳遞到焦點層次。

```

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

```

`InputDelegate` 應該在遊戲中用來管理預期輸入資訊。默認 `InputDelegate` 處理鍵盤控制、方向鍵轉換以及通用遊戲導航鍵 W、A、S 和 D 直接轉換到相等的導航鍵。

4.6.2 多滑鼠游標

在一些系統中，如 `Nintendo Wii™`，支援多游標設備。`CLIK` 元件框架支援多個游標，但是不允許在單個 `SWF` 文件中多個元件獲得焦點。如果兩個用戶都點擊獨立的按鈕，最後的點擊項即為焦點項。

所有在框架中指派的滑鼠事件包含了一個滑鼠索引屬性，作為產生事件的輸入設備索引。

```

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

```

4.7 失效

失效為元件使用的一種機制，用來限制元件在多個屬性發生變化時候元件的刷新次數。當一個元件為失效狀態，在下一幀將重繪元件。這使開發者可以暫時忽略元件發生的改變，元件只要在特定時間更新一次即可。在有些情況下元件需要立即更新；但是大多數情況下失效性非常有用。

```

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

```

圖 63: CLIK 元件當特定屬性改變時自動變為無效

4.7.1 最佳使用方法

在任何內部元件發生改變後（通常由設置功能引起），則調用 `UIComponent.invalidate()`函數，該函數最終調用 `UIComponent.draw()`函數重繪元件。`invalidate()`方法基於計時器延時產生 `draw()`調用避免不必要的更新。開發者使用的元件可能不需要失效屬相，但是至少也需要瞭解這項功能。

為了實現最佳性能，`draw()` 內的邏輯被分為若干小節，它們由 `isInvalid()` 檢驗包裝，這些檢驗識別該元件的哪些方面（例如，資料、大小、狀態等）當前無法避免更新該元件尚未更改的部分。調用 `invalidate()` 時，該方法使該元件的各個方面無效，迫使在 `draw()` 內進行一次完整重繪。因此，為了實現最佳性能，子類應通過在 `isInvalid()` 檢驗內包裝 `draw()` 邏輯來遵循同一范式，而且並非調用 `invalidate()`，而是使用專用的 `invalidate()` 方法（如 `invalidateData()`, `invalidateSize()`）來避免不必要的重繪。

`invalidate()` 方法在下一階段無效 (Event.RENDER) 或下一幀 (Event.ENTER_FRAME) 上生成 `draw()` 調用，以便於該元件將多項更改批次處理到單個 `draw()` 調用之中。

在需要立即重繪元件的情況下，開發者可以使用 `UIComponent.validateNow()`方法函數。

請注意，如果該元件中以前什麼也沒有標為無效，就不會調用 `draw()`。

4.8 元件縮放

Scaleform CLIK 元件按照兩種方式進行縮放：

1. 使用重複繪製，重新安排元件尺寸，元件元素進行縮放適合原來的比例尺寸。具有子黨員並沒有背景的元件採用此方法。
2. 元素不進行縮放維持縱橫比例關係，元件進行縮放，但是元素不縮放。該方法是用在有圖形背景的元件中，`scale9grid` 進行了伸展，內部的文本縮放而不扭曲。

由於受到 Flash scale9Grid 的限制，元件通常使用回流方法。這要求開發者建立“皮膚”標識，類似於 Flash/Flex 元件。如果部件擁有可能放縮的子元素，則回流方法的效果最好，因此它應用於 container 和類似實體。

反縮放方法專門為 CLIK 創建，主要是由於 Scaleform 中的 scale9Grid 擴展功能。它允許利用幀狀態創建一個單一資產元件，而無需使用設置其他元件時所用的密集分層方法。基本的 CLIK 元件都具有簡約的特性，通常包括一個背景、一個標籤和一個可選圖示或子按鈕，因此非常適合反縮放方法。然而，反縮放並不打算進行類 container 設置（面板設計等等）。在這種情況下，我們建議使用回流方法，因為它具有約束其餘子元素的背景。

元件可以在 Flash IDE 的場景中進行縮放，或者使用寬度和高度屬性進行動態縮放，或者使用 setSize() 方法函數。縮放後的元件外觀可能在 Flash IDE 中不是那麼精確。這就是其局限性，元件作為未編譯的視頻剪輯 MovieClips 不能進行即時預覽 LivePreviews。在 Scaleform Player 中測試動畫為確認縮放元件在遊戲中外觀是否精確的唯一方法。CLIK 擴展了一個啓動面板改進了這項工作流。

4.8.1 Scale9Grid

多數元件使用第二種縮放方法。Scaleform Player 中的 Scale9Grid 視頻剪輯 MovieClip 資源在多數部分都附帶了 scale9grid，儘管 Flash 在包含了 MovieClips 的情況下將丟棄柵格。這意味著即使 scale9grid 在 Flash Player 中不能工作，在 Scaleform 中可能可以很好發揮作用。

需要注意的一點是當視頻剪輯 MovieClip 使用了 scale9grid，其子單元也將根據此規則進行縮放。增加 Scale9grid 到子單元將導致忽略其上級柵格，正常進行繪製。

4.8.2 強制

Constraints 類(scaleform.clik.utils.Constraints)輔助縮放元件內部的資源縮放和定位。使開發者在場景中定位資源，使資源保持到上級元件邊緣的距離。例如，捲軸 ScrollBar 元件將根據在場景中的位置重新縮放軌道大小及其位置。Constraints 在兩種元件縮放方法中都被用到。

以下代碼增加捲軸 ScrollBar 資源到 initialize() 方法的強制物件，向下方向鍵底部對齊，縮放軌道根據其上級元件進行延伸。draw() 方法函數包含了代碼以更新強制物件，從而任何元素都用此進行登記。這項更新在 draw() 函數中完成，因為在元件無效時被調用，通常在元件尺寸發生改變後。

```
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 元件和資料設置

元件需要使用一個資料源 `dataProvider` 方法的列表資料。資料源 `dataProvider` 為一個資料存儲和物件檢索單元，開放了 Scaleform CLIK `IDataProvider` 類(`scaleform.clik.interfaces.IDataProvider`)中定義的所有或部分 API 函數。

資料源 `dataProvider` 方法是用一個調用函數的請求模型，代替直接的屬性訪問。這允許資料源在需要時可以從遊戲引擎獲取資料。這具有存儲優勢，也可以將大塊資料設置為小塊，易於管理的資料。

元件使用資料源 `dataProvider` 包括任何擴展的 `CoreList` (`ScrollingList`, `TileList`)、`OptionStepper` 和 `DropdownMenu`。

4.9.1 最佳使用方法

資料源 `DataProvider` 類(`scaleform.clik.data.DataProvider`)包含在框架中使用其靜態 `initialize()`方法，將資料源 `dataProvider` 方法添加到任何 ActionScript 序列中去。元件使用一個資料源 `dataProvider` 將自動初始化序列使他們可以使用 `dataProvider` 方法進行訪問。這意味著下列語法初始化一個靜態聲明序列作為完全可操作的資料源 `dataProvider`，以及在 `IDataProvider` 中描述的方法。

```

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

```

資料源 `dataProvider` 應該實現內容為：

- *length*: 可以作為一個屬性或獲取函數返回資料設置的長度；

- *requestItemAt*:從資料源 `dataProvider` 請求一個指定項，通常被列表元件使用以在任何時間顯示一個項，如 `OptionStepper`；
- *requestItemRange*:從資料源 `dataProvider` 請求一系列項包括一個開始和結束序號。通常由列表元件使用以顯示更多項，如 `ScrollingList`；
- *indexOf*:返回項的序號；
- *invalidate*:標記資料源 `dataProvider` 變化，提供一個新的資料長度。該方法也應該指派一個“`dataChange`”事件來通知元件資料已更新。資料源 `dataProvider` 應該支援一個可以公開訪問和反應資料長度的長度屬性。

實例需要資料簡介列表能夠使用一個資料作為資料源 `dataProvider`。開發者應該明白在 ActionScript 2 中存儲資料比在應用程式中自帶資料需要更多的存儲空間和性能開銷。在大量資料設置推薦綁定資料源 `dataProvider` 和 `ExternalInterface.call` 基於需求從遊戲引擎獲取資料。

4.10 動態動畫

Scaleform CLIK 提供一個自訂的 `Tween` 類 (`scaleform.clik.motion.Tween`)，該類具有與任何可比 `Tween` 類相似的行為，但卻完全相容 Scaleform。CLIK `Tween` 支援所有標準緩動函數，例如，`fl.transitions.easing.*` 包裝下的緩動函數。

要啟動一個 `tween`，請創建一個新的 `Tween()`，並提供一個持續時間(以毫秒為單位)、目標 `Sprite`、要 `tween` 的屬性和這些屬性應 `tween` 到的值，以及一個包含作為附加 `Tween` 參數的屬性的物件。支援下列屬性/參數：

- `ease` : Function(函數)類型。緩動函數。
- `easeParam` : Object(物件)類型。您可以傳遞到緩動函數的一個額外參數。
- `onComplete` : Function 類型。`Tween` 完成時將會調用此關閉 (closure)。
- `onChange` : Function 類型。當 `Tween` 更新其值(每個刻度/幀 (tick/frame))時會調用此關閉。
- `data` : Object 類型。您需要附加到 `Tween` 的任何自訂資料(這絲毫不會影響 `Tween` 的行為)。
- `nextTween` : `Tween` 類型。當您想要連結另一個 `Tween` 時使用。當前 `Tween` 完成時，就會把連結的 `Tween` 設置為 `pause=false`。
- `frameBased` : Boolean 類型。使用基於幀的定時，而不是即時。
- `delay` : Number 類型。延遲 `tween x` 毫秒。
- `fastTransform` : Boolean。使用顯示物件屬性的矩陣數學。
- `paused` : Boolean 類型。是否暫停 `Tween`。

如果一個已經出發一個 `Tween` 的 `MovieClip` 需要創建一個新的 `Tween`，則新的 `Tween` 將停止之前的一個。然而 `Tween` 方法支援單個 `MovieClip` 的多個屬性，允許同一個物件的不同屬性在相同時間發生變化。下節中的例子展示了如何創建 `Tweens` 在同一時間影響多個屬性。

4.10.1 最佳使用方法

`Tween` 的一個實例將會把目標 `MovieClip` 從該 `MovieClip` 的當前屬性值 `tween` 到函數參數清單中指定的一個值。

要想獲知 `tween` 完成時間,只需創建一個函數引用,並將其就像在 `onComplete` 屬性中一樣添加到作為 `Tween` 的構造函數的第四個參數傳遞的物件。此回檔會在 `Tween` 完成時被調用並傳遞到一個參數,調用該參數的 `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 佈局框架

`CLIK` 佈局框架旨在說明開發者創建涉及多個元素並容易適應多種解析度的動態佈局。雖然有點簡單化,`CLIK` 佈局框架還是應說明開發者創建針對多種解析度和平臺的 UI,同時為在 `CLIK` 範圍內開發自訂佈局系統打下基礎。

注意:`Scaleform v4.0.14` 是 `CLIK AS3` 佈局框架的初始版本,而且其中的類和 API 可能隨時更改。如果您有關於 `CLIK AS3` 佈局框架的任何疑問、問題、建議或意見,請隨時通過 `Scaleform` 開發者中心打開一張諮詢單。

兩個類構成 `CLIK` 佈局框架的核心：

- **scaleform.clik.layout.Layout**：一個將管理任何 `Sprite` 的佈局的佈局,這些 `Sprite` 在同一父屬性內正確實現 `.layoutData` 屬性。
- **scaleform.clik.layout.LayoutData**：包含有關應如何對某個 `Sprite` 進行佈局的資訊的資料結構。此資料由相關佈局實例讀取,並用來創建該佈局。

在 `CLIK` 內創建一個新佈局的方法有多種。下麵介紹創建一個新佈局的最簡單的方法：

1. 創建 `Sprite/MovieClip` 並將其添加到 `Stage` 上的一個 `DisplayObjectContainer`。

2. 用 `LayoutData` 類的新實例為這些 `Sprite/MovieClip` 定義 `.layoutData` 屬性。
3. 根據需要填寫針對每個 `Sprite/MovieClip` 的 `LayoutData` 物件。 創建 `Layout` 類的一個新實例。
4. 將其添加到前面第 1 步中 `Sprites/MovieClips` 使用的同一 `DisplayObjectContainer`。

在此情況下,假如 `Layout`(佈局)實例的任何屬性均未被修改,該 `Layout` 將會管理父輩內的所有 `Sprite` 和 `MovieClip`,這些 `Sprite` 和 `MovieClip` 使用該父輩的寬度和高度作為用於定位的矩形(x、y、寬度和高度),來定義 `.layoutData` 屬性。

`Scaleform\Resources\AS3\CLIK\demos\` 目錄中的 `SDK` 中包含有此佈局框架的兩個演示。第一個演示 `Layout_Main` 可通過 `CLIK` 元件流覽器使用,該流覽器可通過 `Scaleform SDK` 流覽器打開。`Layout_Main` 演示在一個可重新調整大小的 `MovieClip` 內創建一個簡單的佈局。另一個演示 `Layout_FullScreen_Demo` 設計為在 `FxPlayer` 的一個獨立實例中運行,以便於其使用 `Stage` 的完整大小。`Layout_FullScreen_Demo` 演示使用者如何將其全屏內容用於多種解析度以及 `Layout` 系統如何處理各種參數。

4.11.1 佈局

佈局 (`Layout`) 是一個元件,它可用來對可按大小和/或比例發生變化的上下文內的佈局元素進行佈局。佈局只能管理同一父輩記憶體在的 `Sprite`(以及其中的任何子類)。也就是說,任何元素都不應作為佈局元素本身的一個子實例進行添加,但可以在該佈局實例的同一層級添加。例如,在同一層級擁有兩個子實例 (`myWindow.myMovieClip` 和 `myWindow.layoutInstance`)就允許 `layoutInstance` 管理 `myMovieClip`。

佈局僅管理 `Sprite`,該 `Sprite` 用 `LayoutData` 類的一個實例定義了 `.layoutData` 屬性。多個佈局可以存在於同一父輩內,條件是,每個佈局都具有一個唯一的 `.identifier` 屬性集,而且同一層級 `Sprite` 內的所有 `LayoutData` 還定義其 `.layoutIdentifier` 屬性。 當把佈局添加到 `Stage` 時,佈局將查看一個父輩內的所有子項,並檢查這些子項是否定義了 `.layoutData`。

從此以後,`Layout` 就會跟蹤添加到該父輩的新子項,並且,如果它們在被添加到 `Stage` 之前定義 `.layoutData`,`Layout` 就會將它們添加到其受到管理的 `Sprite/MovieClip` 清單。請注意,受到管理的 `Sprite/MovieClip` 偏移量以及它們的佈局順序僅在初次將其添加到 `Stage` 時更新。假如使用者想要重新計算從一個已更新位置的偏移量,或者對清單進行重新排序,他們可以使用下列函數：

public function reset():void

通過清空其受到管理的 `Sprite` 的清單、搜索包含 `LayoutData` 的新 `Sprite` 並重新計算偏移量 / 從頭回流那些新的 `Sprite`,來重新設置 `Layout`。

public function resortManagedSprites():void

按其 `layoutData.layoutIndex` 屬性對受到管理的 `Sprite` 進行排序,該屬性定義應用佈局的順序。如果最初設置佈局之後任何 `layoutIndex` 被更改,就應調用此函數。

使用者可以將 Layout 類附加到其 Flash 檔的庫中的符號 (Symbol) 上,以便於美術師在設計時將佈局放在 Stage 上。假如沒有將 Layout 連接到一個 Symbol(而是通過寬度和高度均為 0 的代碼進行了產生實體),該 Layout 的大小將預設設置為父輩的大小。這可以通過設置 Layout 的 .rect 屬性進行更改,該屬性定義 Layout 用來對其受到管理的元素進行佈局的區域。

理想的情況是,Layout 應作為最後一個元素添加到父輩,以確保已經正確定義了其它 Sprite/MovieClip 的所有 LayoutData 。

4.11.1.1 公共屬性

Rect	佈局的“大小”。裡面的元素將按照此屬性的 x、y、寬度和高度進行佈局。假如 Layout 的寬度 != 0(聯接到一個 Symbol,並放置到 Stage 上),則 Layout 將使用 MovieClip 的 x、y、寬度和高度。假如 Layout 的寬度 == 0(addChild() 不包含一個支援 Symbol),則 Layout 將使用父項的 x、y、寬度和高度。您也可以使用 .rect 屬性指定一個自訂的 Rectangle(矩形)。
tiedToStageSize	為 true – 假如此 Layout 始終更新以匹配 Stage 大小,否則為 false 。
tiedToParent	true – 假如此 Layout 的大小始終更新以匹配其父輩的大小,否則為 false 。
Hidden	為 true - 假如在運行時隱藏此 Layout,否則就是 false 。允許 Layout 擁有一個可見背景或預留位置圖像,該圖像將在運行時立即被設置為 visible = false 。

4.11.2 LayoutData

定義某個特定 Sprite 的佈局的資料。必須與一個有效的 scaleform.clik.layout.Layout 實例一起使用 。

4.11.2. 1 公共屬性

alignH	此 Sprite 的水準對齊。有效值 : LayoutMode.ALIGN_NONE 、 LayoutMode.ALIGN_LEFT 、 LayoutMode.ALIGN_RIGHT 。
alignV	此 Sprite 的垂直對齊。有效值 : LayoutMode.ALIGN_NONE 、 LayoutMode.TOP 、 LayoutMode.BOTTOM.property 到 TextFieldAutoSize.NONE 将保持大小不变 。
offsetH	從 Layout 或 relativeToH Sprite 的邊緣的水準偏移量。如果該偏移量保持不變 (-1),Layout 就會自動將其設置為從 Stage 上的 Layout/Sprite 的原始水準偏移量(由美術師在設計時定義)。
offsetV	從 Layout 或 relativeToH Sprite 的邊緣的垂直偏移量。如果該偏移量保持不變 (-1),Layout 就會自動將其設置為從 Stage 上的 Layout/Sprite 的原始

	垂直偏移量(由美術師在設計時定義)。
offsetHashH	一個雜湊表 (Hash Table),包含使用者定義的適用於各種長寬比的水準偏移量。如果將 Layout 聯接到 Stage 的大小 (Layout.tiedToStageSize == true),就會查詢這些值。例如,假如 Layout 聯接到 Stage 的大小,而且長寬比當前為 4:3,"LayoutData.offsetHashH[LayoutData.ASPECT_RATIO_4_3] = 70;" 將會導致此 Sprite 使用 70px 的水準偏移量。
offsetHashV	一個雜湊表 (Hash Table),包含使用者定義的適用於各種長寬比的垂直偏移量。如果將 Layout 聯接到 Stage 的大小 (Layout.tiedToStageSize == true),就會查詢這些值。例如,假如 Layout 聯接到 Stage 的大小,而且長寬比當前為 4:3,"LayoutData.offsetHashV[LayoutData.ASPECT_RATIO_4_3] = 70;" 將會導致此 Sprite 使用 70px 的垂直偏移量。
relativeToH	此 Sprite 應與其水準相關的 Sprite 的實例名稱。假如保留為 null,Sprite 就會相對於 Layout 而對齊。
relativeToV	此 Sprite 應與其垂直相關的 Sprite 的實例名稱。假如保留為 null,Sprite 就會相對於 Layout 而對齊。
layoutIndex	控制相對於同一 Layout 中的其它 Sprite 應對此 Sprite 進行佈局的順序的索引。應設置 – 假如使用 relativeToH 或 relativeToV 確保在此 Sprite 之前更新該 Sprite 的 Layout。假如 layoutIndex 保留不變 (-1),就會任意將其添加到清單末尾。
layoutIdentifier	定義應將此 LayoutData 物件與哪個 Layout 關聯(假如單個 DisplayObjectContainer 記憶體在多個 Layout)的字串。假如保留不變 (null),就會自動由同一父輩內的所有 Layout 對其進行管理。此屬性(如果設置了的話)應匹配一個 Layout 實例的識別字屬性。

4.12 彈出式支援

Scaleform CLIK 包含了 PopUpManager 類(scaleform.clik.managers.PopUpManager)，支援彈出視窗如對話方塊和提示工具。

4.12.1 最佳使用方法

PopUpManager 擁有幾個靜態方法輔助彈出視窗的創建和維護。第一個參數是一個用於唯一連結 ID 的字串。第二個參數 initProperties 允許您提供一個物件,該物件的屬性將會被覆制到創建的新 PopUp。第三和第四個參數分別針對相對於該快顯視窗的新快顯視窗定義 x 和 y 座標。relativeTo 參數,Sprite 類型,可用來定位相對於另一個 Sprite 的快顯視窗。

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

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

Scaleform 擴展 InteractiveObjectEx.setTopmostLevel() 參數用來保證彈出視窗總是顯示在場景中所有其他部件的上面，由於與庫相關的問題，使用此方案替代在 root 層創建彈出窗口。如果導入一個子 SWF 文件到上級元件，試圖創建一個在上級內容所在的路徑的庫定義的符號實例，則將產生符號查找錯誤，因為上級元件無法訪問子元素的庫。不幸的是，沒有專門工作區域來解決這個問題，因此 topmostLevel 方案為最佳選擇。

注意 InteractiveObjectEx.setTopmostLevel() 也能應用到非彈出視窗，在場景中保持此類元素的 Z 軸序列。因此，在彈出功能表頂部繪製滑鼠游標，游標可以創建在最高處位置或層次並設置 topmostLevel 屬性為 true。

4.13 拖放

DragManager 類 (scaleform.clik.managers.DragManager) 支援初始化和拖放操作。該類是一單個元件，提供了一個實例屬性訪問唯一的物件。使用此物件，開發者可以開始和結束拖放操作。

要啟動一個拖曳，請發出一個具有與新拖曳相關的屬性的 DragEvent.DRAG_STATE。

DragManager.handleStartDragEvent() 將會收到該事件，並根據該事件的屬性啟動此拖曳。然後，此拖曳將會由 DragManager 進行管理，直到收到一個 MouseEvent.MOUSE_UP 事件，因而導致調用 DragManager.handleEndDragEvent()。

4.13.1 最佳使用方法



圖 64:動作中的 DragDemo

DragManager 依靠 IDragSlot 介面，使用者介面該介面定義通過高效地與可用 IDragSlot 進行通信的函數。 DragManager 只執行拖動管理。依賴于開發者創建元件利用 DragManager 來提供拖放功能。

Scaleform 標準包括一個示例 IDragSlot 實現，稱為 DragSlot (`scaleform.clik.controls.DragSlot`)。 DragSlot 擴展 `UIComponent` 因此分類為一個 CLIK 元件。擁有幾個唯一特性作為 DragManager 的補充。 DragSlot 設計為包含一個將會被拖動的影片剪輯/雪碧/Bitmap。不過，DragSlot 也類比按鈕的許多行為，這樣使得 DragSlot 也可用作一個按鈕。歸根結底，DragSlot 只是 IDragSlot 的一個實現示例，它的許多函數已被廢止，取代以更密切地與資料後端進行通信的子類。

DragSlot 具有一個拖放類型的概念。這些拖動類型能夠按照每個 DragSlot 進行配置使元件允許或者禁止拖動操作。為實現這些拖動類型，DragSlot 也需要隨 DragManager 為 `dragBegin` 和 `dragEnd` 安裝事件監聽器。這使得 DragSlot 可以顯示其是否支援拖放操作。

要想瞭解一個功能齊全的 DragSlot 子類的示例，請查看該工具箱將 DragSlot 用作整個 UI 內的可拖曳內容的基類。網路遊戲使用者介面工具箱。

5 實例

下面小節包括一些使用 Scaleform CLIK 元件的例子。

5.1 基礎

一下實例比較簡單，但展示了易於使用的 Scaleform CLIK 框架執行常規任務。

5.1.1 包含兩個 `textField` 的 `ListItem Renderer`

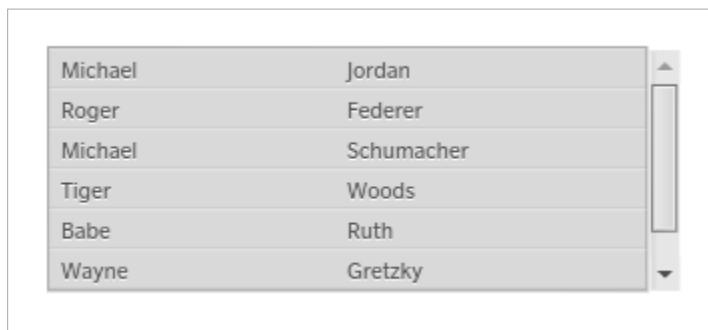


圖 65: 滾動列表 `ScrollingList` 展示包含兩個標簽的列表項

滾動列表元件默認使用 `ListItemRenderer` 來顯示行內容。然而 `ListItemRenderer` 只支援單個文本區域 `textField`。很多情況下列表項需要多個文本區域 `textField` 用來顯示，或者甚至不需要文本區域如圖示。本例展示了如何添加兩個文本區域到列表項。

首先，定義需求，物件為一個自定義 `ListItemRenderer` 支援兩個文本區域。自定義 `ListItemRenderer` 應該與滾動列表 `ScrollingList` 相容。由於目的為使得每個列表項具有兩個文本區域 `textFields`，資料也應該不止單個字串列表。本例中我們使用以下資料源 `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"}];
```

資料源包含的物件具有兩個屬性：`fname` 和 `lname`。該兩個屬性將顯示在兩個列表項的文本區域 `textField` 中。

由於默認的 `ListItemRenderer` 只支援一個文本區域 `textField`，需要能夠支援兩個文本區域 `textField`。最簡單的實現方法為將 `ListItemRenderer` 類作為子類。一下為調用 `MyItemRenderer` 類的源代碼，其中使用了 `ListItemRenderer` 子集並添加了兩個文本區域的基本功能支援。（代碼位於 `MyItemRenderer.as` 文件中，文件位於 FLA 工作目錄）：

```
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` 方法在本子類中沒有涉及。`setData` 方法在列表項收到來自容器列表元件(ScrollingList、TileList 等)項資料時被調用。在 `ListItemRenderer` 中，本方法設置一個 `textField` 的值，而 `MyItemRenderer` 用來設置 `textField` 的值，同時也存儲一個索引到內部專案物件。此專案物件在 `updateAfterStateChange` 方法中被重用，此方法在 `ListItemRenderer` 狀態發生改變時被調用，此狀態的變化需要文本區域 `textField` 刷新值。

到此，已經定義了具有複雜列表項支援列表項渲染的 `ListItemRenderer` 類的資料源 `dataProvider`。要連接所有相關列表元件，必須創建一個符號以支援該型的 `ListItemRenderer` 類。本例中，最快的實現方法為修改 `ListItemRenderer` 符號使其具有兩個文本區域 `textFild` 調用 ‘`textField1`’和‘`textField2`’。下一步該符號標識和類必須修改為 `MyItemRenderer`。為在列表中使用 `MyItemRenderer` 元件，修改列表實例的 `itemRenderer` 檢查屬性，從 ‘`ListItemRenderer`’改變到 ‘`MyItemRenderer`’。

現在運行 FLA，列表應該顯示出來包含列表元素顯示兩個標簽：在資料源 `dataProvider` 中設置的 `fname` 和 `lname` 屬性。

5.1.2 圖元滾動視圖

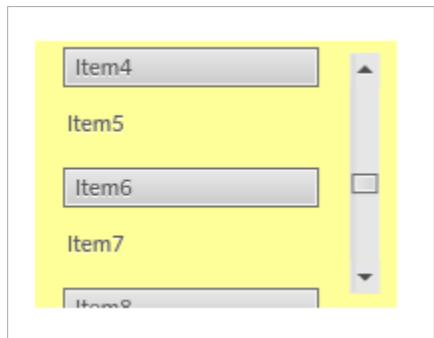


圖 66: 按圖元滾動包含 CLIK 元素的視圖

按圖元滾動通常用於複雜用戶介面。本例展現了如何使用 CLIK 滾軸 ScrollBar 元件簡單得實現此功能。

首先，創建一個新的符號用於滾動視圖，這提供了一個容器，簡化了所需的偏移量計算。在滾動視圖容器中，從頂部到底部順序設置一下圖層。這些圖層並不需要，但是用來作為說明：

- **actions**: 包含 ActionScript 代碼使得滾動視圖工作；
- **scrollbar**: 包含一個 CLIK 滾軸 ScrollBar 實例，該實例稱為 ‘sb’ ；
- **mask** :由矩形或 MovieClip 定義的蒙板圖層，設置圖層屬性為一個蒙板；
- **content** :包含需要滾動的內容；
- **background** :可選層包含了一個背景突出無蒙板區域。

添加相關元素到圖層並創建一個符號將其保存。通過創建一個符號保存所有的內容，滾動內容的任務變得十分簡單。調用這些內容實例 ‘content’並設置 y 軸為 0。這確保滾動邏輯上不需要計算不同的偏移量。然而，這些偏移根據滾動視圖的複雜性也許需要用到。

到此，定義了所需創建的一個簡單滾動視圖，以及需要互相關聯的元素名稱結構。將以下 ActionScript 代碼放到 code 圖層的第一幀：

```
// 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;
}
```

由於捲軸不與其他元件連接，需要手動配置。**setScrollProperties** 方法就是用來使其在一個位置滾動，並設置完全顯示內容的最大值和最小值。本例中的捲軸位置為圖元格式。運行文件，滾動視圖可以通過與捲軸交互進行改變。

6 常見問題解答

1. 常見問題解答

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。)
```

您之所以收到此錯誤,是因為該類的某個元素被定義了兩次。如果沒有將一個 .FLA 設置為禁用 "Automatically Declare Stage Instances"(這會導致任何 ActionScript 定義與發佈 .FLA 時 .FLA 自動生成的定義之間發生衝突),此錯誤消息最常出現。

CLIK 架構要求將此設置禁用才能使所有 CLIK 元件正常工作,因為通常要求明確定義 Stage 元素,以便於引用 ActionScript 類中的強類型子項

。

您可以通過打開 File -> Publish Settings -> "Flash" tab -> Actionscript 3.0 Settings 並確保不選中 "Automatically declare stage instances",來針對某個 .FLA 禁用 "Automatically Declare Stage Instances"。請注意,此設置不是 Flash Studio 全域設置,並且僅保存到特定 .FLA 。

2. 我正從另一個 .FLA 的庫中導入一個 CLIK 元件。當我將該元件的一個實例放在 Stage 上並更改其可檢查 (inspectable) 屬性時,收到以下錯誤：

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：) ...
```

當您的元件聲明不適當時最常發生此錯誤。一般情況下,您可以使用下列步驟解決此錯誤：

- 確保元件有一個有效的實例名稱。
- 確保元件是在 ActionScript 中定義的。
- 確保 ActionScript 定義引用正確的類。有時,這可能不是元件的匯出名稱 – 而應該使用 ActionScript 類名稱。例如 ,myButton:Button (scaleform.clik.controls.), 而不是 myButton:MyButtonSymbol,其中,MyButtonSymbol 是導入的符號的匯出名稱,而 myButton 是元件的實例名稱。

3. 我的庫中有兩個元件,它們是從同一基類中派生而來的。當我發佈我的 .SWF 檔時,我收到以下錯誤：

```
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.
```

這些錯誤通常是由於一個符號的類名稱與其基類的名稱相同導致的。例如，一個符號具有類“Button”和基類“scaleform.clik.controls.Button”。這會導致該符號的時間線定義與基類合併在一起。因此，共用同一基類的任何其它符號都會繼承那些時間線定義，因而導致意外的行為。

要解決此問題，請檢查瞭解是否存在一個基類，該基類有多個符號與其關聯，然後確保沒有任何符號的類名稱匹配基類的名稱。更改共用同一名稱的符號的類就會解決此問題。

7 潛在缺陷

避免一個符號的類的名稱與其基類名稱相同,除非不再有其它符號會擴展該基類。如果一個符號具有與其基類相同的類名稱,其時間線定義將會與基類合併,這可能會對該基類的其它子類造成問題。

假如該類不會有其它子類存在,我們建議將基類用作該類,並且不要為符號提供任何基類。這樣會緊密耦合基類和符號,因而在您產生實體該基類的一個新實例時,就會自動將符號聯接到該實例,而不必使用像 `getDefinitionByName()` 這樣的查找。

8 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.

