Autodesk®
Scaleform®

Scaleform CLIK ユーザー ガイド

本書では Scaleform CLIK フレームワークと、付属のプリビルド コンポーネントの実装について詳しく説明しています。

著者: Prasad Silva、Matthew Doyle
バージョン: 2.0
最終更新日: 2010 年 8 月 19 日

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.4

Disclaimer

Autodesk Scaleform の連絡先:

ドキュメント	Scaleform CLIK User Guide (Scaleform CLIK ユーザー ガイド)
住所	Autodesk Scaleform Corporation 6305 Ivy Lane, Suite 310 Greenbelt, MD 20770, USA
ホームページ	www.scaleform.com
電子メール	info@scaleform.com
電話	(301) 446-3200
Fax	(301) 446-3199

目次

1 はじめに

本書では、Scaleform® Common Lightweight Interface Kit (CLIK™) のフレームワークとコンポーネントの実装について詳しく説明します。この CLIK User Guide を読み進める前に、「Getting Started with CLIK」と「 Getting Started with CLIK Buttons」に目を通すことをお勧めします。この 2 つのマニュアルは Scaleform CLIK を起動して実行するまでに必要なステップの説明、基本的なコア コンセプトの紹介、さらに CLIK コンポーネントの作成とスキニングに関する詳細なチュートリアルを提供しています。ただし、上級ユーザーは本書をすぐに熟読するか、または上記のマニュアルと並行して参照したい場合もあるでしょう。

1.1 はじめに

Scaleform CLIK は、ActionScript™ 2.0 (AS) ユーザー インターフェイス (UI) エレメント、ライブラリ、ワークフロー拡張ツールのセットです。Scaleform 4.4 以降のユーザーが、家庭用ゲーム機・PC・携帯ゲーム機アプリケーションに、リッチで効率的な UI をすぐに実装できるように用意しました。このフレームワークの主要目的は、(メモリと CPU の使用の点で) 軽量化し、簡単にスキニングを行い、大幅にカスタマイズできるようにすることです。UI コア クラスとシステムの基本フレームワークの外、CLIK には 15 以上のプリビルド汎用インターフェイス エレメント (ボタン、スライダ、テキスト入力など) が添付されており、これらのエレメントを使うと開発者は直ちにユーザー インターフェイス画面を作成、イテレートすることができます。

1.1.1 Scaleform CLIK の内容

コンポーネント: 簡単で拡張可能なプリビルドの UI コントロール

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

クラス: コアの システム API

InputManager	Locale
FocusManager	Tween
DragManager	DataProvider
PopUpManager	IdataProvider

EventDispatcher	Ilist
Delegate	IlistItemRenderer
Constraints	GameDataProvider

マニュアルとサンプル ファイル:

Getting Started with CLIK
Getting Started with CLIK Buttons
CLIK API Reference
CLIK User Guide
CLIK Flash Samples
CLIK Video Tutorials

1.2 コンポーネントについて

始める前に、開発者は Flash コンポーネントについて正確に理解することが重要です。Flash は一連のデフォルトのインターフェイス作成ツールと、コンポーネントと呼ばれる構成ブロックを内蔵しています。しかし、本書の「コンポーネント」とは、ワールドワイドで知られている gskinner.com チームと共同で、Scaleform が開発した Scaleform CLIK フレームワークを使って作成されたプリビルドのコンポーネントを指しています。

gskinner.com はグラント・スキナー氏が率いており、彼は Adobe の命によって Flash Creative Suite® 4 (CS4) のコンポーネントを作成しました。gskinner.com は優れた Flash チームの 1 つとして世界的に知られています。gskinner.com の詳細については、http://gskinner.com/blog を参照してください。

プリビルドの CLIK コンポーネントについてさらに詳しく理解するため、以下のデフォルトの CLIK ファイル パレットを Flash Studio で開きます:

C:\Program Files\Scaleform\GFx 4.4\Resources\AS2\CLIK\components\CLIK_Components.fla

1.3 UI についての考察

優れた UI を作成するには、まず始めに紙や、Microsoft Visio®などの図形エディタでそのコンセプトを立案します。次に、UI の特定のグラフィック デザインを始める前に、すべてのインターフェイス オプションとその流れを考え出すために、Flash で UI のプロトタイプ化を始めます。Scaleform CLIK は特に、ユーザーが直ちにプロトタイプ化して、完成するまでイテレートすることができるように設計されています。

フロント エンド UI を作成する方法はいくつもあります。Flash では、Visio や他のフローチャート プログラムと同じようには、ページの概念は存在していません。キーフレームとムービー クリップがその役割を果たします。あなたが作ったすべてのページを同じ Flash ファイルに入れ込んだ場合、そのファイルはより多くのメモリを使用する可能性がありますが、ペー

ジ間の移動が簡単で速くなる場合もあります。各ページをそれぞれ個別のファイルにすると、全体のメモリ使用量は少なくなるかもしれませんが、ロード時間が長くなる場合があります。また、各ページを個別のファイルとして持つことは、複数のデザイナーやアーティストが同じインターフェイスを同時に作業できるという利点もあります。技術的な必要条件や設計上の必要条件に加えて、自身の UI プロジェクトの構造を決めるときにワークフローを必ず考慮に入れてください。

従来のデスクトップや Web アプリケーションとは異なり、優れたマルチメディア ゲーム インターフェイスを作成する方法はいくつもあるということを理解しておくことが大切です。それぞれのエンジン、さらに数えるほどしかない各プラットフォームに対しても、少しずつ異なるアプローチがあると言えます。選択が、結果として良く出ることもあれば、悪く出てしまうこともあるでしょう。たとえば、複数ページのインターフェイス デザインを 1 つの Flash ファイルに入れると仮定します。こうすると管理が楽になりトランジションの作成が簡単になりますが、メモリの使用量も増えるので、旧型の家庭用や携帯ゲーム機には悪影響を及ぼす場合もあります。1 つの Flash ファイルですべてを行う場合、複数のデザイナーが同時にそのインターフェイスの別々のパーツで作業することができません。複雑なインターフェイスが求められるプロジェクトで、大人数のデザイン チームが存在する、またはその他の特定の技術上、あるいは設計上の必要条件がある場合、UI の各ページを個別の Flash ファイルに分けるほうが良い場合があります。多くの場合、両方のソリューションとも効果がありますが、そのプロジェクトに対しては、1 つのソリューションが他方のソリューションよりも大幅な利点を提供できる場合があります。たとえば、画面をオンデマンドでロード/アンロードしたり、動的に更新したり、ネットワーク経由でストリーミングすることが必要な場合もあります。つまり、UI のアセット管理は、論理的なレイアウトから実装のワークフローとパフォーマンスの考察に至るまで、常に周到に行うべきです。

Scaleform は開発者が UI の動作の大部分に Flash と ActionScript を使用することを強く推奨していますが、完璧な答えはないので、アプリケーション エンジンのスクリプト言語 (Lua や UnrealScript など) を使って、難しい作業の大部分を行いたいと思うチームもあるかもしれません。このような場合、Flash はどちらかというと、、ファイル自体に少量の ActionScript とインタラクティビティを備えたアニメーション プレゼンテーション ツールとして使われる必要があります。

プロジェクトを先の方まで進行させてしまう前に、技術的、設計上、さらにワークフローの必要条件を早い段階で理解し、次にそのプロセスを通した全体的な手法を継続的に評価して、円滑で良好な結果が出ることを確実にしておくことが、とりわけ重要です。

2 プリビルド コンポーネント

一見すると Scaleform CLIK はプリビルド UI コンポーネントの基本セットに見えますが、CLIK の真意はより豊富なコンポーネントとインターフェイスを作成するためのフレームワークを提供することです。開発者は自由に 、とはいえ、ある程度は外部から求められて、ニーズに合わせてカスタムのコンポーネントを作成し、この CLIK フレームワークに積み重ねることができます。

プリビルド CLIK コンポーネントは、基本的なボタンやチェックボックスから、リスト ボックス、ドロップダウン メニュー、モーダルなダイアログ ボックスなどの複雑な複合コンポーネントまでの標準の UI 機能を備えています。開発者は簡単に、これらの標準のコンポーネントを拡張して機能を追加することが出来、一方、カスタム コンポーネントを最初から作成するためのリファレンスとして使うこともできます。

これ以降の章では、各プリビルド コンポーネントについて詳細に説明しています。コンポーネントは複雑さと機能ごとにグループ化されています。それぞれのコンポーネントの説明には、以下のサブセクションを伴っています。

- **ユーザーの操作**：ユーザーがそのコンポーネントを操作する方法。
- **コンポーネントのセットアップ**：Flash オーサリング環境でコンポーネントを作成するときに必要なエレメント。
- **ステート**：コンポーネントが動作するために必要な各種のビジュアル ステート (キーフレーム) 。
- **検証可能なプロパティ：**コードを使わずにコンポーネントの特定の状態を簡単に作成するために、Flash オーサリング環境で公開されているプロパティ。
- **イベント：**UI の他のオブジェクトがリッスン可能なコンポーネントによって発生するイベントのリスト。
- **ヒントとコツ：**そのコンポーネントに関連したさまざまなタスクの達成に役立つコード サンプル。

2.1 基本的なボタンとテキストのタイプ

基本タイプには Button、CheckBox、Label、TextInput、TextArea コンポーネントが含まれています。ほとんどのユーザー インターフェイスにおいて Button は重要な要素であり、CheckBox の機能はこの Button から派生しています。Label は静的なラベル タイプですが、TextInput と TextArea は一行と複数行の textField タイプを表しています。

図 1: *Free Realms* – メインメニューのボタンの例

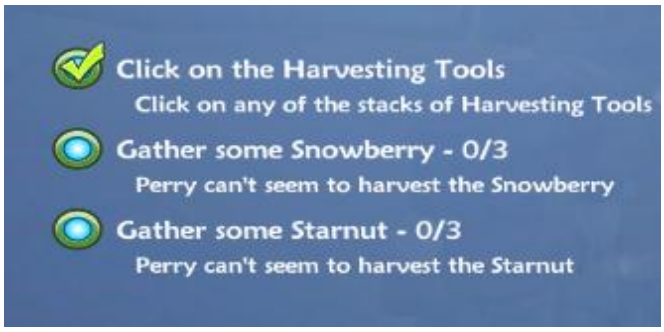

図 2: *Free Realms* - チェックボックスの例

図 3: *Free Realms* - ラベルの例

図 4: *Crysis Warhead* - テキスト入力（TextInput）の例

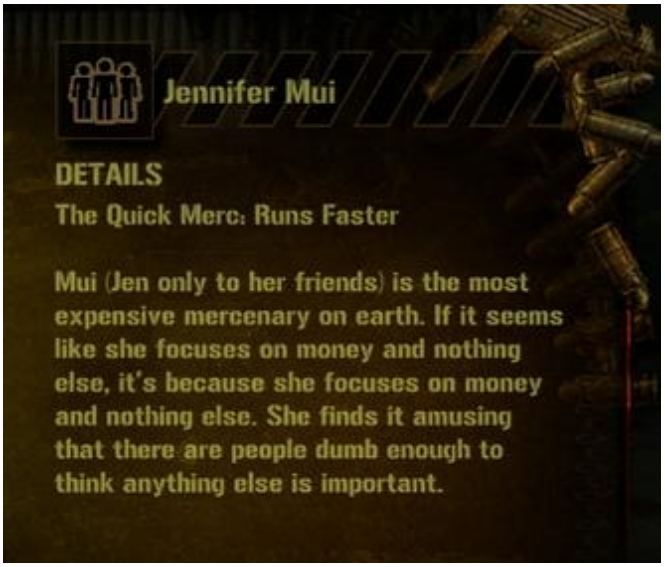

図 5: *Mercenaries 2* - テキスト領域（TextArea）の例

2.1.1 Button

図 6: スキニング前のボタン

Button は CLIK フレームワークの基礎コンポーネントです。クリック可能なインターフェイス コントロールが必要な箇所であればどこでも使えます。デフォルトの Button クラス (gfx.controls.Button) はラベルを表示する textField と、視覚的にユーザーの操作を示すステートをサポートしています。Button は単独でも、ScrollBar の矢印や Slider のサムなど複合コンポーネントの一部としても使用することができます。クリックでアクティブになるインタラクティブ コンポーネントのほとんどは、Button で構成、または拡張しています。

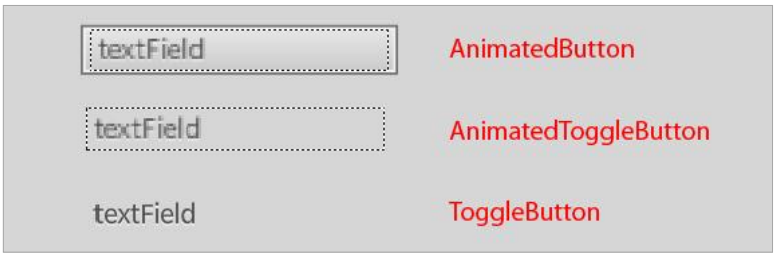

図 7: AnimatedButton、AnimatedToggleButton、ToggleButton

CLIK Button は汎用のボタン コンポーネントです。マウス操作、キーボード操作、ステート、さらにその他の機能をサポートしていますので、多くのユーザー インターフェイスでの使用が可能です。また、切り替え機能やアニメーション ステートもサポートします。Button.fla コンポーネント ソース ファイルで提供される ToggleButton、AnimatedButton、AnimatedToggleButton はすべて、同じ基本となるコンポーネント クラスを使用しています。

2.1.1.1 ユーザーの操作

Button コンポーネントはマウスやその他の同等のコントローラを使って押すことができます。また、フォーカスが適用されたときには、キーボードを使って押すこともできます。

一般的に、フォーカスが適用されたコンポーネントだけがキーボード イベントを受け取ります。コンポーネント上にフォーカスを設定する方法はいくつもあります。その方法の一部は、本書の「プログラミングの詳細」で説明しています。ほとんどの CLIK コンポーネントは、操作されているとき、特にマウスの左ボタン、あるいは同等のコントローラがそのコンポーネント上で押された (クリックされた) ときにもフォーカスが適用されます。Tab キーや

Shift+Tab キー (または同等のナビゲーション コントロール) は、表示されているフォーカス可能なコンポーネント間でフォーカス移動します。これはほとんどのデスクトップ アプリケーションや Web サイトで目にするものと同じ動作です。CLIK 以外のエレメントで使用されるフォーカスは、CLIK コンポーネントでも使用されます。つまり、シーン内で Tab キーと Shift+Tab キーを使ってフォーカスを移動するような場合に、Flash 開発者は CLIK エレメントと非 CLIK エレメントをうまく組み合わせて、意図したとおりにフォーカスを動かすことができます。

デフォルトでは Button は Enter キーとスペースバーに応答します。マウス カーソルが Button 上を移動したり、Button から離れるとき、さらに Button にドラッグイン/ドラッグアウトするときもコンポーネントに影響します。

コンソールや携帯ゲーム機では、開発者は簡単にゲームパッドのコントロールを、キーボードやマウスのコントロールにマップすることができます。たとえば、Enter キーは通常、Xbox360 や PS3 のコンソール コントローラの A ボタンと X ボタンにそれぞれマップされます。このマッピングによって CLIK で作成した UI は、各種のプラットフォームで動作できます。

2.1.1.2 コンポーネントのセットアップ

CLIK Button クラスを使用する MovieClip には、以下の指定されたサブエレメントが必須です。オプションのエレメントである場合には、以下に表記されています。

- *textField :* (オプション) TextField タイプ。ボタンのラベルです。
- *focusIndicator :* (オプション) MovieClip タイプ。フォーカスが適用されたステートの表示に使用される別の MovieClip です。存在する場合、この MovieClip は "show" と "hide" の 2 つの指定されたフレームを備えている必要があります。デフォルトでは *over* ステートを使って、フォーカスされた Button コンポーネントを表しています。ただし場合によっては、この動作はあまりに限定的ともいえます。デザイナーはフォーカスが適用されたステートと、マウス オーバー ステートを区別したい場合があるからです。focusIndicator サブエレメントはこのような場合に便利です。このサブエレメントを追加すると、Button ステートはフォーカスが適用されたステートに影響を受けません。コンポーネント ステートの詳細については、次の章を参照してください。

2.1.1.3 ステート

CLIK Button コンポーネントはユーザーの操作に基づいた様々なステートをサポートします。これらのステートには以下のものが含まれます:

- *up* またはデフォルトのステート
- マウス カーソルがコンポーネント上にあるとき、またはフォーカスが適用されているときの *over* ステート
- ボタンが押下されたときの *down* ステート

- *disabled* ステート

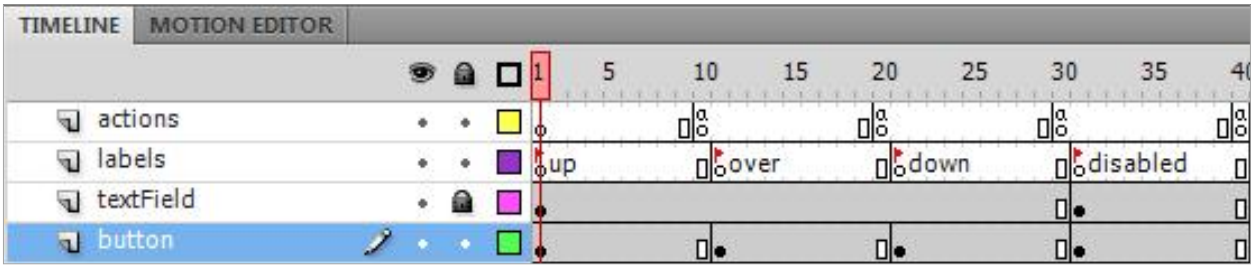

図 8: Button のタイムライン

これらのステートは Flash のタイムラインではキーフレームとして表現され、Button コンポーネントが正しく動作するために必要な最小限のキーフレーム セットです。コンポーネントの機能を拡張して、複雑なユーザー操作やアニメーション トランジションをサポートするその他のステートもあります。このようなステートについては、「Getting Started with CLIK Buttons」を参照してください。

2.1.1.4 検証可能なプロパティ

コンポーネントの重要なプロパティは、Flash IDE の [コンポーネント インスペクタ] パネル、または [パラメータ] タブで構成されています。このパネルを CS4 で開くには、[ウィンドウ] メニューの [コンポーネント インスペクタ] を選択するか、またはキーボードで Shift + F7 キーを押します。これで [コンポーネント インスペクタ] パネルが表示されます。このようなプロパティは「検証可能なプロパティ」と呼ばれています。このパネルでは、AS プログラミングに慣れていないアーティストやデザイナーに、コンポーネントの動作や機能を構成するための便利な方法を提示しています。

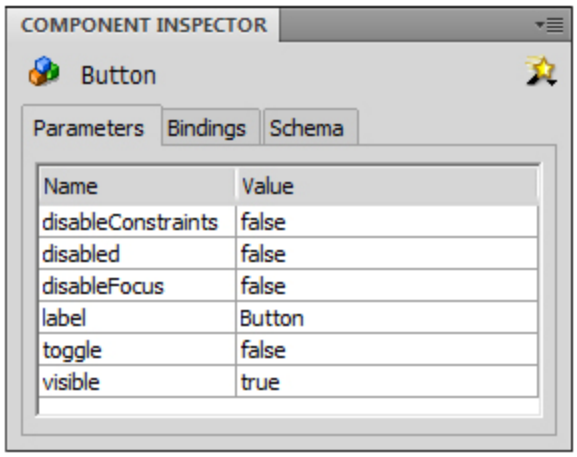

図 9: CS4 の [コンポーネント インスペクタ] に表示された Button コンポーネントの検証可能なプロパティ

Button コンポーネントの検証可能なプロパティは以下のとおりです:

disableConstraints	この Button コンポーネントには、コンポーネントのサイズを

	変更したときに、ボタン内部の textField の配置やスケーリングを決定する constraints (制約) オブジェクトが含まれています。このプロパティを true に設定すると、constraints オブジェクトは無効になります。タイムライン アニメーションでボタンの textField のサイズ変更や再配置の必要がある場合、このプロパティは特に便利です。さらに、そのボタン コンポーネントは、サイズが変わりません。無効にしない場合、その textField は有効期間中ずっとそのデフォルトの値に移動しスケーリングされるので、ボタンのタイムラインに作成された可能性のある textField トランジション/スケーリング トゥイーンを無効にします。
Disabled	true に設定した場合、ボタンを無効にします。一度無効にすると、ボタンはフォーカスを適用されなくなります。
disableFocus	デフォルトでは、ボタンはユーザーの操作に対してフォーカスを適用されます。このプロパティを true に設定すると、フォーカスの取得ができなくなります。
Label	Button 内部に表示されるテキストを設定します。
Toggle	Button のトグル (切り替え) モードを設定します。true に設定すると、この Button はトグル ボタンとして動作します。
Visible	false に設定した場合、ボタンを非表示にします。

これらのプロパティを変更しても、そのコンポーネントを含む SWF ファイルをパブリッシュするまではその変更は確認できません。CLIK コンポーネントはコンパイルされたクリップではないため、Flash IDE は設計時のステージ上に変更を表示しません。。これは、コンポーネントが簡単にアクセスでき、スキニング可能であるために必要なことです。

2.1.1.5 イベント

ほとんどのコンポーネントはユーザーの操作、ステートの変更、そしてフォーカスの管理のためのイベントを作成します。これらのイベントは、CLIK コンポーネントを使って有効なユーザー インターフェイスを作成するために重要となります。

すべてのイベント コールバックは、そのイベントの関連情報を含む 1 つの Object パラメータを受け取ります。以下のプロパティは、すべてのイベントに共通です。

- *type :* イベントのタイプ
- *target :* イベントを作成したターゲット

Button コンポーネントが作成するイベントは以下の表のとおりです。イベントの隣に表示されているプロパティは、共通プロパティに追加して提供されています。

Show	visible プロパティが、実行時に true に設定されました。
Hide	visible プロパティが、実行時に false に設定されました。
focusIn	ボタンにフォーカスが、適用されました。
focusOut	ボタンのフォーカスが、解除されました。

Select	selected プロパティが、変更されました。 *selected :* Button の選択状態で、true の場合は選択されています。ブール型です。
stateChange	ボタンのステートが、変更されました。 *state :* Button の新規ステートです。文字列タイプで、値は "up"、"over"、"down" などです。ステートの全リストについては、「Getting Started with CLIK Buttons」を参照してください。
rollOver	マウス カーソルが、ボタン上に移動しました。 *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
rollOut	マウス カーソルが、ボタンから離れました。 *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
Press	ボタンが、押下されました。 *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
doubleClick	ボタンが、ダブルクリックされました。*doubleClickEnabled* プロパティが true の場合に限って発生します。 *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
Click	ボタンが、クリックされました。 *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
dragOver	マウス カーソルが (マウスの左ボタンを押したままの状態で)、 ボタン上にドラッグされました。 *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
dragOut	マウス カーソルが(マウスの左ボタンを押したままの状態で)、 ボタンから離れました。 *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
releaseOutside	マウス カーソルが、ボタンから離れ、マウスの左ボタンが、解放されました。 *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。

ActionScript コード スニペットがこれらのイベントのキャッチや処理に必要となります。以下の例ではボタンのクリックを処理する方法を示しています:

```
myButton.addEventListener("click", this, "onButtonPress");
function onButtonPress(event:Object) {
	// ここに何かを追加する
}
```

一行目は 'myButton' という名前のボタンに、“click” イベントのイベント リスナーを追加して、そのイベントが起きたときに、onButtonPress 関数を呼び出すように指示しています。同じコード パターンをこれ以外のイベントの処理にも使用することができます。イベント ハンドラで返される Object パラメータ (この例では 'event' という名前です) は、そのイベントの関連情報を含んでいます。この情報は Object パラメータのプロパティとして返され、ほとんどのイベントで異なります。

2.1.1.6 ヒントとコツ

カスタムのプロパティを持つ Button コンポーネントを実行時に作成する:

```
import gfx.controls.Button;
attachMovie("Button", "btnInstanceName", someDepth, {_x:33, _y:262, …});
```

CLIK Button インスタンスをセットアップして、選択解除と選択されたステートを切り替えます。検証可能なプロパティで toggle プロパティが設定されている場合は、これは必要ありません:

```
btn.toggle = true;
```

マウスの左ボタン ダブルクリックを有効にする:

```
btn.doubleClickEnabled = true;
btn.addEventListener("doubleClick", this, "onDoubleClick");
function onDoubleClick(e:Object):Void {
	// ボタンがダブルクリックされた!
}
```

ボタンが押されたままのときに、click イベントの繰り返しを有効にする:

```
btn.autoRepeat = true;
```

2.1.2 CheckBox

図 10: スキニング前の CheckBox

CheckBox (gfx.controls.CheckBox) は、クリックされたときに選択状態を切り替えるように設定された Button コンポーネントです。CheckBoxes は true/false (ブール) 値の表示と変更に使用されます。これは機能的には ToggleButton と同じですが、意識することなく toggle プロパティを設定します。

2.1.2.1 ユーザーの操作

マウス、または同等のキーボード コントロールを使って CheckBox コンポーネントをクリックすると、連続してそのコンポーネントを選択/選択解除します。それ以外の点では、この CheckBox は Button と同じように動作します。

2.1.2.2 コンポーネントのセットアップ

CLIK CheckBox クラスを使用する MovieClip は以下の指定されたサブエレメントが必須です。オプションのエレメントである場合には、以下に表記されています:

- *textField :* (オプション) TextField タイプ。ボタンのラベルです。
- *focusIndicator :* (オプション) MovieClip タイプ。フォーカスが適用されたステートの表示に使用される別の MovieClip です。存在する場合、この MovieClip は "show" と "hide" の 2 つの指定されたフレームを備えている必要があります。

2.1.2.3 ステート

自身の toggle プロパティのために、CheckBox は選択状態を表す別のキーフレーム セットが必要となります。このようなステートには以下が含まれます:

- *up* またはデフォルトのステート

- マウス カーソルがコンポーネント上にあるとき、またはフォーカスが適用されているときの *over* ステート
- ボタンが押下されたときの *down* ステート
- *disabled* ステート
- *selected_up* またはデフォルトのステート
- マウス カーソルがコンポーネント上にあるとき、またはフォーカスが適用されているときの *selected_over* ステート
- ボタンが押下されたときの *selected_down* ステート
- *selected_disabled* ステート

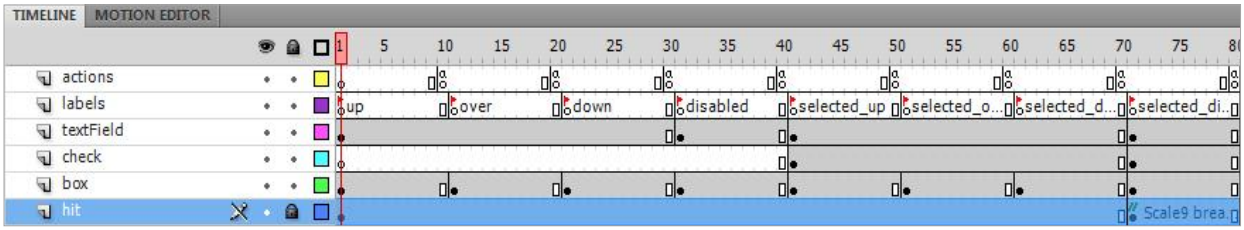

図 11: CheckBox のタイムライン

これは CheckBox に必要な最小限のキーフレーム セットです。Button コンポーネントがサポートするステートとキーフレームの拡張セットと、それに伴う CheckBox コンポーネントについては、「Getting Started with CLIK Buttons」を参照してください。

2.1.2.4 検証可能なプロパティ

CheckBox は Button コントロールから生じているので、このコンポーネントには Button と同じ検証可能なプロパティが含まれていますが、toggle と disableFocus プロパティは除外されます。

Data	CheckBox のラベルとは別のデータとして、コンポーネントに追加できるカスタムの文字列です。
disableConstraints	この Button コンポーネントには、コンポーネントのサイズを変更したときに、ボタン内部の textField の配置やスケーリングを決定する constraints (制約) オブジェクトが含まれています。このプロパティを true に設定すると、constraints オブジェクトは無効になります。タイムライン アニメーションでボタンの textField のサイズ変更や再配置の必要がある場合、このプロパティは特に便利です。さらに、そのボタン コンポーネントはサイズが変わりません。無効にしない場合、その textField は有効期間中ずっとそのデフォルトの値に移動しスケーリングされるので、ボタンのタイムラインに作成された可能性のある textField トランジション/スケーリング トゥイーンを無効にします。
disabled	true に設定した場合、ボタンを無効にします。
Label	Button のラベルを設定します。

selected	true に設定されているときに、CheckBox をオンに (または選択) します。
Visible	false に設定した場合、ボタンを非表示にします。

2.1.2.5 イベント

すべてのイベント コールバックは、そのイベントの関連情報を含む 1 つの Object パラメータを受け取ります。以下のプロパティは、すべてのイベントに共通です。

- *type :* イベントのタイプ
- *target :* イベントを作成したターゲット

CheckBox コンポーネントが作成するイベントは以下の表のとおりです。イベントの隣に表示されているプロパティは、共通プロパティに追加して提供されています。

Show	visible プロパティが、実行時に true に設定されました。
Hide	visible プロパティが、実行時に false に設定されました。
focusIn	このコンポーネントにフォーカスが、適用されました。
focusOut	このコンポーネントのフォーカスが、解除されました。
Select	コンポーネントの selected プロパティが、変更されました。 *selected :* Button の selected プロパティです。ブール型です。
stateChange	ボタンのステートが、変更されました。 *state :* Button の新規ステートです。文字列タイプで、値は "up"、"over"、"down" などです。ステートの全リストについては、「Getting Started with CLIK Buttons」を参照してください。
rollOver	マウス カーソルが、ボタン上に移動しました。 *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
rollOut	マウス カーソルが、ボタンから離れました。 *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
Press	ボタンが、押下されました。 *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
doubleClick	ボタンが、ダブルクリックされました。*doubleClickEnabled* プロパティが true の場合に限って発生します。 *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
Click	ボタンが、クリックされました。 *mouseIndex :* このイベントの作成に使用されたマウス カーソル

	のインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
dragOver	マウス カーソルが (マウスの左ボタンを押したままの状態で)、 ボタン上にドラッグされました。 *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
dragOut	マウス カーソルが(マウスの左ボタンを押したままの状態で)、 ボタンから離れました。 *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
releaseOutside	マウス カーソルが、ボタンから離れ、マウスの左ボタンが解放されました。 *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。

以下のサンプル コードは CheckBox の切り替え方法を示しています。

```
myCheckBox.addEventListener("select", this, "onCheckBoxToggle");
function onCheckBoxToggle(event:Object) {
      // ここに何かを追加する
}
```

2.1.3 Label

図 12: スキニング前の Label

CLIK Label コンポーネント (gfx.controls.Label) は簡単に言うと、便利な機能が少し追加された MovieClip シンボルでラップされた編集不可の標準の textField です。内部では、この Label は標準の textField と同じプロパティと動作をサポートしますが、このコンポーネント自体は一般的に使用されるほんの一握りの機能しか公開していません。ユーザーがプロパティを直接変更する必要がある場合、Label の実際の textField にアクセスすることができます。後述するような特定のケースでは、開発者はこの Label コンポーネントの代わりに標準の textField を使用することもできます。

Label は MovieClip シンボルなので、標準の textField では不可能な、グラフィック エレメントで装飾することができます。シンボルなので、textField インスタンスのように、インスタンスごとに構成する必要はありません。また、Label はタイムラインで定義できる disabled ステートも提供します。一方で、標準の textField でこの動作を模倣するには、複雑な AS2 コードが必要となります。
Label コンポーネントはデフォルトで制限があります。つまり、ステージで Label インスタンスのサイズを変更しても、実行時に目に見える効果はないということです。textField のサイズ変更が必要な場合、ほとんどのケースで開発者は、Label ではなく標準の textField を使用する必要があります。一般的に、テキスト エレメントに一貫した再利用性が必要ではない場合、標準の textField の方が Label コンポーネントよりも軽量です。

2.1.3.1 ユーザーの操作

Label ではユーザーの操作は一切できません。

2.1.3.2 コンポーネントのセットアップ

CLIK Label クラスを使用する MovieClip には、以下の指定されたサブエレメントが必須です。オプションのエレメントである場合には、その旨表記されています:

- ***textField*** *:* TextField タイプ。Label のテキストです。

2.1.3.3 ステート

CLIK Label コンポーネントは disabled プロパティに基づいた 2 つのステートをサポートします:

- *default* または有効のステート
- *disabled* ステート

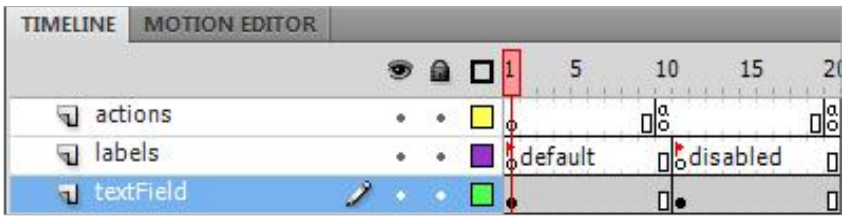

図 13: Label のタイムライン

2.1.3.4 検証可能なプロパティ

Label コンポーネントの検証可能なプロパティは以下のとおりです:

Text	ラベルのテキストを設定します。
Visible	false に設定した場合、ラベルを非表示にします。
disabled	true に設定した場合、ラベルを無効にします。

2.1.3.5 イベント

すべてのイベント コールバックは、そのイベントの関連情報を含む 1 つの Object パラメータを受け取ります。以下のプロパティは、すべてのイベントに共通です。

- *type :* イベントのタイプ
- *target :* イベントを作成したターゲット

Label コンポーネントが作成するイベントは以下の表のとおりです。イベントの隣に表示されているプロパティは、共通プロパティに追加して提供されています。

Show	コンポーネントの visible プロパティが、実行時に true に設定されました。
Hide	コンポーネントの visible プロパティが、実行時に false に設定されました。
stateChange	Label のステートが、変更されました。 *state :* 新規の Label ステートです。文字列タイプで、値は "default"または "disabled" です。

以下の例は Label ステートの変更をリッスンする方法について示しています:

```
myLabel.addEventListener("stateChange", this, "onStateChange");
function onStateChange(event:Object) {
      if (event.state == "default") {
      // ここに何かを追加する
      }
}
```

2.1.3.6 ヒントとコツ

Label コンポーネントに HTML テキストを表示します。標準の textField でサポートされる HTML タグについては、Flash 8 のマニュアルを参照してください:

```
lbl.htmlText = "<b>foo</b>bar";
```

2.1.4 TextInput

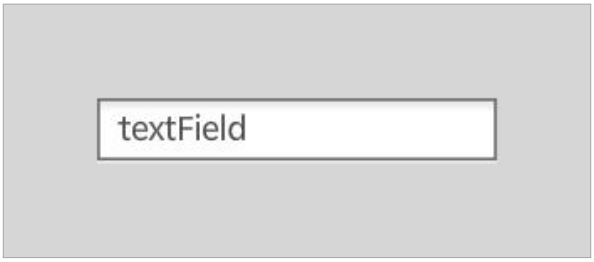

図 14: スキニング前の TextInput

TextInput (gfx.controls.TextInput) は、テキストによるユーザー入力の取得に使用される編集可能な textField コンポーネントです。Label と同様に、このコンポーネントは標準の textField のラッパーに過ぎないので、パスワード モード、最大文字数、HTML テキストなど textField と同じ機能をサポートします。このコンポーネント自体はこれらのプロパティのほんの一部しか公開していませんが、残りは直接 TextInput の textField インスタンスにアクセスすることで変更することができます。

TextInput コンポーネントは入力のために使用します。編集不可のテキストは Label を使って表示できるからです。Label と同様に、開発者は必要に応じて TextInput コンポーネントの代わりに標準の textField を使用することができます。ただし、特に PC アプリケーション向けの高度な UI を開発している場合、この TextInput コンポーネントは標準の textField に勝る価値のある拡張機能を提供します。

第一に、TextInput は focused と disabled ステートをサポートします。これらのステートは標準の textField では簡単に実現できません。フォーカス ステートが分けられているために、TextInput はカスタムのフォーカス インジケータをサポートすることができます。これは標準の textField には含まれていません。標準の textField の表示スタイルを変更するには複雑な AS2 コードが必要ですが、TextInput の表示スタイルはタイムライン上で簡単に構成することができます。TextInput の検証可能プロパティは、Flash Studio に詳しくないデザイナーやプログラマに簡単なワークフローを提供します。また、開発者は TextInput が発生させるイベントを簡単にリッスンして、カスタムの動作を作成することもできます。

TextInput は複数段落の HTML フォーマット テキストも含め、textField が提供する標準の選択、さらにカット、コピー、ペースト機能もサポートします。デフォルトでは、キーボードコマンドは選択 (Shift+矢印キー)、カット (Shift+Delete キー)、コピー (Ctrl+Insert キー)、ペースト (Shift+Insert) です。

TextInput は、マウスカーソルのロールオーバーとロールアウト イベントのハイライト表示をサポート可能です。特別な actAsButton inspectable（検証可能な）プロパティが、このモードをサポートします。このモードは、これら二つのマウスイベントを再生する 2 個の追加

キーフレームをサポートします。ここでの二つのキーフレームは、“over”と“out”と名付けます。これらは、それぞれ rollOver と rollOut ステートを表します。actAsButton モードがセットされても、“over”または“out”フレームが存在していない場合には、TextInput は“default”のキーフレームをどちらのイベントでも再生します。これらのフレームは、CLIK に用意してある TextInput コンポーネントには、デフォルトで存在していない、というポイントに留意してください。開発者の方が必要であれば、ご自分で追加していただくことになります。

このコンポーネントは、また、全く値がセットされていない場合、あるいは、ユーザーが値を入力していない場合に、表示されるデフォルトのテキストをサポートしています。デフォルト Text プロパティは、いかなる String にもセット可能です。デフォルトテキストの色とスタイルのテーマは、明灰色（0xAAAAAA）で、イタリックです。新たな Text フォーマット オブジェクトを、デフォルト TextFormat プロパティにアサインすることで、カスタムのスタイル設定を行います。

2.1.4.1 ユーザーの操作

TextInput をクリックするとこのコンポーネントにフォーカスが適用され、次にカーソルが textField に表示されます。カーソルが表示されると、ユーザーはキーボードや同等のコントローラで文字を入力することができます。左右の矢印キーを押すとカーソルが該当する方向に移動します。すでにカーソルが textField の左端にあるときに左の矢印キーが使用された場合、フォーカスは左側のコントロールに移動します。右矢印の場合も同じです。

2.1.4.2 コンポーネントのセットアップ

CLIK TextInput クラスを使用する MovieClip には、以下の指定されたサブエレメントが必須です。オプションのエレメントである場合には、その旨表記されています:

- ***textField*** *:* textField タイプ

2.1.4.3 ステート

CLIK TextInput コンポーネントは、その focused と disabled プロパティに基づいて、3 つのステートをサポートします:

- *default* または有効のステート
- *focused* ステート、通常、textField の周囲の強調表示された境界線で表現されます。
- *disabled* ステート

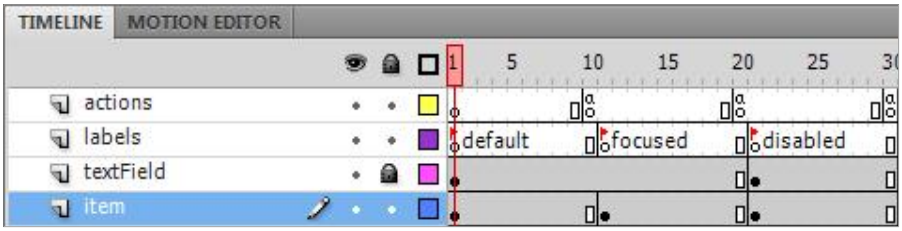

図 15: TextInput のタイムライン

2.1.4.4 検証可能なプロパティ

TextInput コンポーネントの検証可能なプロパティは以下のとおりです:

Text	textField のテキストを設定します。
Visible	false に設定した場合、このコンポーネントを非表示にします。
disabled	true に設定した場合、コンポーネントを無効にします。
editable	false に設定した場合、TextInput が編集不可になります。
maxChars	ゼロより大きな数値は、textField に入力できる文字数を制限します。
password	true の場合、実際の文字の代わりに '*' を表示するように textField を設定します。textField の値はユーザーが入力した実際の文字になり、text プロパティを返します。
defaultText	textField が空の場合に表示されるテキスト。このテキストはデフォルトの defaultTextFormat オブジェクトによってフォーマットされます。デフォルトの設定は、明灰色とイタリックです。
actAsButton	true の場合、TextInput は、フォーカスされていなければ、Button と同様に動作します。rollOver と rollOut ステートをサポートします。マウス クリックまたは、タブキーの押下によって、フォーカスされると、TextInput は、フォーカスを失うまで、ノーマルの状態に戻ります。

2.1.4.5 イベント

すべてのイベント コールバックは、そのイベントの関連情報を含む 1 つの Object パラメータを受け取ります。以下のプロパティは、すべてのイベントに共通です。

- ***type*** *:* イベントのタイプ
- ***target :*** イベントを作成したターゲット

TextInput コンポーネントが作成するイベントは以下の表のとおりです。イベントの隣に表示されているプロパティは、共通プロパティに追加して提供されています。

Show	コンポーネントの visible プロパティが、実行時に true に設定されました。
Hide	コンポーネントの visible プロパティが、実行時に false に設定され

	ました。
focusIn	このコンポーネントにフォーカスが、適用されました。
focusOut	このコンポーネントのフォーカスが、解除されました。
textChange	textField の内容が、変更されました。
rollOver	フォーカスされていない時、マウスカーソルが、コンポーネントにロールオーバーしました。actAsButton プロパティが設定されている時だけ発動されます。 *mouseIndex:* このマウスカーソルのインデックスは、イベントを発生させるときに使われます（マルチ・マウスカーソル環境でのみ有効です）。数値型です。値は、0 から 3。
rollOut	フォーカスされていない時、マウスカーソルが、コンポーネントにロールアウトしました。actAsButton プロパティが設定されている時だけ発動されます。 *mouseIndex:* このマウスカーソルのインデックスは、イベントを発生させるときに使われます（マルチ・マウスカーソル環境でのみ有効です）。数値型です。値は、0 から 3。

以下のコード サンプルは textField の内容への変更をリッスンする方法を示しています:

```
myTextInput.addEventListener("textChange", this, "onTextChange");
function onTextChange(event:Object) {
      // ここに何かを追加する
}
```

2.1.4.6 ヒントとコツ

フォーカスが適用されたときの、自動テキスト選択を停止する:

```
_global.gfxExtensions = true;
textinput.textField.noAutoSelection = true;
```

TextInput コンポーネントに HTML テキストを表示します。標準の textField でサポートされる HTML タグについては、Flash 8 のマニュアルを参照してください:

```
textinput.htmlText = "<b>foo</b>bar";
```

2.1.5 TextArea

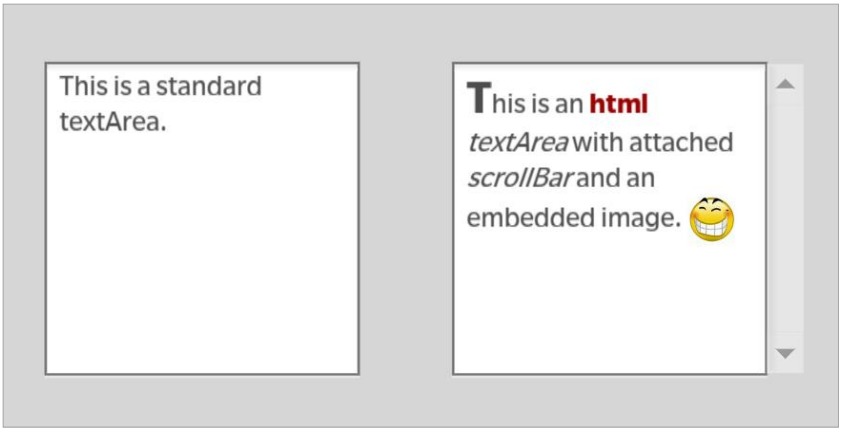

図 16: スキニング前の TextArea

TextArea (gfx.controls.TextArea) は CLIK TextInput から派生したものであり、同じ機能、プロパティ、ステートを共有していますが、オプションとして、複数行の編集可能で、スクロール可能なテキスト入力のための ScrollBar を備えています。 TextInput と TextArea で共有される特別な機能について、TextInput の項での記述を参考にしてください。

Label や TextInput と同様に、TextArea は標準の複数行 textField のラッパーでもあるので、textField のプロパティや、HTML テキスト、テキストの折り返し、選択、カット、コピー、ペーストなどの動作もサポートします。開発者は自由に TextArea を標準の textField に置き換えることができますが、拡張機能、ステート、検証可能なプロパティ、イベント等を考慮すると、このコンポーネントを使用することを強くお勧めします。

標準の textField は ScrollIndicator や ScrollBar に組み合わせることができますが、TextArea はこれらのコンポーネントとより密接につながります。標準の textField とは異なり、編集不可であっても、フォーカスが適用されたときに TextArea はキーボードや同等のコントローラを使ってスクロールすることができます。スクロール関連のコンポーネントはフォーカスが適用できないので、TextArea は、自分自身とフォーカス ステートのスクロール コンポーネントの両方を包む、さらに洗練されたフォーカスが適用されたグラフィック ステートを表示することができます。

2.1.5.1 ユーザーの操作

TextArea をクリックするとこのコンポーネントにフォーカスが適用され、次にカーソルが textField に表示されます。カーソルが表示されると、ユーザーはキーボードや同等のコントローラで文字を入力することができます。4 つの矢印キーを押すとカーソルが該当する方向に

移動します。カーソルがすでに端にあるときに矢印キーが使用された場合、フォーカスはその矢印キーの方向の隣接するコントロールに移動します。

2.1.5.2 コンポーネントのセットアップ

CLIK TextArea クラスを使用する MovieClip には、以下の指定されたサブエレメントが必須です。オプションのエレメントであれば、その旨表記されています:

- *textField :* TextField タイプ

2.1.5.3 ステート

その親コンポーネントである TextInput と同様に、TextArea コンポーネントは、その focused と disabled プロパティに基づいて、3 つのステートをサポートします:

- ***default*** または有効のステート
- ***focused*** ステート、通常、textField の周囲の強調表示された境界線で表現されます。
- ***disabled*** ステート

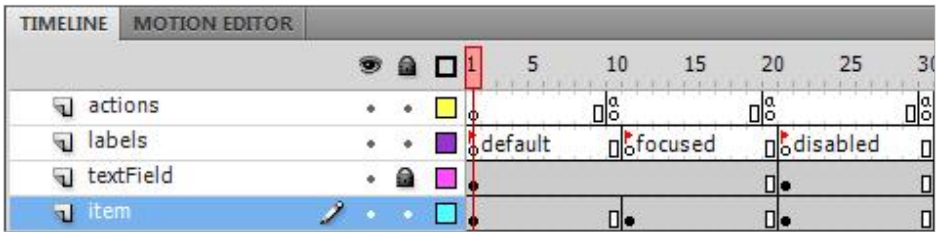

図 17: TextArea のタイムライン

2.1.5.4 検証可能なプロパティ

TextArea コンポーネントの検証可能なプロパティは TextInput に似ていますが、若干の追加があり password プロパティは含まれていません。追加されたプロパティは CLIK ScrollBar コンポーネントに関係します。これは 2.4 章で説明します:

Text	textField のテキストを設定します。
Visible	false に設定した場合、このコンポーネントを非表示にします。
disabled	true に設定した場合、コンポーネントを無効にします。
editable	false に設定した場合、TextInput が編集不可になります。
maxChars	ゼロより大きな数値は、textField に入力できる文字数を制限します。
scrollBar	CLIK ScrollBar コンポーネントが使用するインスタンス名、または ScrollBar シンボルへのリンケージ ID (この場合インスタンスは TextArea が作成します) です。

scrollPolicy	"auto" に設定された場合、スクロール バーはスクロールする十分なテキストがある場合に限って表示されます。"on" に設定されると ScrollBar は常に表示され、"off" に設定されると表示されません。このプロパティは ScrollBar が割り当てられている場合に限って、そのコンポーネントに影響します (ScrollBar プロパティを参照してください)。
defaultText	テキストフィールドが空の時に表示されるテキストです。デフォルトの defaultTextFormat オブジェクトによってフォーマットされています。デフォルト設定は、明灰色とイタリックです。
actAsButton	true の場合、TextInput は、フォーカスされていなければ、Button と同様に動作します。rollOver と rollOut ステートをサポートします。マウスのクリック、あるいはタブボタンの押下によってフォーカスされると、TextArea は、フォーカスが失われるまでの間、標準のモードに戻ります。

2.1.5.5 イベント

すべてのイベント コールバックは、そのイベントの関連情報を含む 1 つの Object パラメータを受け取ります。以下のプロパティは、すべてのイベントに共通です。

- ***type :*** イベントのタイプ
- ***target :*** イベントを作成したターゲット

TextArea コンポーネントが作成するイベントは以下の表のとおりです。イベントの隣に表示されているプロパティは、共通プロパティに追加して提供されています。

Show	コンポーネントの visible プロパティが、実行時に true に設定されました。
Hide	コンポーネントの visible プロパティが、実行時に false に設定されました。
focusIn	このコンポーネントにフォーカスが、適用されました。
focusOut	このコンポーネントのフォーカスが、解除されました。
textChange	textField の内容が、変更されました。
Scroll	テキスト領域が、スクロールされました。
rollOver	フォーカスされていない時、マウスカーソルが、コンポーネントにロールオーバーしました。actAsButton プロパティが設定されている時だけ発動されます。 *mouseIndex:* このマウスカーソルのインデックスは、イベントを発生させるときに使われます（マルチ・マウスカーソル環境でのみ有効です）。数値型です。値は、0 から 3。
rollOut	フォーカスされていない時、マウスカーソルが、コンポーネントにロールアウトしました。actAsButton プロパティが設定されている時だけ発動されます。 *mouseIndex:* このマウスカーソルのインデックスは、イベントを発生させるときに使われます（マルチ・マウスカーソル環境でのみ有効

です）。数値型です。値は、0 から 3。

以下の例は TextArea のスクロール イベントをリッスンする方法を示しています:

```
myTextArea.addEventListener("scroll", this, "onTextScroll");
function onTextScroll(event:Object) {
    // ここに何かを追加する
}
```

2.1.5.6 ヒントとコツ

フォーカスが適用されたときの、自動テキスト選択を停止する:

```
_global.gfxExtensions = true;
textinput.textField.noAutoSelection = true;
```

TextArea コンポーネントに HTML テキストを表示します。標準の textField でサポートされる HTML タグについては、Flash 8 のマニュアルを参照してください:

```
textinput.htmlText = "<b>foo</b>bar";
```

2.2 グループ タイプ

グループ タイプには、RadioButton、ButtonGroup、そして ButtonBar コンポーネントが含まれています。ButtonGroup は、Button コンポーネントのグループを管理する特殊なロジックを備えたマネージャ タイプです。これは視覚的な表現を持たず、ステージ上には存在しません。ただし、ButtonBar はステージ上にあり、Button のグループの管理も行います。RadioButton は自動的にその兄弟と ButtonGroup にグループ化される特殊な Button です。

図 18: *Dawn of War II* - ラジオ ボタン グループの例

2.2.1 RadioButton

図 19: スキニング前の RadioButton

RadioButton (gfx.controls.RadioButton) は、通常 1 つの値を表示し変更するセットに属するボタン コンポーネントです。セット内の RadioButton は 1 つだけしか選択できず、そのセットの別の RadioButton をクリックするとその新規のコンポーネントが選択され、前回選択されていたコンポーネントの選択は解除されます。

CLIK RadioButton は CheckBox コンポーネントに非常に良く似ており、その機能、ステート、動作を共有します。大きな違いは RadioButton がグループ プロパティをサポートすることです。そのプロパティにカスタムの ButtonGroup (次の章を参照してください) を割り当てることができます。また、RadioButton は toggle プロパティを設定しません。切り替え (トグル) はそれを管理する ButtonGroup インスタンスが行うからです。

2.2.1.1 ユーザーの操作

マウス、または同等のコントロールを使って RadioButton コンポーネントをクリックすると、まだ選択されていなければ、連続してそのコンポーネントを選択します。RadioButton が選択されていて、たった今クリックされた別の RadioButton と同じ ButtonGroup に属している場合、最初の RadioButton が選択解除されます。それ以外の点では、この RadioButton は Button と同じように動作します。

2.2.1.2 コンポーネントのセットアップ

CLIK RadioButton クラスを使用する MovieClip には、以下の指定されたサブエレメントが必須です。オプションのエレメントであれば、その旨表記されています:

- ***textField*** *:* (オプション) TextField タイプ。ボタンのラベルです。
- *focusIndicator :* (オプション) MovieClip タイプ。フォーカスが適用されたステートの表示に使用される別の MovieClip です。存在する場合、この MovieClip は "show" と "hide" の 2 つの指定されたフレームを備えている必要があります。

2.2.1.3 ステート

RadioButton は選択したステートと選択解除されたステートを切り替えることができるので、CheckBox と同様に、最低でも以下のステートが必要となります:

- *up* またはデフォルトのステート
- マウス カーソルがコンポーネント上にあるとき、またはフォーカスが適用されているときの *over* ステート
- ボタンが押下されたときの *down* ステート
- *disabled* ステート
- *selected_up* またはデフォルトのステート
- マウス カーソルがコンポーネント上にあるとき、またはフォーカスが適用されているときの *selected_over* ステート
- ボタンが押下されたときの *selected_down* ステート
- *selected_disabled* ステート

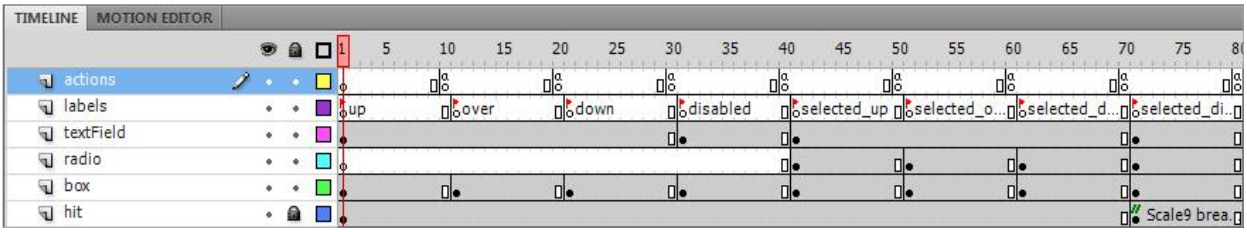

図 20: RadioButton のタイムライン

これは RadioButton に必要な最小限のキーフレーム セットです。Button コンポーネントがサポートするステートとキーフレームの拡張セット、それに伴う RadioButton コンポーネントについては、「Getting Started with CLIK Buttons」を参照してください。

2.2.1.4 検証可能なプロパティ

RadioButton は Button コントロールから派生したものなので、このコンポーネントには Button と同じ検証可能なプロパティが含まれていますが、toggle と disableFocus プロパティは除外されます。

Label	Button のラベルを設定します。
Visible	false に設定した場合、ボタンを非表示にします。
disabled	true に設定した場合、ボタンを無効にします。
disableConstraints	この Button コンポーネントには、コンポーネントのサイズを変更したときに、ボタン内部の textField の配置やスケーリングを決定する constraints (制約) オブジェクトが含まれています。このプロパティを true に設定すると、constraints オブジェクトは無効になります。タイムライン アニメーション

	でボタンの textField のサイズ変更や再配置の必要性がある場合、このプロパティは特に便利です。さらに、この Button コンポーネントはサイズが変わりません。無効にしない場合、その textField は有効期間中ずっとそのデフォルトの値に移動しスケーリングされるので、Button のタイムラインに作成された可能性のある textField トランジション/スケーリング トゥイーンを無効にします。
selected	true に設定されているときに、RadioButton をオンに (または選択) します。
Data	RadioButton のラベルとは別のデータとして、コンポーネントに追加できるカスタムの文字列です。
group	存在する、または RadioButton が自動的に作成するはずの ButtonGroup インスタンスの名前です。RadioButton が作成した場合、この新規の ButtonGroup はその RadioButton のコンテナ内に存在します。たとえば、RadioButton が*_root* に存在する場合、その ButtonGroup オブジェクトは*_root* に作成されます。同じグループを使用するすべての RadioButton は 1 つのセットに属します。

2.2.1.5 イベント

すべてのイベント コールバックは、そのイベントの関連情報を含む 1 つの Object パラメータを受け取ります。以下のプロパティは、すべてのイベントに共通です。

- ***type*** *:* イベントのタイプ
- ***target :*** イベントを作成したターゲット

RadioButton コンポーネントが作成するイベントは以下の表のとおりです。イベントの隣に表示されているプロパティは、共通プロパティに追加して提供されています。

Show	コンポーネントの visible プロパティが、実行時に true に設定されました。
Hide	コンポーネントの visible プロパティが、実行時に false に設定されました。
focusIn	このコンポーネントにフォーカスが、適用されました。
focusOut	このコンポーネントのフォーカスが、解除されました。
select	コンポーネントの selected プロパティが、変更されました。 *selected :* Button の selected プロパティです。ブール型です。
stateChange	ボタンのステートが、変更されました。 *state :* Button の新規ステートです。文字列タイプで、値は "up"、"over"、"down" などです。ステートの全リストについては、「Getting Started with CLIK Buttons」を参照してください。
rollOver	マウス カーソルが、ボタン上に移動しました。 *mouseIndex :*このイベントの作成に使用されたマウス カーソルの

	インデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
rollOut	マウス カーソルが、ボタンから離れました。 *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
press	ボタンが、押下されました。 *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
doubleClick	ボタンが、ダブルクリックされました。*doubleClickEnabled* プロパティが true の場合に限って発生します。 *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
click	ボタンが、クリックされました。 *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
dragOver	マウス カーソルが (マウスの左ボタンを押したままの状態で)、 ボタン上にドラッグされました。 *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
dragOut	マウス カーソルが(マウスの左ボタンを押したままの状態で)、 ボタンから離れました。 *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
releaseOutside	マウス カーソルが、ボタンから離れ、マウスの左ボタンが解放されました。 *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。

以下の例は RadioButton のトグルを処理する方法を示しています:

```
myRadio.addEventListener("select", this, "onRadioToggle");
function onRadioToggle(event:Object) {
      // ここに何かを追加する
}
```

2.2.1.6 ヒントとコツ

同じレベルにはない RadioButton をグループ化する:

```
import gfx.controls.ButtonGroup;
var myGroup:ButtonGroup = new ButtonGroup("groupName");
radio1.group = myGroup;
radio2.group = myGroup;
someMovie.radio1.group = myGroup;
someMovie.radio2.group = myGroup;
```

2.2.2 ButtonGroup

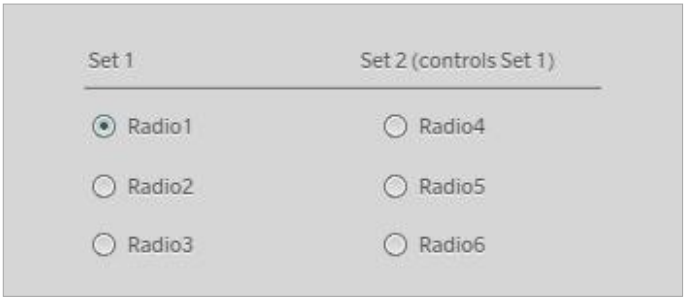

図 21: スキニング前の ButtonGroup

CLIK ButtonGroup (gfx.controls.ButtonGroup) はそれ自体はコンポーネントではありませんが、それでも重要です。ボタン セットの管理に使用されるからです。ButtonGroup はセット内のボタンを 1 つだけ選択させ、残りは未選択であることを保証します。ユーザーがセット内の別のボタンを選択した場合、現在選択されているボタンは選択解除されます。CLIK Button コンポーネント (CheckBox や RadioButton) から派生するコンポーネントはすべて、ButtonGroup インスタンスに割り当てることができます。

2.2.2.1 ユーザーの操作

ButtonGroup にはユーザーの操作はありません。視覚的な表示を持たないからです。ただし、そのグループに属する RadioButton がクリックされたときに、間接的な影響は受けます。

2.2.2.2 コンポーネントのセットアップ

CLIK ButtonGroup クラスを使用する MovieClip にはサブエレメントはいっさい必要ありません。視覚的な表示を持たないからです。

2.2.2.3 ステート

ButtonGroup はステージ上の視覚的な表示は備えていません。したがって、割り当てられるステートもありません。

2.2.2.4 検証可能なプロパティ

ButtonGroup はステージ上の視覚的な表示は備えていません。したがって、検証可能なプロパティは使用できません。

2.2.2.5 イベント

すべてのイベント コールバックは、そのイベントの関連情報を含む 1 つの Object パラメータを受け取ります。以下のプロパティは、すべてのイベントに共通です。

- ***type*** *：イベントのタイプ*
- *target：イベントを作成したターゲット*

ButtonGroup コンポーネントが作成するイベントは以下の表のとおりです。イベントの隣に表示されているプロパティは、共通プロパティに追加して提供されています。

change	グループの新規のボタンが、選択されました。 • *item：*選択されたボタンです。CLIK Button のタイプです。 • *data：*選択されたボタンのデータ値です。AS2 の Object タイプです。
itemClick	グループ内のボタンが、クリックされました。 • *item：*クリックされたボタンです。CLIK Button のタイプです。

以下の例は、ButtonGroup のどのボタンが選択されたかを判断する方法を示しています:

```
myGroup.addEventListener("change", this, "onNewSelection");
function onNewSelection(event:Object) {
      if (event.item == myRadio) {
      //ここに何かを追加する
      }
}
```

2.2.2.6 ヒントとコツ

グループの現在選択されている RadioButton を探す:

```
var selectedRadio:Button = group.selectedButton;
```

ステージ上の RadioButton セットに割り当てられたデフォルトの ButtonGroup に、リスナーをインストールする:

```
radioBtn1.group.addEventListener("itemClick", this, "onClick");
```

2.2.3 ButtonBar

図 22: スキニング前の ButtonBar

ButtonBar (gfx.controls.ButtonBar) は ButtonGroup に似ていますが、視覚的な表示を備えています。また、dataProvider に基づいて、オンザフライで Button インスタンスを作成することもできます (dataProvider については、「プログラミングの詳細」を参照してください)。ButtonBar は、ダイナミックなタブバーのような UI エレメントの作成に便利です。このコンポーネントは dataProvider の割り当てによって設定されます:

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

2.2.3.1 ユーザーの操作

ButtonBar は ButtonGroup と同じように動作します。ユーザーは直接そのバーを操作することはできませんが、ButtonBar は自らが管理する Button がクリックされたときに、間接的に影響も受けます。

2.2.3.2 コンポーネントのセットアップ

CLIK ButtonBar クラスを使用する MovieClip にはサブエレメントはいっさい必要ありません。実行時にサブエレメントを作成するからです。

2.2.3.3 ステート

CLIK ButtonBar は視覚的なステートは備えていません。自らが管理する Button コンポーネントが、グループ ステートの表示に使用されるからです。

2.2.3.4 検証可能なプロパティ

ButtonBar コンポーネントにはコンテンツはありませんが (Flash IDE のステージ上では単に小さな丸として表示されます)、複数の検証可能なプロパティが含まれています。そのプロパティの大部分は、ButtonBar が作成した Button インスタンスの配置設定を処理します。

visible	false に設定した場合、ButtonBar を非表示にします。
disabled	true に設定した場合、ButtonBar を無効にします。
itemRenderer	Button コンポーネント シンボルのリンケージ ID です。このシンボルは ButtonBar に割り当てられたデータに基づいて、必要に応じてインスタンス化されます。
direction	Button の配置です。horizontal は Button インスタンスを横に並べて配置しますが、vertical は上下に配置します。
spacing	Button インスタンス間のスペースです。現在設定されている方向 (*direction* プロパティを参照してください) だけに影響します。
autoSize	true に設定すると、Button インスタンスは表示されているラベルに合わせてサイズ変更されます。
buttonWidth	すべての Button インスタンスに共通の幅を設定します。autoSize が true に設定されている場合、このプロパティは無視されます。

2.2.3.5 イベント

すべてのイベント コールバックは、そのイベントの関連情報を含む 1 つの Object パラメータを受け取ります。以下のプロパティは、すべてのイベントに共通です。

- *type :* イベントのタイプ
- *target :* イベントを作成したターゲット

ButtonBar コンポーネントが作成するイベントは以下の表のとおりです。イベントの隣に表示されているプロパティは、共通プロパティに追加して提供されています。

show	コンポーネントの visible プロパティが、実行時に true に設定されました。
hide	コンポーネントの visible プロパティが、実行時に false に設定されました。
focusIn	このコンポーネントにフォーカスが、適用されました。
focusOut	このコンポーネントのフォーカスが、解除されました。
change	グループの新規のボタンが、選択されました。 • *index :* ButtonBar の選択されたインデックスです。数値型で、値は 0 からボタン数から 1 を引いたものまでです。

	• *item：*選択された Button です。CLIK Button のタイプです。 • *item：*dataProvider の選択されたアイテムです。AS2 の Object タイプです。 • *data：*選択された dataProvider アイテムのデータ値です。AS2 の Object タイプです。
itemClick	グループ内のボタンが、クリックされました。 • *index：*クリックされた Button の ButtonBar インデックスです。数値型で、値は 0 からボタン数から 1 を引いたものまでです。 • *item：*dataProvider の選択されたアイテムです。AS2 の Object タイプです。 • *data：*選択された dataProvider アイテムのデータ値です。AS2 の Object タイプです。 • *mouseIndex：*このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。

以下の例は、ButtonBar 内部で Button インスタンスがクリックされたときを検出する方法を示しています:

```
myBar.addEventListener("itemClick", this, "onItemClick");
function onItemClick(event:Object) {
      processData(event.data);
      // ここに何かを追加する
}
```

2.3 スクロール タイプ

スクロール タイプには ScrollIndicator、ScrollBar、Slider コンポーネントが含まれます。ScrollIndicator は非インタラクティブで、単にサムを使ってターゲット コンポーネントのスクロール インデックスを表示するだけですが、ScrollBar はスクロールの位置に影響するユーザーの操作を許可します。ScrollBar は 4 つの Button (サムまたはツマミ、トラック、上下の矢印) で構成されています。Slider は ScrollBar に似ていますが、インタラクティブなサムとトラックしか含んでいません。また、ターゲット コンポーネント内のエレメント数に基づいたサムのサイズ変更は行いません。 Slider は ScrollBar に似ていますが、インタラクティブなサムとトラックしか含んでいません。また、ターゲット コンポーネント内のエレメント数に基づいたサムのサイズ変更は行いません。

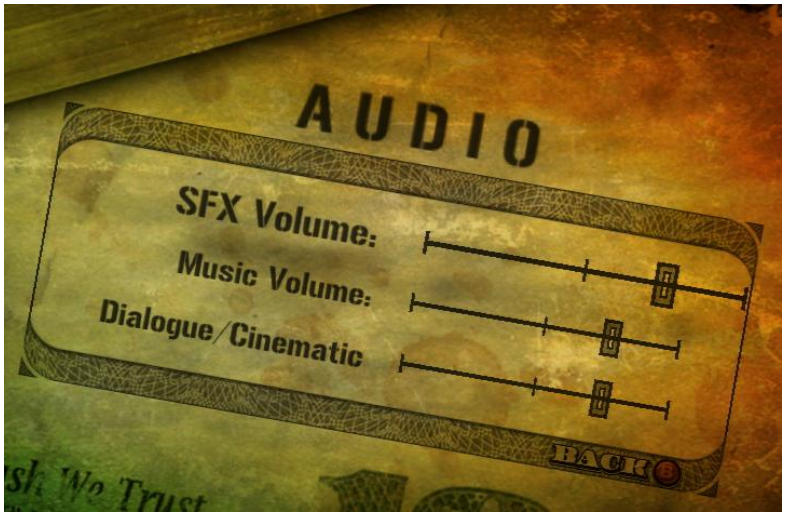

図 23: *Mercenaries 2* - オーディオ メニュー スライダの例

2.3.1 ScrollIndicator

図 24: スキニング前の ScrollIndicator

CLIK ScrollIndicator (gfx.controls.ScrollIndicator) は、複数行の textField など、別のコンポーネントのスクロール位置を表示します。ScrollIndicator は自動的にスクロール位置を指

すように、textField でポイントすることができます。すべてのリストベースのコンポーネントは、TextArea と同様に、ScrollIndicator や ScrollBar (次の章を参照してください) インスタンス、あるいはリンケージ ID を指す scrollBar プロパティを備えています。

2.3.1.1 ユーザーの操作

ScrollIndicator にはユーザーの操作はありません。表示のためだけに使用されます。

2.3.1.2 コンポーネントのセットアップ

CLIK ScrollIndicator クラスを使用する MovieClip には、以下の指定されたサブエレメントが必須です。オプションのエレメントであれば、その旨表記されています:

- *thumb :* CLIK Button のタイプ。スクロール インジケータのツマミです。
- *track :* MovieClip タイプ。スクロール インジケータのトラックです。このトラックの範囲は、ツマミが移動できる領域を決定します。

2.3.1.3 ステート

ScrollIndicator は明確なステートは備えていません。その子エレメントである thumb（サム）と track（トラック）の Button コンポーネントのステートを使用します。

図 25: ScrollIndicator のタイムライン

2.3.1.4 検証可能なプロパティ

ScrollIndicator の検証可能なプロパティは以下のとおりです:

scrollTarget	TextArea、または標準の複数行の textField をスクロール ターゲットとして設定し、自動的にスクロールのイベントに応答します。非 textField タイプは手動で ScrollIndicator のプロパティを更新する必要があります。
visible	false に設定した場合、このコンポーネントを非表示にします。
disabled	true に設定した場合、コンポーネントを無効にします。
offsetTop	サム（thumb）を上端でオフセットします。正数値で、サムの最上端の位置を高くします。

offsetBottom	サム（thumb）を下端でオフセットします。正数値で、サムの最下端の位置を下げます。

2.3.1.5 イベント

すべてのイベント コールバックは、そのイベントの関連情報を含む 1 つの Object パラメータを受け取ります。以下のプロパティは、すべてのイベントに共通です。

- *type*：イベントのタイプ
- *target*：イベントを作成したターゲット

ScrollIndicator コンポーネントが作成するイベントは以下の表のとおりです。イベントの隣に表示されているプロパティは、共通プロパティに追加して提供されています。

show	コンポーネントの visible プロパティが、実行時に true に設定されました。
hide	コンポーネントの visible プロパティが、実行時に false に設定されました。
scroll	スクロールの位置が、変更されました。 • *position*：新規のスクロール位置です。数値型で、値は最も低い位置から最も高い位置までです。

以下の例はスクロール イベントをリッスンする方法を示しています:

```
mySI.addEventListener("scroll", this, "onTextScroll");
function onTextScroll(event:Object) {
       trace("scroll position: " + event.position);
      // ここに何かを追加する
}
```

2.3.1.6 ヒントとコツ

スタンドアローン スクロール インジケータ インスタンスを手動でセットアップする:

```
scrollInd.setScrollProperties(pageSize, minPos, maxPos, pageScrollSz);
scrollInd.positon = currPos;
```

スクロールの方向を設定します。これは_rotation プロパティを使ってステージ上に作成されたコンポーネントに、自動的に設定されます。ScrollIndicator コンポーネントが回転されず、デフォルトで水平方向になっている場合、この値は明確に設定する必要があります:

```
scrollInd.direction = "horizontal";
```

2.3.2 ScrollBar

図 26: スキニング前の ScrollBar

CLIK ScrollBar (gfx.controls.ScrollBar) は、別のコンポーネントのスクロール位置を表示しコントロールします。ドラッグ可能なサム ボタンと、オプションの上下の矢印ボタン、さらにクリック可能なトラックと共に ScrollIndicator にインタラクティビティを追加します。

2.3.2.1 ユーザーの操作

ScrollBar のサムはマウスや同等のコントロールでつかみ、ScrollBar トラックの範囲をドラッグします。マウス カーソルが ScrollBar の上にあるときにマウス ホイールを動かすと、スクロール操作を行うことができます。上向き矢印をクリックするとサムは上に移動し、下向き矢印をクリックすると下に移動します。トラックのクリックには 2 つの動作があります。1 つはサムが継続してクリックした地点に向かってスクロールすることで、もう 1 つは直ちにその地点にジャンプして、ドラッグするように設定されることです。このトラック モードは *trackMode* inspectable（検証可能な）プロパティが設定します (検証可能なプロパティの章を参照してください)。trackMode の設定に関係なく、Shift キーを押しながらトラックをクリックすると、サムは直ちにそのカーソルまで移動して、ドラッグするように設定されます。

2.3.2.2 コンポーネントのセットアップ

CLIK ScrollBar クラスを使用する MovieClip には、以下の指定されたサブエレメントが必須です。オプションのエレメントであれば、表記されています:

- *upArrow :* CLIK Button のタイプです。スクロール アップする Button で、通常はスクロールバーの先頭に置かれます。
- *downArrow :* CLIK Button のタイプです。スクロール ダウンする Button で、通常はスクロールバーの最後に置かれます。
- *thumb :* CLIK Button のタイプ。スクロールバーのツマミです。
- *track :* CLIK Button のタイプです。ツマミの移動範囲を決める境界を持つスクロールバーのトラックです。

2.3.2.3 ステート

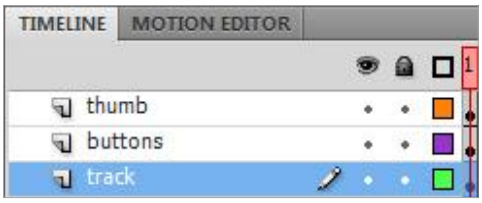

図 27: ScrollBar のタイムライン

ScrollIndicator と同様に、ScrollBar は明確なステートは備えていません。その子エレメントである thumb、up、down、track の Button コンポーネントのステートを使用します。

2.3.2.4 検証可能なプロパティ

ScrollBar の検証可能なプロパティは ScrollIndicator に似ていますが、1 つ追加があります:

scrollTarget	TextArea、または標準の複数行の textField をスクロール ターゲットとして設定し、自動的にスクロールのイベントに応答します。非 textField タイプは手動で ScrollIndicator のプロパティを更新する必要があります。
trackMode	ユーザーがカーソルでトラックをクリックすると、カーソルが解放されるまで、scrollPage 設定によってサムはページ単位で継続してスクロールされます。scrollToCursor 設定によりサムは直ちにカーソルにジャンプし、さらにカーソルが解放されるまで、サムをドラッグ モードに移行します。
visible	false に設定した場合、このコンポーネントを非表示にします。
disabled	true に設定した場合、コンポーネントを無効にします。
offsetTop	サム（thumb）を上端でオフセットします。正数値で、サムの最上端の位置を高くします。
offsetBottom	サム（thumb）を下端でオフセットします。正数値で、サムの最下端の位置を下げます。

2.3.2.5 イベント

すべてのイベント コールバックは、そのイベントの関連情報を含む 1 つの Object パラメータを受け取ります。以下のプロパティは、すべてのイベントに共通です。

- *type :* イベントのタイプ
- *target :* イベントを作成したターゲット

ScrollBar コンポーネントが作成するイベントは以下の表のとおりです。イベントの隣に表示されているプロパティは、共通プロパティに追加して提供されています。

show	コンポーネントの visible プロパティが、実行時に true に設定されました。
hide	コンポーネントの visible プロパティが、実行時に false に設定されました。

scroll	スクロールの位置が、変更されました。 • *position:* 新規のスクロール位置です。数値型で、値は最も低い位置から最も高い位置までです。

以下の例はスクロール イベントをリッスンする方法を示しています:

```
mySB.addEventListener("scroll", this, "onListScroll");
function onListScroll(event:Object) {
      trace("scroll position: " + event.position);
      // ここに何かを追加する
}
```

2.3.2.6 ヒントとコツ

スタンドアローン スクロール インジケータ インスタンスを手動でセットアップする:

```
scrollBar.setScrollProperties(pageSize, minPos, maxPos, pageScrollSz);
scrollBar.positon = currPos;
```

スクロールの方向を設定します。これは_rotation プロパティを使ってステージ上に作成されたコンポーネントに、自動的に設定されます。ScrollBar コンポーネントが回転されず、デフォルトで水平方向になっている場合、この値は明確に設定する必要があります:

```
scrollBar.direction = "horizontal";
```

トラックが *scrollPage* モードで押されたときに、スクロールされる位置の数を設定します。デフォルトではこの値は 1 です:

```
scrollBar.trackScrollPageSize = pageSize;
```

2.3.3 Slider

図 28: スキニング前の Slider

Slider（gfx.controls.Slider）は範囲内の数値を表示します。その数値を表現するサムを含み、ドラッグによって変更します。

2.3.3.1 ユーザーの操作

Slider のサムはマウス等のコントロールで、Slider トラックの範囲をドラッグできます。トラックをクリックすると直ちにカーソルの位置に移動し、ドラッグするように設定されます。フォーカスが適用されている間、左右の矢印はサムを該当する方向に移動しますが、Home キーや End キーはサムをトラックの先頭や最後に移動します。

2.3.3.2 コンポーネントのセットアップ

CLIK Slider クラスを使用する MovieClip には、以下の指定されたサブエレメントが必須です。オプションのエレメントであれば、その旨表記されています:

- *thumb :* CLIK Button のタイプ。スライダのツマミ（サム）です。
- *track :* CLIK Button のタイプです。スライダ トラックの範囲はツマミ（サム）が移動できる領域を決定します。

2.3.3.3 ステート

ScrollIndicator や ScrollBar と同様に、Slider は明確なステートは備えていません。その子エレメントである thumb と track の Button コンポーネントのステートを使用します。

図 29: Slider のタイムライン

2.3.3.4 検証可能なプロパティ

Slider コンポーネントの検証可能なプロパティは以下のとおりです:

visible	false に設定した場合、このコンポーネントを非表示にします。
disabled	true に設定した場合、コンポーネントを無効にします。
value	Slider で表示される数値です。
minimum	Slider の範囲の最小値です。
maximum	Slider の範囲の最大値です。
snapping	true に設定されると、サムは snapInterval の倍数の値にスナップします。
snapInterval	snapInterval はサムがスナップする値の倍数を決定します。snapping が false に設定されている場合は、効果はありません。
liveDragging	true に設定されると、Slider はサムをドラッグしているときに change イベントを作成します。false の場合、Slider はドラッグが終わるまで change イベントを作成しません。
offsetLeft	サム（thumb）を左端でオフセットします。正数値で、サムを内側に移動します。
offsetRight	サム（thumb）を右端でオフセットします。正数値で、サムを内側に移動します。

2.3.3.5 イベント

すべてのイベント コールバックは、そのイベントの関連情報を含む 1 つの Object パラメータを受け取ります。以下のプロパティは、すべてのイベントに共通です。

- *type*：イベントのタイプ
- *target*：イベントを作成したターゲット

Slider コンポーネントが作成するイベントは以下の表のとおりです。イベントの隣に表示されているプロパティは、共通プロパティに追加して提供されています。

Show	コンポーネントの visible プロパティが、実行時に true に設定されました。
Hide	コンポーネントの visible プロパティが、実行時に false に設定されました。
focusIn	このコンポーネントにフォーカスが、適用されました。
focusOut	このコンポーネントのフォーカスが、解除されました。
change	Slider 値が、変更されました。

以下の例は Slider 値の変更をリッスンする方法を示しています:

```
mySlider.addEventListener("change", this, "onValueChange");
function onValueChange(event:Object) {
      trace("slider value: " + event.target.value);
      // ここに何かを追加する
}
```

2.4 リスト タイプ

CLIK のリスト タイプには、NumericStepper、OptionStepper、ScrollingList、TileList、DropdownMenu コンポーネントが含まれます。NumericStepper を除き、これらのコンポーネントはすべて DataProvider と連携して、表示される情報を管理します。また、ListItemRenderer コンポーネントもこのカテゴリーに含まれます。ScrollingList や TileList コンポーネントがこれを使用して、リストの項目を表示するためです。

NumericStepper と OptionStepper は一度に 1 つのエレメントしか表示できませんが、ScrollingList と TileList は複数のエレメントを表示することができます。後者の 2 つのエレメントは ScrollIndicator または ScrollBar コンポーネントのいずれかをサポートすることができます。DropdownMenu コンポーネントは、アイドル状態では 1 つのエレメントしか表示しませんが、展開されると ScrollingList または TileList のいずれかを使ってさらに多くのエレメントを表示します。

図 30: *Mercenaries 2* - スクロール インジケータ付きのスクロール リストの例

図 31: *Halo Wars* -「難易度設定」ドロップダウン メニューの例

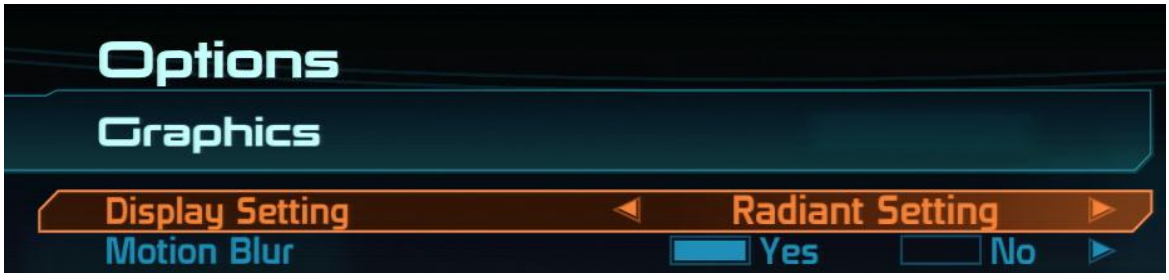

図 32: *Mass Effect* -「表示設定」オプション コントロールの例

2.4.1 NumericStepper

図 33: スキニング前の NumericStepper

NumericStepper (gfx.controls.NumericStepper) は、割り当てられた範囲内の 1 つの数値を表示し、任意の刻み幅単位で、その値を増減する機能をサポートします。

2.4.1.1 ユーザーの操作

NumericStepper には 2 つの矢印ボタンが含まれており、これらのボタンをマウスや同等のコントローラでクリックして、その数値を変更することができます。フォーカスが適用されると、この数値は左右の矢印キーや同等のコントロールを使って、キーボードで変更することもできます。これらのキーは刻み幅単位で現在の値を増減します。Home キーや End キー、または同等のコントロールを押すと、この数値がそれぞれ最小値と最大値に変更されます。

2.4.1.2 コンポーネントのセットアップ

NumericStepper クラスを使用する MovieClip には、以下の指定されたサブエレメントが必須です。オプションのエレメントであれば、その旨表記されています:

- *textField :* TextField タイプ。現在の値の表示に使用されます。
- *nextBtn :* CLIK Button のタイプです。これをクリックすると、刻み幅単位で現在の値を増加させます。
- *prevBtn :* CLIK Button のタイプです。これをクリックすると、刻み幅単位で現在の値を減少させます。

2.4.1.3 ステート

NumericStepper コンポーネントは、その focused と disabled プロパティに基づいて、3 つのステートをサポートします:

- *default* または有効のステート
- *focused* ステート、textField 領域を強調表示します。

- *disabled* ステート

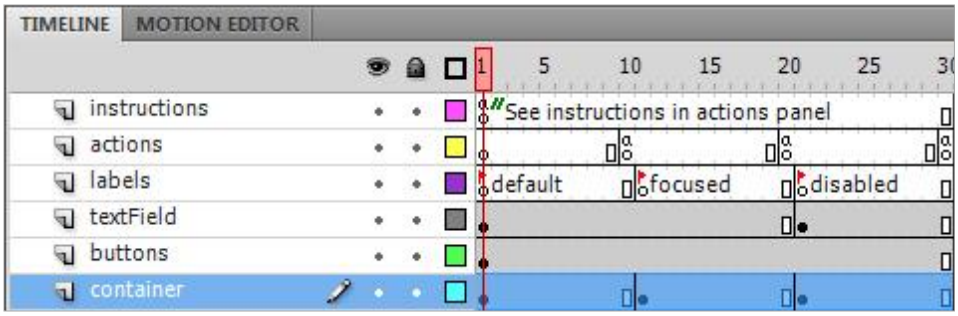

図 34: NumericStepper のタイムライン

2.4.1.4 検証可能なプロパティ

NumericStepper コンポーネントから生じる MovieClip は、以下の検証可能なプロパティを備えています:

visible	false に設定した場合、このコンポーネントを非表示にします。
disabled	true に設定した場合、コンポーネントを無効にします。
value	NumericStepper で表示される数値です。
minimum	NumericStepper の範囲の最小値です。
maximum	NumericStepper の範囲の最大値です。

2.4.1.5 イベント

すべてのイベント コールバックは、そのイベントの関連情報を含む 1 つの Object パラメータを受け取ります。以下のプロパティは、すべてのイベントに共通です。

- *type：*イベントのタイプ
- *targe：*イベントを作成したターゲット

NumericStepper コンポーネントが作成するイベントは以下の表のとおりです。イベントの隣に表示されているプロパティは、共通プロパティに追加して提供されています。

show	コンポーネントの visible プロパティが、実行時に true に設定されました。
hide	コンポーネントの visible プロパティが、実行時に false に設定されました。
change	NumericStepper の値が、変更されました。
stateChange	NumericStepper の focused または disabled プロパティが、変更されています。 • *state：*新規ステートの名前です。文字列タイプで、値は "default"、"focused"、または "disabled" です。

以下の例は NumericStepper 値の変更をリッスンする方法を示しています:

```
myNS.addEventListener("change", this, "onValueChange");
function onValueChange(event:Object) {
	trace("ns value: " + event.target.value);
	// ここに何かを追加する
}
```

2.4.1.6 ヒントとコツ

増減の幅を変更する:

```
ns.stepSize = 0.5;
```

数値にデコレータを追加する。例:

```
ns.labelFunction = function(value:Number) {
	switch(value) {
			case 1:
				return value + "st";
			default:
				return value + "th";
		}
}
```

2.4.2 OptionStepper

図 35: スキニング前の OptionStepper

NumericStepper と同様に、OptionStepper (gfx.controls.OptionStepper) は 1 つの数値を表示しますが、数字以外も表示することができます。このコンポーネントは dataProvider インスタンスを使って現在の値を照会するので、さまざまなタイプの任意数のエレメントをサポートすることができます。表示される数値は、OptionStepper の selectedIndex プロパティ

を使ったコードで設定します。このプロパティは、付属している dataProvider への 0-ベースのインデックスです。以下の例のように dataProvider はコードを使って割り当てられます:

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

2.4.2.1 ユーザーの操作

NumericStepper のように、OptionStepper には 2 つの矢印ボタンが含まれており、これらのボタンをマウスや同等のコントローラでクリックして、その現在の値を変更することができます。フォーカスが適用されると、現在の値は左右の矢印キーや同等のコントロールを使って、キーボードで変更することもできます。これらのキーは現在の値を前後の値に変更します。Home キーや End キー、または同等のコントロールを押すと、現在の値が dataProvider の最初と最後のエレメントに変更されます。

2.4.2.2 コンポーネントのセットアップ

NumericStepper クラスを使用する MovieClip には、以下の指定されたサブエレメントが必須です。オプションのエレメントであれば、その旨表記されています:

- *textField :* TextField タイプ。現在の値の表示に使用されます。
- *nextBtn :* CLIK Button のタイプです。現在の値を dataProvider の次のエレメントに変更します。
- *prevBtn :* CLIK Button のタイプです。現在の値を dataProvider の前のエレメントに変更します。

2.4.2.3 ステート

OptionStepper コンポーネントは、その focused と disabled プロパティに基づいて、3 つのステートをサポートします:

- *default* または有効のステート
- *focused* ステート、textField 領域を強調表示します。
- *disabled* ステート

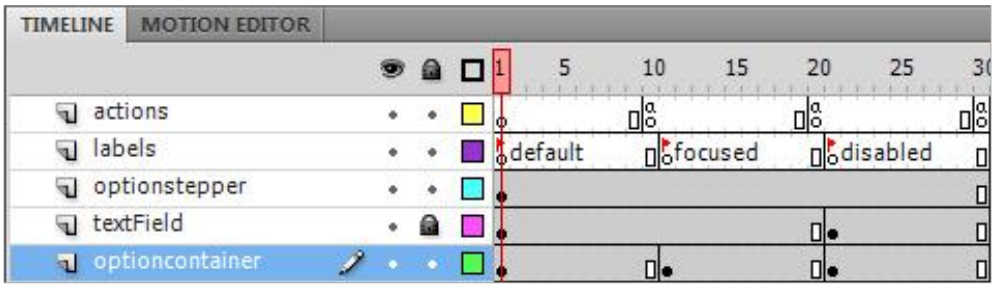

図 36: OptionStepper のタイムライン

2.4.2.4 検証可能なプロパティ

OptionStepper コンポーネントから生じる MovieClip は、以下の検証可能なプロパティを備えています:

visible	false に設定した場合、このコンポーネントを非表示にします。
disabled	true に設定した場合、コンポーネントを無効にします。

2.4.2.5 イベント

すべてのイベント コールバックは、そのイベントの関連情報を含む 1 つの Object パラメータを受け取ります。以下のプロパティは、すべてのイベントに共通です。

- *type :* イベントのタイプ
- *target :* イベントを作成したターゲット

OptionStepper コンポーネントが作成するイベントは以下の表のとおりです。イベントの隣に表示されているプロパティは、共通プロパティに追加して提供されています。

show	コンポーネントの visible プロパティが、実行時に true に設定されました。

hide	コンポーネントの visible プロパティが、実行時に false に設定されました。
change	OptionStepper の値が、変更されました。
stateChange	OptionStepper の focused または disabled プロパティが、変更されました。 • *state* ： 新規ステートの名前です。文字列タイプで、値は "default"、"focused"、または "disabled" です。

以下の例は OptionStepper 値の変更をリッスンする方法を示しています:

```
myOS.addEventListener("change", this, "onValueChange");
function onValueChange(event:Object) {
      trace("os value: " + event.target.selectedItem);
      // ここに何かを追加する
}
```

2.4.3 ListItemRenderer

図 37: スキニング前の ListItemRenderer

ListItemRenderer (gfx.controls.ListItemRenderer) は CLIK Button クラスから派生したものであり、このクラスを拡張して、そのコンテナ コンポーネントに便利なリスト関連プロパティを含めます。しかし、これはスタンドアローン コンポーネントとして設計されたものではなく、ScrollingList、TileList、DropdownMenu コンポーネントと組み合わせて使用されるだけです。

2.4.3.1 ユーザーの操作

ListItemRenderer は Button コンポーネントから派生したものなので、マウスを使って押すなど Button と似たユーザーの操作を備えています。マウス カーソルを ListItemRenderer 上に置いたり、そこから離すときもコンポーネントに影響し、さらにマウス カーソルをその

上にドラッグイン／ドラッグアウトするときも同様です。キーボードや同等のコントローラ操作は、この ListItemRenderer のコンテナ コンポーネントで定義されます。

2.4.3.2 コンポーネントのセットアップ

CLIK ListItemRenderer クラスを使用する MovieClip には、以下の指定されたサブエレメントが必須です。オプションのエレメントである場合はその旨表記されています:

- *textField :* (オプション) TextField タイプ。リスト アイテムのラベルです。
- *focusIndicator :* (オプション) MovieClip タイプ。フォーカスが適用されたステートの表示に使用される別の MovieClip です。存在する場合、この MovieClip は "show" と "hide" の 2 つの指定されたフレームを備えている必要があります。

2.4.3.3 ステート

ListItemRenderer はコンテナ コンポーネント内で選択できるので、選択状態を表すためにキーフレームの *selected* セットが必要となります。このコンポーネントのステートは以下のとおりです:

- *up* またはデフォルトのステート
- マウス カーソルがコンポーネント上にあるとき、またはフォーカスが適用されているときの *over* ステート
- ボタンが押下されたときの *down* ステート
- *disabled* ステート
- *selected_up* またはデフォルトのステート
- マウス カーソルがコンポーネント上にあるとき、またはフォーカスが適用されているときの *selected_over* ステート
- ボタンが押下されたときの *selected_down* ステート
- *selected_disabled* ステート

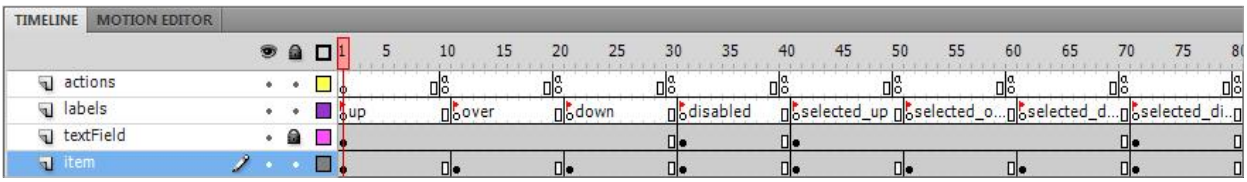

図 38: ListItemRenderer のタイムライン

これは ListItemRenderer に必要な最小限のキーフレーム セットです。Button コンポーネントがサポートするステートとキーフレームの拡張セットと、それに伴う ListItemRenderer コンポーネントについては、「Getting Started with CLIK Buttons」を参照してください。

2.4.3.4 検証可能なプロパティ

ListItemRenderer はコンテナ コンポーネントが制御し、ユーザーが手動で構成することはないので、少数の Button の検証可能プロパティのサブセットしか含まれていません。

label	ListItemRenderer のラベルを設定します。
visible	false に設定した場合、ボタンを非表示にします。
disabled	true に設定した場合、ボタンを無効にします。

2.4.3.5 イベント

すべてのイベント コールバックは、そのイベントの関連情報を含む 1 つの Object パラメータを受け取ります。以下のプロパティは、すべてのイベントに共通です。

- *type :* イベントのタイプ
- *target :* イベントを作成したターゲット

ListItemRenderer コンポーネントが生成するイベントは、Button コンポーネントと同じです。イベントの隣に表示されているプロパティは、共通プロパティに追加して提供されています。

show	コンポーネントの visible プロパティが、実行時に true に設定されました。
hide	コンポーネントの visible プロパティが、実行時に false に設定されました。
focusIn	このコンポーネントにフォーカスが、適用されました。
focusOut	このコンポーネントのフォーカスが、解除されました。
select	コンポーネントの selected プロパティが、変更されました。 • *selected :* Button の selected プロパティです。ブール型です。
stateChange	ボタンのステートが、変更されました。 • *state :* Button の新規ステートです。文字列タイプで、値は "up"、"over"、"down" などです。ステートの全リストについては、「Getting Started with CLIK Buttons」を参照してください。
rollOver	マウス カーソルが、ボタン上に移動しました。 • *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
rollOut	マウス カーソルが、ボタンから離れました。 • *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
press	ボタンが、押下されました。 • *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合

	に限って適用できます)。数値型で、値は 0 から 3 までです。
doubleClick	ボタンが、ダブルクリックされました。*doubleClickEnabled* プロパティが true の場合に限って発生します。 • *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
click	ボタンが、クリックされました。 • *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
dragOver	マウス カーソルが (マウスの左ボタンを押したままの状態で)、 ボタン上にドラッグされました。 • *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
dragOut	マウス カーソルが(マウスの左ボタンを押したままの状態で)、 ボタンから離れました。 • *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
releaseOutside	マウス カーソルが、ボタンから離れ、マウスの左ボタンが、解放されました。 • *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。

2.4.4 ScrollingList

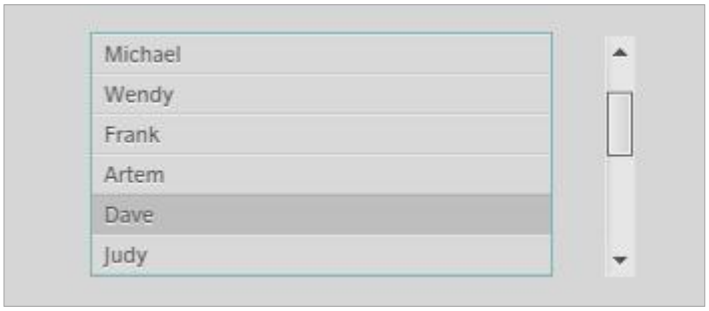

図 39: スキニング前の ScrollingList

ScrollingList (gfx.controls.ScrollingList) はそのエレメントをスクロールできるリスト コンポーネントです。このコンポーネントは自分でリスト項目をインスタンス化したり、ステージ上の既存のリスト項目を使用することができます。また、ScrollIndicator や ScrollBar コンポーネントをこのリスト コンポーネントに追加して、スクロールのフィードバックや制御を行います。このコンポーネントは dataProvider で設定されます。以下の例のように dataProvider はコードを使って割り当てられます:

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

デフォルトでは、ScrollingList はそのコンテンツのために ListItemRenderer コンポーネントを使用します。したがって、itemRenderer の検証可能プロパティが別のコンポーネントに変更されていない限り、ListItemRenderer は動作するために FLA ファイルの [ライブラリ] にも存在しなければなりません。詳細は「検証可能プロパティ」の章を参照してください。

2.4.4.1 ユーザーの操作

リスト項目、または添付されている ScrollBar インスタンスをクリックすると、フォーカスが ScrollingList コンポーネントに移動します。フォーカスが適用されている間、キーボードの上下の矢印や同等のコントロールを押すと、1 要素ごとにリストの選択範囲をスクロールします。何の要素も選択されていない場合は、最上部の要素が自動的にこの動作に選択されます。カーソルが ScrollingList の境界の先頭にある場合は、マウス ホイールはリストをスクロールします。

リストの境界でスクロールするという動作は、ScrollingList の *wrapping* プロパティで判断され、検証可能ではありません。*wrapping* が "normal" に設定されている場合、選択範囲がリストの先頭、または最後に達すると、フォーカスはそのコンポーネントを離れます。*wrapping* が "wrap" に設定されている場合、選択範囲は先頭か最後にまとめられます。*wrapping* が "stick" に設定されている場合、データの最後に到達すると選択はそこで止まり、フォーカスは隣接するコンポーネントに移動しません。

キーボードの Page Up や Page Down キー、または同等のコントロールを押すと、ページ単位で、つまりリストに表示されている項目の数で選択範囲をスクロールします。Home キーや End キー、または同等のコントロールを押すと、リストをそれぞれ最初と最後のエレメントにスクロールします。添付されている ScrollBar コンポーネントを操作すると、予想通り ScrollingList に影響します。ScrollBar 独自のユーザーの操作については、ScrollBar の章を参照してください。

開発者は簡単にゲームパッドのコントロールを、キーボードやマウスのコントロールにマップすることができます。たとえば、キーボードの矢印キーは通常、コンソール コントローラの D パッドにマップされます。このマッピングによって CLIK で作成した UI は、各種のプラットフォームで動作することができます。

2.4.4.2 コンポーネントのセットアップ

ScrollingList には指定されたサブエレメントはいっさい必要ありません。ただし、ステージで ScrollingList コンポーネントのインスタンスの配置やサイズ変更を行うときに、目に見える背景は便利です。

2.4.4.3 ステート

ScrollingList コンポーネントは、その focused と disabled プロパティに基づいて、3 つのステートをサポートします:

- *default* または有効のステート
- *focused* ステート、通常コンポーネントの境界領域を強調表示します。
- *disabled* ステート

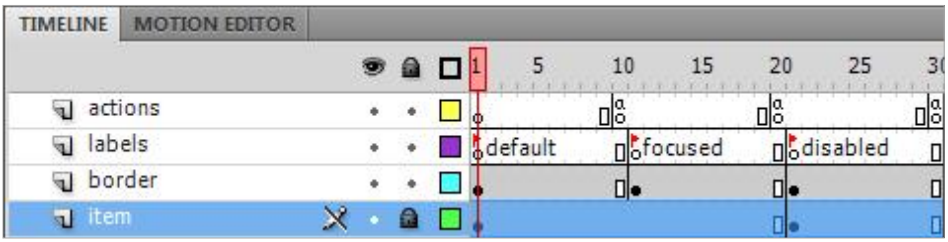

図 40: ScrollingList のタイムライン

2.4.4.4 検証可能なプロパティ

ScrollingList コンポーネントから生じる MovieClip は、以下の検証可能なプロパティを備えています:

visible	false に設定した場合、このコンポーネントを非表示にします。添付されているスクロールバー、または外部のリスト項目レンダラは表示します。
disabled	true に設定した場合、コンポーネントを無効にします。これは添付されているスクロールバーとリスト項目 (内部で作成されたものと外部のレンダラの両方) も無効にします。
itemRenderer	ListItemRenderer のシンボル名です。リスト項目インスタンスを内部的に作成するため使用されます。*rendererInstanceName* プロパティが設定されている場合、効果はありません。
renderInstanceName	この ScrollingList コンポーネントと併用するための、外部のリスト項目レンダラのプレフィックスです。ステージ上のリスト項目インスタンスは、このプロパティ値をプレフィックスとして付ける必要があります。このプロパティが 'r' という値に設定されていれば、このコンポーネントで使用されるすべてのリスト項目インスタンスは、'r1'、'r2'、'r3'… という値にする必要があります。最初の項目は 1 になります。
scrollBar	ステージ上の ScrollBar コンポーネントのインスタンス名、またはシンボル名です。インスタンス名が指定されると、ScrollingList はそのインスタンスにつながります。シンボル名

	が指定されると、そのシンボルのインスタンスを ScrollingList が作成します。
margin	リスト コンポーネントの境界と内部で作成されたリスト項目の間のマージンです。*rendererInstanceName* プロパティが設定されている場合、この値に効果はありません。
rowHeight	内部で作成されたリスト項目インスタンスの高さです。*rendererInstanceName* プロパティが設定されている場合、この値に効果はありません。
paddingTop	追加の余白（パディング）をリストアイテムの上端に挿入。rendererInstanceName プロパティが設定されているときには、この値は効果がありません。自動で生成されているスクロールバーには影響しません。
padding Bottom	追加の余白（パディング）をリストアイテムの下端に挿入。rendererInstanceName プロパティが設定されているときには、この値は効果がありません。自動で生成されているスクロールバーには影響しません。
paddingLeft	追加の余白（パディング）をリストアイテムの左端に挿入。rendererInstanceName プロパティが設定されているときには、この値は効果がありません。自動で生成されているスクロールバーには影響しません。
paddingRight	追加の余白（パディング）をリストアイテムの右端に挿入。rendererInstanceName プロパティが設定されているときには、この値は効果がありません。自動で生成されているスクロールバーには影響しません。
thumbOffsetTop	スクロールバーのサムを上端でオフセット。リストが、自動的にスクロールバー インスタンスを生成しないのであれば、このプロパティの効果はありません。
thumbOffsetBottom	スクロールバーのサムを下端でオフセット。リストが、自動的にスクロールバー インスタンスを生成しないのであれば、このプロパティの効果はありません。
thumbSizeFactor	スクロールバーのサムのためのページサイズの係数です。数値が 1.0 より大きい場合には、係数値によってサムが大きくなります。正数であって、1.0 より小さい場合には、サムのサイズが小さくなります。スクロールバーがリストに付属していない場合には、効果がありません。

2.4.4.5 イベント

すべてのイベント コールバックは、そのイベントの関連情報を含む 1 つの Object パラメータを受け取ります。以下のプロパティは、すべてのイベントに共通です。

- *type：*イベントのタイプ
- *target：*イベントを作成したターゲット

ScrollingList コンポーネントが作成するイベントは以下の表のとおりです。イベントの隣に表示されているプロパティは、共通プロパティに追加して提供されています。

show	コンポーネントの visible プロパティが、実行時に true に設定されました。
hide	コンポーネントの visible プロパティが、実行時に false に設定されました。
focusIn	このコンポーネントにフォーカスが、適用されました。
focusOut	このコンポーネントのフォーカスが、解除されました。
change	選択されたインデックスが、変更されました。 • *index :* 新規に選択されたインデックスです。数値型で、値は 0 からリストの項目数から 1 を引いたものまでです。
itemPress	リスト項目が、押下されました。 • *renderer :* 押下されたリスト項目です。CLIK Button のタイプです。 • *item :* そのリスト項目に関連するデータです。この値はリストの dataProvider から取得されます。AS2 の Object タイプです。 • *index :* リストの dataProvider に関連するリスト項目のインデックスです。数値型で、値は 0 からリストの項目数から 1 を引いたものまでです。 • *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
itemClick	リスト項目が、クリックされました。 • *renderer :* クリックされたリスト項目です。CLIK Button のタイプです。 • *item :* そのリスト項目に関連するデータです。この値はリストの dataProvider から取得されます。AS2 の Object タイプです。 • *index :* リストの dataProvider に関連するリスト項目のインデックスです。数値型で、値は 0 からリストの項目数から 1 を引いたものまでです。 • *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
itemDoubleClick	リスト項目が、ダブルクリックされました。 • *renderer :* ダブルクリックされたリスト項目です。CLIK Button のタイプです。 • *item :* そのリスト項目に関連するデータです。この値はリストの dataProvider から取得されます。AS2 の Object タイプです。 • *index :* リストの dataProvider に関連するリスト項目のインデックスです。数値型で、値は 0 からリストの項目数から 1 を引いたものまでです。 • *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合

	に限って適用できます)。数値型で、値は 0 から 3 までです。
itemRollOver	マウス カーソルが、リスト項目上に移動しました。 • *renderer：*カーソルが上に置かれたリスト項目です。CLIK Button のタイプです。 • *item：*そのリスト項目に関連するデータです。この値はリストの dataProvider から取得されます。AS2 の Object タイプです。 • *index：*リストの dataProvider に関連するリスト項目のインデックスです。数値型で、値は 0 からリストの項目数から 1 を引いたものまでです。 • *mouseIndex：*このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
itemRollOut	マウス カーソルが、リスト項目から離れました。 • *renderer：*カーソルが離れたリスト項目です。CLIK Button のタイプです。 • *item：*そのリスト項目に関連するデータです。この値はリストの dataProvider から取得されます。AS2 の Object タイプです。 • *index：*リストの dataProvider に関連するリスト項目のインデックスです。数値型で、値は 0 からリストの項目数から 1 を引いたものまでです。 • *mouseIndex：*このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。

以下の例は、クリックされているリスト項目をリッスンする方法を示しています:

```
myList.addEventListener("itemClick", this, "onItemClicked");
function onItemClicked(event:Object) {
      trace("list item was clicked: " + event.renderer);
      // ここに何かを追加する
}
```

2.4.4.6 ヒントとコツ

複合オブジェクトのセットから 1 つのラベルを表示する:

```
// 以下のように、項目オブジェクトにある場合、
// ScrollingList は自動的に 'label' という名前のプロパティを使用する:
list.dataProvider = [{label: "one", data:1}, {label: "two", data:2}];

// ただし、その項目オブジェクトが以下のように他の label プロパティを持つ場合、
// リストは代わりにそのプロパティを使用するように
```

```
    // 構成することができる:
    list.labelField = "name";
    list.dataProvider = [{name: "one", data:1}, {name: "two", data:2}];

    // 項目オブジェクトからラベルを作成するロジックが複雑で、
    // 関数を必要とする場合、labelFunction プロパティを
    // 設定する:
list.labelFunction = function(itemObj:Object):String {
    // ラベルを作成するロジック
}
list.dataProvider = [{p1: "foo", p2: 1}, {p1: "bar", p2: 2}];
```

2.4.5 TileList

図 41: スキニング前の TileList

TileList (gfx.controls.TileList) は ScrollingList と同様に、そのエレメントをスクロールできるリスト コンポーネントです。このコンポーネントは自分でリスト項目をインスタンス化したり、ステージ上の既存のリスト項目を使用することができます。また、ScrollIndicator や ScrollBar コンポーネントをこのリスト コンポーネントに追加して、スクロールのフィードバックや制御を行います。TileList と ScrollingList の違いは、TileList が同時に複数の行と列をサポートできることです。リスト項目の選択範囲は 4 つの基本方位すべてに移動できます。このコンポーネントは dataProvider で設定されます。以下の例のように dataProvider はコードを使って割り当てられます:

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

デフォルトでは TileList はそのコンテンツのために ListItemRenderer コンポーネントを使用します。したがって、itemRenderer の検証可能プロパティが別のコンポーネントに変更されていない限り、ListItemRenderer は動作するために FLA ファイルの [ライブラリ] にも存在しなければなりません。詳細は「検証可能プロパティ」の章を参照してください。

2.4.5.1 ユーザーの操作

リスト項目、または添付されている ScrollBar インスタンスをクリックすると、フォーカスが TileList コンポーネントに移動します。フォーカスが適用されている間、キーボードの上下の矢印や同等のコントロールを押すと、複数の行が含まれている場合は 1 要素ごとにリストの選択範囲を縦にスクロールします。左右の矢印、または同等のコントロールは、複数の列が含まれている場合は横にリストの選択範囲をスクロールします。TileList に複数の行と列が含まれている場合、4 つすべての矢印キー、または同等のコントロールを使って、リストの選択範囲を移動することができます。 何の要素も選択されていない場合は、最上部の要素が自動的にこの動作に選択されます。カーソルが TileList の境界の先頭にある場合は、マウス ホイールはリストをスクロールします。

キーボードの Page Up や Page Down キー、または同等のコントロールを押すと、ページ単位で、つまりリストに表示されている行数で選択範囲をスクロールします。Home キーや End キー、または同等のコントロールを押すと、リストをそれぞれ最初と最後のエレメントにスクロールします。添付されている ScrollBar コンポーネントを操作すると、予想通り ScrollingList に影響します。ScrollBar 独自のユーザーの操作については、ScrollBar の章を参照してください。

2.4.5.2 コンポーネントのセットアップ

TileList には指定されたサブエレメントはいっさい必要ありません。ただし、ステージで TileList コンポーネントのインスタンスの配置やサイズ変更を行うときに、目に見える背景は便利です。

2.4.5.3 ステート

TileList コンポーネントは、その focused と disabled プロパティに基づいて、3 つのステートをサポートします:

- *default* または有効のステート
- *focused* ステート、通常コンポーネントの境界領域を強調表示します。
- *disabled* ステート

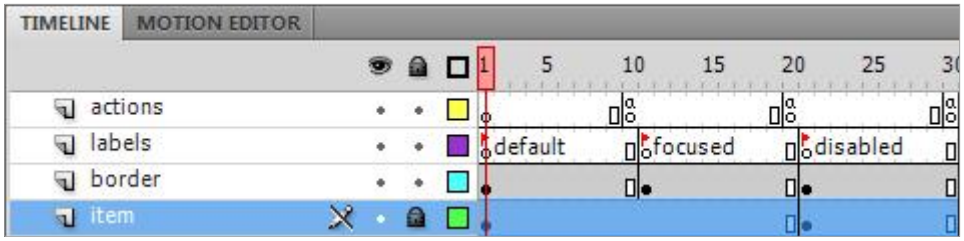

図 42: TileList のタイムライン

2.4.5.4 検証可能なプロパティ

TileList コンポーネントから生じる MovieClip は、以下の検証可能なプロパティを備えています:

visible	false に設定した場合、このコンポーネントを非表示にします。添付されているスクロールバー、または外部のリスト項目レンダラは表示します。
disabled	true に設定した場合、コンポーネントを無効にします。これは添付されているスクロールバーとリスト項目 (内部で作成されたものと外部のレンダラの両方) も無効にします。
itemRenderer	ListItemRenderer のシンボル名です。リスト項目インスタンスを内部的に作成するため使用されます。*rendererInstanceName* プロパティが設定されている場合、効果はありません。
renderInstanceName	この TileList コンポーネントと併用するための、外部のリスト項目レンダラのプレフィックスです。ステージ上のリスト項目インスタンスは、このプロパティ値をプレフィックスとして付ける必要があります。このプロパティが 'r' という値に設定されていれば、このコンポーネントで使用されるすべてのリスト項目インスタンスは、'r1'、'r2'、'r3'… という値にする必要があります。最初の項目は 1 になります。
scrollBar	ステージ上の ScrollBar コンポーネントのインスタンス名、またはシンボル名です。インスタンス名が指定されると、TileList はそのインスタンスにつながります。シンボル名が指定されると、そのシンボルのインスタンスを TileList が作成します。
margin	リスト コンポーネントの境界と内部で作成されたリスト項目の間のマージンです。*rendererInstanceName* プロパティが設定されている場合、この値に効果はありません。
rowHeight	内部で作成されたリスト項目インスタンスの高さです。*rendererInstanceName* プロパティが設定されている場合、この値に効果はありません。
columnWidth	内部で作成されたリスト項目インスタンスの幅です。*rendererInstanceName* プロパティが設定されている場合、この値に効果はありません。
externalColumnCount	*rendererInstanceName* プロパティが設定されると、この値を使って外部のレンダラが使用する列の数を TileList に通知します。
direction	スクロールの方向です。行と列のセマンティクスはこの値によって変わることはありません。

2.4.5.5 イベント

すべてのイベント コールバックは、そのイベントの関連情報を含む 1 つの Object パラメータを受け取ります。以下のプロパティは、すべてのイベントに共通です。

- *type :* イベントのタイプ
- *target :* イベントを作成したターゲット

TileList コンポーネントが作成するイベントは以下の表のとおりです。イベントの隣に表示されているプロパティは、共通プロパティに追加して提供されています。

show	コンポーネントの visible プロパティが、実行時に true に設定されました。
hide	コンポーネントの visible プロパティが、実行時に false に設定されました。
focusIn	このコンポーネントにフォーカスが、適用されました。
focusOut	このコンポーネントのフォーカスが、解除されました。
change	選択されたインデックスが、変更されました。 • *index :* 新規に選択されたインデックスです。数値型で、値は 0 からリストの項目数から 1 を引いたものまでです。
itemPress	リスト項目が、押下されました。 • *renderer :* 押下されたリスト項目です。CLIK Button のタイプです。 • *item :* そのリスト項目に関連するデータです。この値はリストの dataProvider から取得されます。AS2 の Object タイプです。 • *index :* リストの dataProvider に関連するリスト項目のインデックスです。数値型で、値は 0 からリストの項目数から 1 を引いたものまでです。 • *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
itemClick	リスト項目が、クリックされました。 • *renderer :* クリックされたリスト項目です。CLIK Button のタイプです。 • *item :* そのリスト項目に関連するデータです。この値はリストの dataProvider から取得されます。AS2 の Object タイプです。 • *index :* リストの dataProvider に関連するリスト項目のインデックスです。数値型で、値は 0 からリストの項目数から 1 を引いたものまでです。 • *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。

itemDoubleClick	リスト項目が、ダブルクリックされました。 • *renderer :* ダブルクリックされたリスト項目です。CLIK Button のタイプです。 • *item :* そのリスト項目に関連するデータです。この値はリストの dataProvider から取得されます。AS2 の Object タイプです。 • *index :* リストの dataProvider に関連するリスト項目のインデックスです。数値型で、値は 0 からリストの項目数から 1 を引いたものまでです。 • *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
itemRollOver	マウス カーソルが、リスト項目上に移動しました。 • *renderer :* カーソルが上に置かれたリスト項目です。CLIK Button のタイプです。 • *item :* そのリスト項目に関連するデータです。この値はリストの dataProvider から取得されます。 • *index :* リストの dataProvider に関連するリスト項目のインデックスです。数値型で、値は 0 からリストの項目数から 1 を引いたものまでです。 • *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
itemRollOut	マウス カーソルが、リスト項目から離れました。 • *renderer :* カーソルが離れたリスト項目です。CLIK Button のタイプです。 • *item :* そのリスト項目に関連するデータです。この値はリストの dataProvider から取得されます。AS2 の Object タイプです。 • *index :* リストの dataProvider に関連するリスト項目のインデックスです。数値型で、値は 0 からリストの項目数から 1 を引いたものまでです。 • *mouseIndex :* このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。

以下の例は TileList がいつフォーカスを受け取ったかを判断する方法を示しています:

```
myList.addEventListener("focusIn", this, "onListFocused");
function onListFocused(event:Object) {
      trace("tile list was focused!");
      // ここに何かを追加する
}
```

2.4.5.6 ヒントとコツ

複合オブジェクトのセットから 1 つのラベルを表示する:

```
    // 以下のように、項目オブジェクトにある場合、
    // TileList は自動的に 'label' という名前のプロパティを使用する:
    list.dataProvider = [{label: "one", data:1}, {label: "two", data:2}];

    // ただし、その項目オブジェクトが以下のように他の label プロパティを持つ場合、
    // リストは代わりにそのプロパティを使用するように
    // 構成することができる:
    list.labelField = "name";
    list.dataProvider = [{name: "one", data:1}, {name: "two", data:2}];

    // 項目オブジェクトからラベルを作成するロジックが複雑で、
    // 関数を必要とする場合、labelFunction プロパティを
    // 設定する:
  list.labelFunction = function(itemObj:Object):String {
    // ラベルを作成するロジック
  }
  list.dataProvider = [{p1: "foo", p2: 1}, {p1: "bar", p2: 2}];
```

2.4.6 DropdownMenu

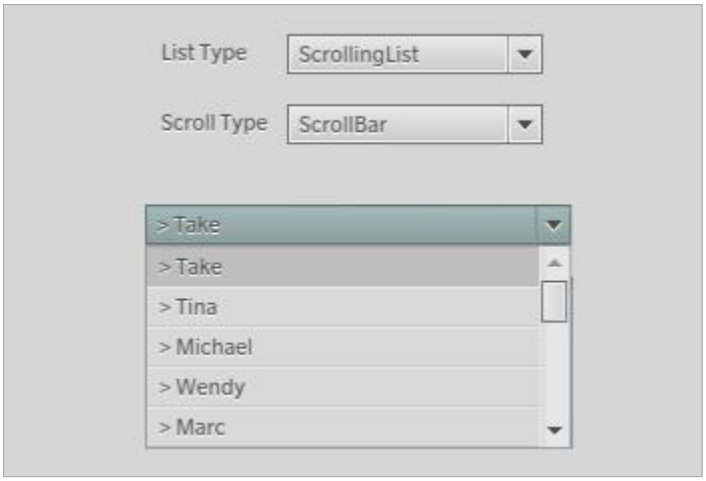

図 43: スキニング前の DropdownMenu

DropdownMenu (gfx.controls.DropdownMenu) はボタンとリストの動作をラップします。このコンポーネントをクリックすると、選択するエレメントが含まれたリストが表示されます。DropdownMenu はアイドル状態の選択されたエレメントのみを表示します。このコンポーネントは、ScrollBar、または ScrollIndicator のいずれかを伴う ScrollingList か TileList の、どちらかを使用するよう構成できます。このリストはインストールされた dataProvider で設定されます。DropdownMenu のリスト エレメントは dataProvider で設定されます。以下の例のように dataProvider はコードを使って割り当てられます:

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

デフォルトでは DropdownMenu はそのコンテンツのために ScrollingList コンポーネントを使用します。したがって、dropdown 検証可能プロパティが別のコンポーネントに変更されていない限り、ScrollingList と ListItemRenderer は動作するために FLA ファイルの [ライブラリ] にも存在しなければなりません。詳細は「検証可能プロパティ」の章を参照してください。

また、デフォルトでは DropdownMenu はスクロールバーをそのリスト エレメントには添付しないので注意してください。ScrollBar、または ScrollIndicator は、コードを使って DropdownMenu のリスト エレメントに添付する必要があります。詳細は「ヒントとコツ」の章を参照してください。

2.4.6.1 ユーザーの操作

DropdownMenu インスタンスをクリックする、または Enter キー、あるいは同等のコントロールを押すと、選択可能なエレメントのリストが表示されます。リストが開くと、フォーカスもそのリストに移動されます。ユーザーは ScrollingList、TileList、ScrollBar の「ユーザーの操作」の章で説明したように、リストを操作することができます。リスト項目をクリックすると、DropdownMenu コンポーネントでその項目を選択する、リストを閉じる、そして選択した項目を表示するようになります。リストの境界外でクリックすると自動的にリストが閉じられ、フォーカスは DropdownMenu コンポーネントに戻ります。

2.4.6.2 コンポーネントのセットアップ

DropdownMenu のそのほとんどの機能は Button コンポーネントから派生しています。その結果、DropdownMenu クラスを使用する MovieClip は以下の指定されたサブエレメントが必須です。このリストとスクロールバーはダイナミックに作成されます。オプションのエレメントである場合には、その旨表記されています:

- *textField :* (オプション) TextField タイプ。ボタンのラベルです。
- *focusIndicator :* (オプション) MovieClip タイプ。フォーカスが適用されたステートの表示に使用される別の MovieClip です。存在する場合、この MovieClip は "show" と "hide" の 2 つの指定されたフレームを備えている必要があります。

2.4.6.3 ステート

DropdownMenu は開いているときに切り替えられるので、ToggleButton や CheckBox と同じ選択状態を表すステートが必要となります。このようなステートには以下が含まれます:

- *up* またはデフォルトのステート
- マウス カーソルがコンポーネント上にあるとき、またはフォーカスが適用されているときの *over* ステート
- ボタンが押下されたときの *down* ステート
- *disabled* ステート
- *selected_up* またはデフォルトのステート
- マウス カーソルがコンポーネント上にあるとき、またはフォーカスが適用されているときの *selected_over* ステート
- ボタンが押下されたときの *selected_down* ステート
- *selected_disabled* ステート

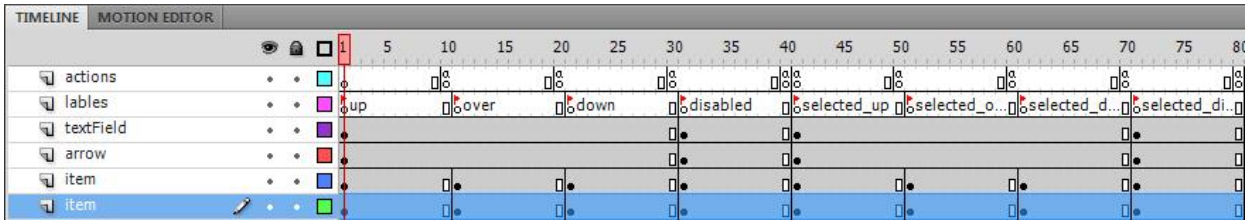

図 44: DropdownMenu のタイムライン

これは DropdownMenu に必要な最小限のキーフレーム セットです。Button コンポーネントがサポートするステートとキーフレームの拡張セットと、それに伴う DropdownMenu コンポーネントについては、「Getting Started with CLIK Buttons」を参照してください。

2.4.6.4 検証可能なプロパティ

DropdownMenu コンポーネントの検証可能なプロパティは以下のとおりです:

visible	false に設定した場合、このコンポーネントを非表示にします。
disabled	true に設定した場合、コンポーネントを無効にします。
dropdown	DropdownMenu コンポーネントと併用するリスト コンポーネント (ScrollingList または TileList) のシンボル名です。
dropdownWidth	ドロップダウン リストの幅です。この値が -1 であれば、DropdownMenu はリストをコンポーネントの幅に合わせます。
itemRenderer	ドロップダウン リストの項目レンダラのシンボル名です。ドロップダウン リスト インスタンスが作成します。
scrollBar	ドロップダウン リストのスクロールバーのシンボル名です。ドロップダウン リスト インスタンスが作成します。値が空白の場合、そのドロップダウン リストにはスクロール バーがありません。
margin	リスト コンポーネントの境界と内部で作成されたリスト項目の間のマージンです。このマージンは、自動的に生成されたスクロールバーにも影響します。
paddingTop	追加の余白（パディング）をリストアイテムの上端に挿入。自動で生成されているスクロールバーには影響しません。
paddingBottom	追加の余白（パディング）をリストアイテムの下端に挿入。自動で生成されているスクロールバーには影響しません。
paddingLeft	追加の余白（パディング）をリストアイテムの左端に挿入。自動で生成されているスクロールバーには影響しません。
paddingRight	追加の余白（パディング）をリストアイテムの右端に挿入。ん。自動で生成されているスクロールバーには影響しません。
thumbOffsetTop	スクロールバーのサムを上端でオフセット。リストが、自動的にスクロールバー インスタンスを生成しないのであれば、このプロパティの効果はありません。
thumbOffsetBottom	スクロールバーのサムを下端でオフセット。リストが、自動的

	にスクロールバー インスタンスを生成しないのであれば、このプロパティの効果はありません。
thumbSizeFactor	スクロールバーのサムのためのページサイズの係数です。数値が 1.0 より大きい場合には、係数値によってサムが大きくなります。スクロールバーがリストに付属していない場合には、効果がありません。
offsetX	ドロップダウン ボタン位置の水平方向オフセット。正数値では、リストをドロップダウン ボタンの水平位置の右方向に動かします。
offsetY	ドロップダウン ボタン位置の垂直方向オフセット。正数値では、リストをボタンから遠ざけます。
extent	上記の offsetX と連係して使われる追加幅のオフセット。dropdownWidth プロパティの値が、-1 である場合以外では、ここでの値に効果はありません。
direction	リストの開く方向。有効な値は、“up”または“down”です。

2.4.6.5 イベント

すべてのイベント コールバックは、そのイベントの関連情報を含む 1 つの Object パラメータを受け取ります。以下のプロパティは、すべてのイベントに共通です。

- *type：*イベントのタイプ
- *target：*イベントを作成したターゲット

DropdownMenu コンポーネントが作成するイベントは以下の表のとおりです。*change* イベント以外、Button コンポーネントと同じです。イベントの隣に表示されているプロパティは、共通プロパティに追加して提供されています。

show	コンポーネントの visible プロパティが、実行時に true に設定されました。
hide	コンポーネントの visible プロパティが、実行時に false に設定されました。
focusIn	このコンポーネントにフォーカスが、適用されました。
focusOut	このコンポーネントのフォーカスが、解除されました。
change	選択されたインデックスが、変更されました。 • *index：*新規に選択されたインデックスです。数値型で、値は 0 からリストの項目数から 1 を引いたものまでです。 • *data：*選択されたインデックスのリスト項目に関連するデータです。AS2 の Object タイプです。
select	コンポーネントの selected プロパティが、変更されました。 • *selected：*Button の selected プロパティです。ブール型です。
stateChange	ボタンのステートが、変更されました。 • *state：*Button の新規ステートです。文字列タイプで、値は "up"、"over"、"down" などです。ステートの全リストについては、「Getting Started with CLIK Buttons」を参照し

	てください。
rollOver	マウス カーソルが、ボタン上に移動しました。 • *mouseIndex：*このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
rollOut	マウス カーソルが、ボタンから離れました。 • *mouseIndex：*このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
press	ボタンが、押下されました。 • *mouseIndex：*このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
doubleClick	ボタンが、ダブルクリックされました。*doubleClickEnabled* プロパティが true の場合に限って発生します。 • *mouseIndex：*このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
click	ボタンが、クリックされました。 • *mouseIndex：*このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
dragOver	マウス カーソルが (マウスの左ボタンを押したままの状態で)、 ボタン上にドラッグされました。 • *mouseIndex*：このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
dragOut	マウス カーソルが(マウスの左ボタンを押したままの状態で)、 ボタンから離れました。 • *mouseIndex：*このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。
releaseOutside	マウス カーソルが、ボタンから離れ、マウスの左ボタンが、解放されました。 • *mouseIndex：*このイベントの作成に使用されたマウス カーソルのインデックスです (マルチマウス カーソル環境の場合に限って適用できます)。数値型で、値は 0 から 3 までです。

2.4.6.6 ヒントとコツ

ドロップダウン メニューが開いているか閉じているかを判断する:

```
    dropdown.addEventListener("click", this, "onClick");
function onClick(e:Object) {
    if (e.target.isOpen) {
```

```
            // 開いているときはここに何かを追加する。
        }
    }
```

実行時に DropdownMenu をダイナミックに作成する:

```
    // コードで使用されているリンケージ ID が使用可能であることを確認する;
    // つまり、それらの ID が参照するシンボル/コンポーネントが
    // fla のライブラリに存在しているということである。たとえば、以下のコードは、
    // DropdownMenu、ScrollingList、ScrollBar コンポーネントを必要とする。
    attachMovie("DropdownMenu", "dd", this.getNextHighestDepth(),
                                        {dropdown: "ScrollingList"});
    dd.dataProvider = ["one", "two", "three", "four", "five", "six"];

    // スクロール バーやスクロール インジケータをドロップダウンの
    // リストにインストールすることはやや複雑である。
    // 以下のコードが提示しているトリックを使って、遅延プロパティの設定が必要なためである。
    // 遅延はスクロールバーをリストに添付する前に、まずドロップダウンにリストを
    // インスタンス化させるために必要である:
onEnterFrame = function() {
    dd.dropdown.scrollBar = "ScrollBar";
    onEnterFrame = null;
}
```

複合オブジェクトのセットから 1 つのラベルを表示する:

```
    // 以下のように、項目オブジェクトにある場合、
    // DropdownMenu は自動的に 'label' という名前のプロパティを使用する:
    list.dataProvider = [{label: "one", data:1}, {label: "two", data:2}];

    // ただし、その項目オブジェクトが以下のように他の label プロパティを持つ場合、
    // リストは代わりにそのプロパティを使用するように
    // 構成することができる:
    list.labelField = "name";
    list.dataProvider = [{name: "one", data:1}, {name: "two", data:2}];

    // 項目オブジェクトからラベルを作成するロジックが複雑で、
    // 関数を必要とする場合、labelFunction プロパティを設定する:
list.labelFunction = function(itemObj:Object):String {
    // ラベルを作成するロジック
}
list.dataProvider = [{p1: "foo", p2: 1}, {p1: "bar", p2: 2}];
```

2.5 プログレス タイプ

プログレス タイプは、イベントや動作のステータスや進捗状況 (プログレス) の表示に使用されます。Scaleform CLIK フレームワークには、このカテゴリーに属する StatusIndicator と ProgressBar という 2 つのコンポーネントが含まれています。StatusIndicator コンポーネントは、イベントや動作のステータスの表示に使用されます。ProgressBar コンポーネントは StatusIndicator と同じセマンティクスを備えていますが、プログレス イベントを作成する他のコンポーネントや動作をリッスンする追加機能を含んでいます。

図 45: *Mercenaries 2* - FACTIONS（勢力）ステータス インジケータの例

2.5.1 StatusIndicator

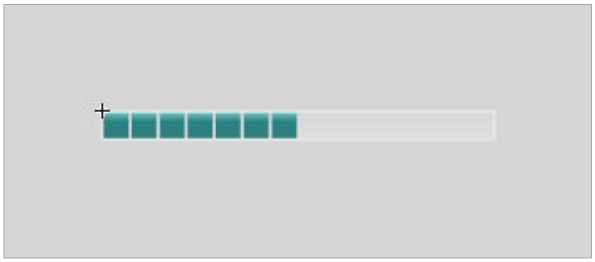

図 46: スキニング前の StatusIndicator

StatusIndicator コンポーネント (gfx.controls.StatusIndicator) は、視覚的なインジケータとしてタイムラインを使用するイベントや動作のステータスを表示します。StatusIndicator の値は、コンポーネントのタイムラインで再生されるフレーム番号を生成するための、最小値と最大値で補間されます。コンポーネントのタイムラインはステータスの表示に使用されるので、画期的な視覚インジケータを完全に自由に作成することができます。

2.5.1.1 ユーザーの操作

StatusIndicator にはユーザーの操作はありません。

2.5.1.2 コンポーネントのセットアップ

CLIK StatusIndicator クラスを使用する MovieClip は、指定されたサブエレメントをいっさい必要としません。ただし、StatusIndicator は、正確な操作に少なくとも 2 つのフレームを必要とします。最初のフレームに stop() コマンドを必ず挿入して、フレームを再生しないようにします。StatusIndicator コンポーネントは、その value プロパティで作成される関連フレームで gotoAndStop() となります。

2.5.1.3 ステート

StatusIndicator コンポーネントにはステートはありません。コンポーネントのフレームは、イベントや動作のステータスの表示に使用されます。

2.5.1.4 検証可能なプロパティ

StatusIndicator コンポーネントから生じる MovieClip は、以下の検証可能なプロパティを備えています:

visible	false に設定した場合、このコンポーネントを非表示にします。
disabled	true に設定した場合、コンポーネントを無効にします。
Value	イベント、または動作のステータス値です。再生されるフレーム番号を生成するため、最小値と最大値で補間されます。
minimum	ターゲット フレームの補間に使用される最小値です。
maximum	ターゲット フレームの補間に使用される最大値です。

2.5.1.5 イベント

すべてのイベント コールバックは、そのイベントの関連情報を含む 1 つの Object パラメータを受け取ります。以下のプロパティは、すべてのイベントに共通です。

- *type :* イベントのタイプ
- *target :* イベントを作成したターゲット

StatusIndicator は特別なイベントをいっさい作成しません。イベントの隣に表示されているプロパティは、共通プロパティに追加して提供されています。

Show	コンポーネントの visible プロパティが、実行時に true に設定されました。
Hide	コンポーネントの visible プロパティが、実行時に false に設定されました。

2.5.2 ProgressBar

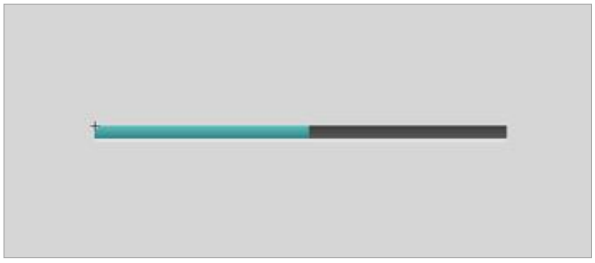

図 47: スキニング前の ProgressBar

ProgressBar (gfx.controls.ProgressBar) はタイムラインを使ってイベントや動作のステータスを表示するという点で、StatusIndicator コンポーネントに似ていますが、プログレスイベントを作成するコンポーネントやイベントと組み合わせて使用することも目的としています。ターゲットを割り当て、そのモードを適切に設定することで、ProgressBar コンポーネントは、ターゲットのロードされた値 (bytesLoaded と bytesTotal) に基づいてその視覚的なステートを自動的に変更します。

2.5.2.1 ユーザーの操作

ProgressBar にはユーザーの操作はありません。

2.5.2.2 コンポーネントのセットアップ

StatusIndicator と同様に、CLIK ProgressBar クラスを使用する MovieClip には、指定されたサブエレメントをいっさい必要としません。ただし、ProgressBar は正確な操作に少なくとも 2 つのフレームが必要となります。最初のフレームに stop() コマンドを必ず挿入して、フレームを再生しないようにします。ProgressBar コンポーネントは、その value プロパティで作成される関連フレームで gotoAndStop() となります。

2.5.2.3 ステート

ProgressBar コンポーネントにはステートはありません。コンポーネントのフレームは、イベントや動作のステータスの表示に使用されます。

2.5.2.4 検証可能なプロパティ

ProgressBar コンポーネントから生じる MovieClip は、以下の検証可能なプロパティを備えています:

visible	false に設定した場合、このコンポーネントを非表示にします。

disabled	true に設定した場合、コンポーネントを無効にします。
Target	bytesLoaded 値と bytesTotal 値を決定するために、ProgressBar が「リッスン」するターゲットです。
Mode	ProgressBar のリッスン モードです。"manual" モードでは、プログレス値は setProgress メソッドを使って設定する必要があります。"polled" モードでは、ターゲットは bytesLoaded と bytesTotal プロパティを公開する必要があり、"event" モードでは、ターゲットは "progress" イベントを bytesLoaded プロパティと bytesTotal プロパティを含めてディスパッチする必要があります。

2.5.2.5 イベント

すべてのイベント コールバックは、そのイベントの関連情報を含む 1 つの Object パラメータを受け取ります。以下のプロパティは、すべてのイベントに共通です。

- *type：*イベントのタイプ
- *target：*イベントを作成したターゲット

ProgressBar コンポーネントが作成するイベントは以下の表のとおりです。イベントの隣に表示されているプロパティは、共通プロパティに追加して提供されています。

Show	コンポーネントの visible プロパティが、実行時に true に設定されました。
Hide	コンポーネントの visible プロパティが、実行時に false に設定されました。
progress	ProgressBar 値が変更されたときに作成されました。
complete	ProgressBar 値が最大値と同じ場合に生成されました。

以下の例はプログレス イベントをリッスンする方法を示しています:

```
myProgress.addEventListener("progress", this, "onProgress");
myProgress.addEventListener("complete", this, "onProgress");
function onProgress(event:Object) {
      if (event.type == "progress") {
            // ここに何かを追加する
      } else
      {
            trace("Loading complete!");
      }
}
```

2.6 その他のタイプ

また、Scaleform CLIK フレームワークは簡単に分類できない複数のコンポーネントでも構成されていますが、それらも UI 開発者に貴重な機能を提供しています。このようなコンポーネントには Dialog、UILoader、ViewStack が含まれます。Dialog コンポーネントはモーダルや非モーダル ダイアログを表示します。UILoader はコンテンツをロードする便利なインターフェイスを提供し、ViewStack は一連のフォームの管理に使用することができます。

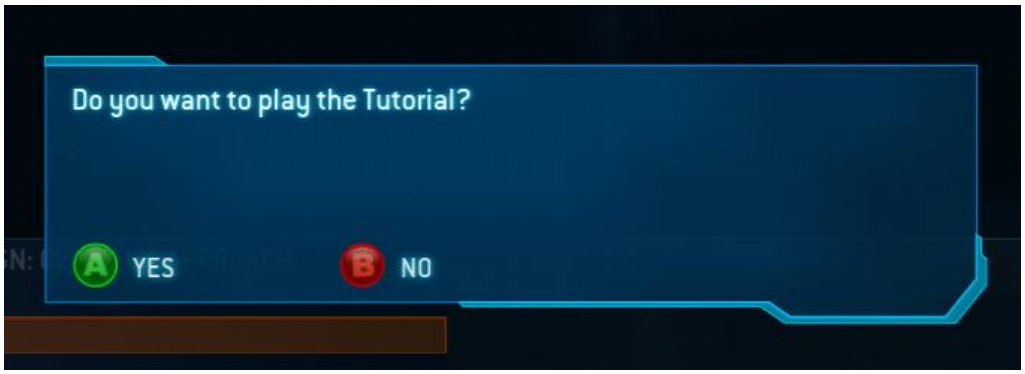

図 48: *Halo Wars* - ダイアログの例

2.6.1 Dialog

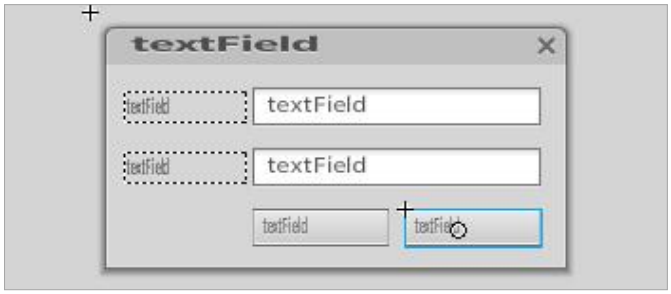

図 49: スキニング前の Dialog の例

CLIK Dialog コンポーネント (gfx.controls.Dialog) は警告ダイアログなどのダイアログを、アプリケーションのそれ以外の部分の上に表示します。このコンポーネントは、ダイアログとして任意の MovieClip のインスタンス化、表示、非表示を行うスタティック インターフェイスや、実際のダイアログ MovieClip として使用 (拡張) できるベース クラスを提供します。一度に 1 つのダイアログだけを確実に開くため、新規の Dialog.show() 呼び出しは、現在開いているダイアログを閉じます。

このプリビルド コンポーネントにはコンテンツは含まれていません。完全にユーザーが定義するものとして設計されているからです。ただし、コンポーネント シンボルを編集するだけでコンテンツは簡単に追加されます。

2.6.1.1 ユーザーの操作

Dialog のユーザー操作は、このコンポーネントが作成するダイアログ内で定義されます。

2.6.1.2 コンポーネントのセットアップ

CLIK Dialog クラスを使用する MovieClip には、以下の指定されたサブエレメントが必須です。オプションのエレメントである場合には、その旨表記しています:

- *closeBtn :* (オプション) CLIK Button のタイプ。ウィンドウの閉じるボタンに似ています。
- *cancelBtn :* (オプション) CLIK Button のタイプ。送信せずにダイアログを閉じるキャンセル ボタンです。
- *submitBtn :* (オプション) CLIK Button のタイプ。送信後にダイアログを閉じる OK ボタンです。
- *dragBar :* (オプション) MovieClip タイプ。ダイアログのドラッグ可能なタイトル バーです。

2.6.1.3 ステート

Dialog コンポーネントにはステートはありません。ダイアログとして表示される MovieClip は、独自のステートを持つ場合も、持たない場合もあります。

2.6.1.4 検証可能なプロパティ

Dialog コンポーネントから生じる MovieClip は、以下の検証可能なプロパティを備えています:

visible	false に設定した場合、このコンポーネントを非表示にします。
disabled	true に設定した場合、コンポーネントを無効にします。

2.6.1.5 イベント

すべてのイベント コールバックは、そのイベントの関連情報を含む 1 つの Object パラメータを受け取ります。以下のプロパティは、すべてのイベントに共通です。

- *type：*イベントのタイプ
- *target：*イベントを作成したターゲット

Dialog コンポーネントが作成するイベントは以下の表のとおりです。イベントの隣に表示されているプロパティは、共通プロパティに追加して提供されています。

Show	コンポーネントの visible プロパティが、実行時に true に設定されました。
Hide	コンポーネントの visible プロパティが、実行時に false に設定されました。
Submit	Dialog の送信ボタンが、クリックされました。 • *data：* Dialog の送信されたデータです。Dialog コンポーネントの getSubmitData メソッドはこのプロパティのために呼び出され、カスタムのダイアログ ボックスでオーバーライドする必要があります。返されるデフォルト値は true (ブール値) です。getSubmitData の戻り値は AS2 Object です。
Close	Dialog の「閉じる」ボタンが、クリックされました。

以下の例はダイアログ送信イベントを処理する方法を示しています:

```
myDialog.addEventListener("submit", this, "onLoginSubmit");
function onLoginSubmit(event:Object) {
        trace("Received data from dialog: " + event.data);
}
```

2.6.2 UILoader

CLIK UILoader (gfx.controls.UILoader) は外部の SWF/GFX、またはイメージをパスだけを使ってロードします。また、UILoader は境界ボックスに合わせるために、ロードされたアセットの自動サイズ変更もサポートします。アセットのロードは、Scaleform とそれを実行しているプラットフォームの両方がスレッドをサポートしている場合は非同期です。

2.6.2.1 ユーザーの操作

UILoader コンポーネントにはユーザーの操作はありません。SWF/GFX が UILoader にロードされた場合、このコンポーネントは独自のユーザーの操作を持つ可能性があります。

2.6.2.2 コンポーネントのセットアップ

CLIK UILoader クラスを使用する MovieClip は以下の指定されたサブエレメントが必須です。オプションのエレメントである場合には、その旨表記されています:

- *bg :* (オプション) MovieClip タイプ。UILoader の背景です。ステージ上の親コンポーネントの視覚的な表現以外、機能上の目的はありません。この背景は実行時に削除されます。

2.6.2.3 ステート

UILoader コンポーネントにはステートはありません。SWF/GFX ファイルが UILoader にロードされた場合、このコンポーネントは独自のステートを持つ可能性があります。

2.6.2.4 検証可能なプロパティ

UILoader コンポーネントから生じる MovieClip は、以下の検証可能なプロパティを備えています:

visible	false に設定した場合、このコンポーネントを非表示にします。
autoSize	true に設定された場合、UILoader の境界に合うように、ロードされたコンテンツのサイズを変更します。
maintainAspectRatio	true の場合、ロードされたコンテンツは、UILoader の境界内のアスペクト比を基準にして合わせられます。false の場合、コンテンツは UILoader の境界に合うように拡張されます。
Source	ロードする SWF/GFX やイメージのファイル名です。
timeout	ミリ秒単位でコンテンツがロードされるのを待ちます。タイムアウトに達しても、コンテンツがロードされていなかった場合、UILoader は 'ioError' イベントを作成します。

2.6.2.5 イベント

すべてのイベント コールバックは、そのイベントの関連情報を含む 1 つの Object パラメータを受け取ります。以下のプロパティは、すべてのイベントに共通です。

- *type :* イベントのタイプ
- *target :* イベントを作成したターゲット

UILoader コンポーネントが作成するイベントは以下の表のとおりです。イベントの隣に表示されているプロパティは、共通プロパティに追加して提供されています。

Show	コンポーネントの visible プロパティが、実行時に true に設定されました。
Hide	コンポーネントの visible プロパティが、実行時に false に設定されました。
progress	コンテンツをロードできるかできないかに関係なく、そのコンテンツはロード処理中です。このイベントは、a) コンテンツがロードされる、または b) ロードのタイムアウトに達するのいずれかになるまで、継続して発生します。

Loaded	ロード済みデータのパーセンテージです。このプロパティの値は 0 から 100 の間です。
complete	コンテンツのロードが、完了しました。
ioError	source プロパティで指定したコンテンツはロードできませんでした。

以下の例はロード エラーのときの通知方法を示しています:

```
myUILoader.addEventListener("ioError", this, "onLoadingError");
function onLoadingError(event:Object) {
      displayErrorMessage("Could not load" + event.target.source);
}
```

2.6.3 ViewStack

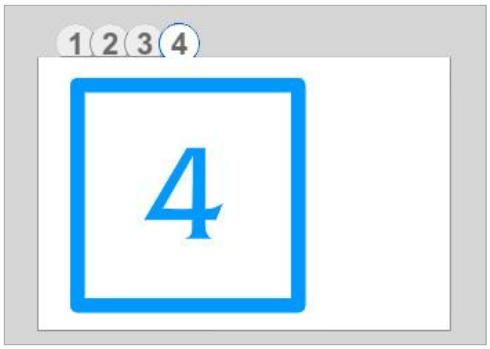

図 50: スキニング前の ViewStack

CLIK ViewStack (gfx.controls.ViewStack) は、ロードされオプションで内部的にキャッシュされたセットから、1 つのビューを表示します。このコンポーネントは TabBox や Accordion などのマルチビュー コンポーネントに使用することができます (この 2 つのコンポーネントのサンプルは、それぞれ demos フォルダにあります)。また、ViewStack は RadioButton グループなどの別のコンポーネントを指して、そのコンポーネントが変更されたときに、自動的にビューを変更することもできます。

2.6.3.1 ユーザーの操作

ViewStack コンポーネントにはユーザーの操作はありません。ViewStack でロードされ表示されるビューは、独自のユーザーの操作を備えている場合があります。

2.6.3.2 コンポーネントのセットアップ

CLIK ViewStack クラスを使用する MovieClip は、指定されたサブエレメントをいっさい必要としません。ただし、ロードされるコンテンツに合わせて、サイズ変更しなければなりません。

2.6.3.3 ステート

ViewStack コンポーネントにはステートはありません。ViewStack でロードされ表示されるビューは、独自のステートを備えている場合があります。

2.6.3.4 検証可能なプロパティ

ViewStack コンポーネントから生じる MovieClip は、以下の検証可能なプロパティを備えています:

visible	false に設定した場合、このコンポーネントを非表示にします。
Cache	true に設定された場合、ロードされたビューは内部でキャッシュされます。これはビューを作成する処理時間を節約しますが、不変の ViewStack targetGroup (下記参照) が必要になります。
targetGroup	ButtonGroup など、'change' イベントを作成する有効なグループ オブジェクト名です。このグループ オブジェクトの現在のエレメントは、ロードされ表示されるビューのリンケージ ID を含むデータ プロパティが必須です。たとえば、RadioButton は Flash IDE の [コンポーネント インスペクタ] でリンケージ ID を割り当てられるデータ プロパティを備えています。

2.6.3.5 イベント

ViewStack はいっさいイベントを作成しません。

3 アートの詳細

この章は Scaleform CLIK コンポーネントのスキンの開発でデザイナーを支援し、簡単なコンポーネント サンプルのスキニングの詳細、さらにベスト プラクティス、アニメーション、フォントの埋め込みについて説明しています。

3.1 ベスト プラクティス

この章には、Scaleform CLIK コンポーネントのスキンを作成するときの、ベスト プラクティスが含まれています。

3.1.1 ピクセル パーフェクトなイメージ

CLIK コンポーネントのベクター ベースのスキンを開発している場合、すべてのアセットをピクセル パーフェクトにすることをお勧めします。ピクセル パーフェクトなアセットとは、Flash グリッド上に完璧に並んでいるアセットです。

グリッドを有効にして変更する:

1. Flash メニューの [表示] を選択します。
2. [グリッド] を選択して [グリッドを表示] をオンにします。
3. Flash メニューの [表示] を再度選択します。
4. [グリッド] を選択し [グリッドの編集] を選択します。
5. 縦横のサイズを 1 px と入力します。
6. [OK] をクリックします。

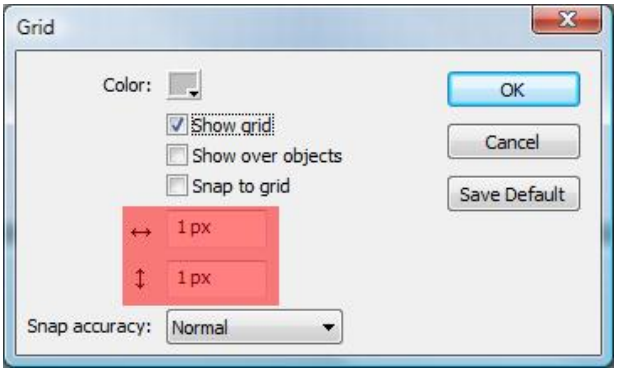

図 51: [グリッドの編集] のウィンドウ

ステージにグリッド オーバーレイが表示されているはずです。各グリッドは正確に 1 ピクセルを表しています。アート アセットを作成するときに、それらのアセットがこのグリッドに

吸着するようにしてください。これにより、最終的な SWF は確実にぼやけないようになります。注意: すべてのイメージサイズを 2 のべき乗に維持してください。そうしないと、設定してもぼやけてしまう場合があります。以下の 2 のべき乗の小見出しを参照してください。

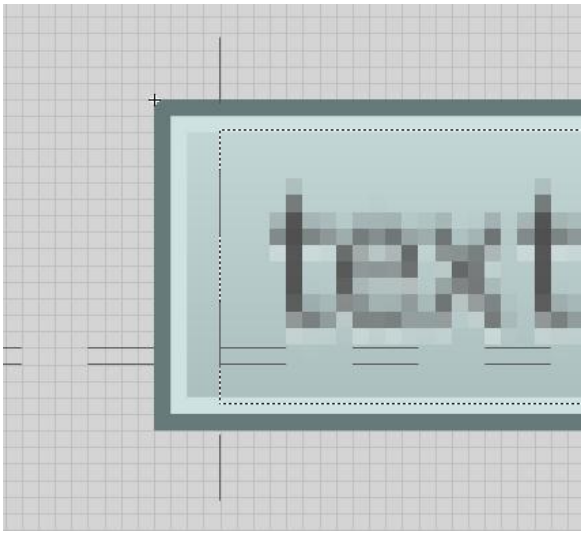

図 52: ピクセル パーフェクトなベクター グラフィック スキン

3.1.2 マスク

Flash のマスクを使うと、デザイナーは実行時にグラフィックを部分的に非表示にすることができます。マスクはアニメーション効果によく使用されますが、実行時に非常に負荷が高くなる場合があります。Scaleform は可能な限りマスクの使用を避けることをお勧めします。マスクを使用しなければならない場合、その数を一桁台に保ち、マスクありとマスクなしでムービーのパフォーマンスをテストしてください。

Flash でマスクの代わりに使用できるものとして、マスク アウトが必要な領域に、アルファブレンド付きの PNG を Photoshop®で作成することがあります。ただし、これは透明な領域のアニメーションがないイメージのみに有効です。

3.1.3 アニメーション

四角形を円に変形するなど、1 つのベクター シェイプを別のシェイプに変形するアニメーションは、避けておくのが最善の策です。そのシェイプをすべてのフレームで再評価しなければならないので、この種のアニメーションは非常に負荷が高くなります。

可能な場合は、アニメーションをベクター グラフィックのためにスケーリングすることを避けてください。追加のテッセレーション (ベクター シェイプを三角形に変換する処理) のため、メモリとパフォーマンスに影響を及ぼす可能性があるからです。最も負荷の低いアニメーションは変形と回転です。追加のテッセレーションが必要ないからです。

プログラム的なトゥイーンを避け、タイムライン ベースのトゥイーンを選択します。タイムライン トゥイーンのほうがパフォーマンスの点でははるかに負荷が低いからです。タイムライン トゥイーンの場合、キーフレーム数を希望するフレームレートで滑らかなアニメーションの実現に必要な最小限の数に維持してください。

3.1.4 レイヤーと描画プリミティブ

Flash でスキンを作成するときは、可能な限り使用するレイヤーの数を少なくします。使用されるレイヤーごとに最低でも 1 つの描画プリミティブが追加されます。使用される描画プリミティブの数が増えると、必要なメモリ サイズも増加し、パフォーマンスも急激に低下します。

3.1.5 複雑なスキン

複雑なスキンにはビットマップを使用するのが最善の方法ですが、簡単なスキンにはベクターグラフィックのほうがメモリの節約になり、画質の劣化 (ぼやけ) を起こさずに、どのような解像度にもスケーリングすることができます。

3.1.6 2 のべき乗

すべてのビットマップのサイズは 2 のべき乗を保つ必要があります。2 のべき乗のビットマップ サイズの例:

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

3.2 既知の問題と推奨ワークフロー

この章では、アートに関係する既知の Flash の問題と、Scaleform CLIK で作業する場合の推奨ワークフローを紹介しています。

3.2.1 コンポーネントの複製

Flash にはコンポーネントを複製するときの問題があり、このためリンケージ情報やコンポーネントの定義情報が、コピー元から新規のコンポーネントにコピーされなくなります。そのような場合、このリンケージ情報のないコンポーネントは機能しません。この章では、この問題を回避する 2 つの方法を紹介しています。

3.2.1.1 コンポーネントの複製 (方法 1)

Button などの未変更 (スキニング前の) CLIK コンポーネントを、外部の FLA ファイルからコピー先の FLA ファイルに複製する最も早い方法は以下のとおりです:

1. *CLIK_Components.fla* ファイルを開きます。
2. そのファイルの [ライブラリ] パネルからコンポーネント (Button など) を、そのコンポーネント上で右クリックして [コピー] を選択し、次にコピー先の FLA の [ライブラリ] パネルで右クリックして [ペースト] を選択してペーストします。
3. コピー先の FLA の [ライブラリ] パネルでそのコンポーネントを右クリックして [名前を変更] を選択し、'Button' 以外の名前に変更します。
4. 再度、コピー先の FLA の [ライブラリ] パネルでそのコンポーネントを右クリックして、[プロパティ] を選択します。
5. ステップ 3 でコンポーネントに選択した新規の名前に合致するように、[識別子] フィールドを変更します。
6. [ライブラリ] パネルの空白の領域で右クリックして、[ペースト] を選択します。元のコンポーネントの新規のコピーが、リンケージ情報はすべてそのままでペーストされます。

3.2.1.2 コンポーネントの複製 (方法 2)

同じライブラリにシンボル (コンポーネント) を複製するとき、リンケージ情報は新規の複製したシンボルにはコピーされません。そのコンポーネントを機能させるために、この情報を入力する必要があります。以下の手順に従います:

1. [ライブラリ] パネルで複製するコンポーネントを右クリックして、[プロパティ] を選択します。
2. [シンボルプロパティ] ウィンドウで、[クラス] テキストフィールドのテキスト (gfx.controlsButton など) をダブルクリックして強調表示し、キーボードで Ctrl+C キーをクリックしてコピーします。
3. [キャンセル] をクリックします。
4. 再度、複製するコンポーネントを右クリックして、[複製] を選択します。
5. [シンボルの複製] ウィンドウで [ActionScript に書き出し] チェックボックスをクリックしてオンにします。
6. [クラス] フィールドをクリックし、Ctrl+V キーを押してリンケージ情報をこのテキストフィールドにペーストします。
7. [OK] をクリックします。

8. [ライブラリ] パネルでこの新規にコピーしたコンポーネントを右クリックして、[コンポーネント定義] を選択します。
9. 空白の [クラス] フィールドをクリックし、Ctrl+V キーを押してリンケージ情報をこのテキストフィールドにペーストします。
10. [OK] をクリックします。

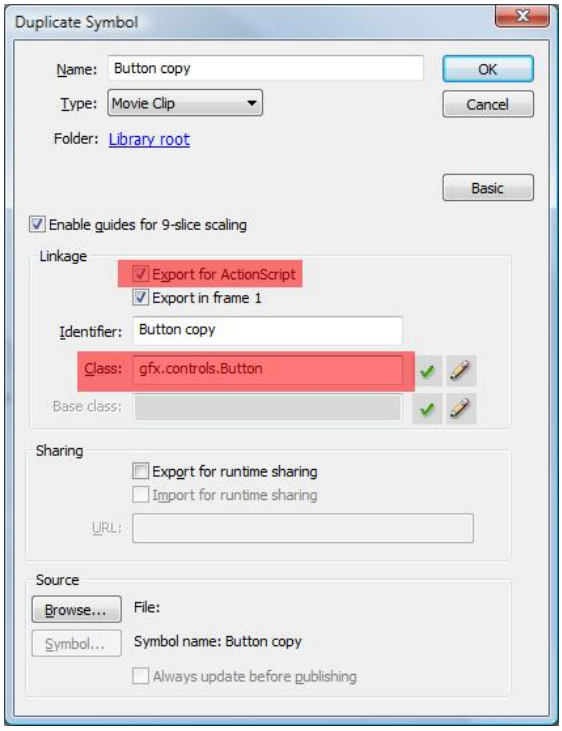

図 53: シンボルの複製ダイアログのリンケージ (赤で強調表示されている領域を設定する)

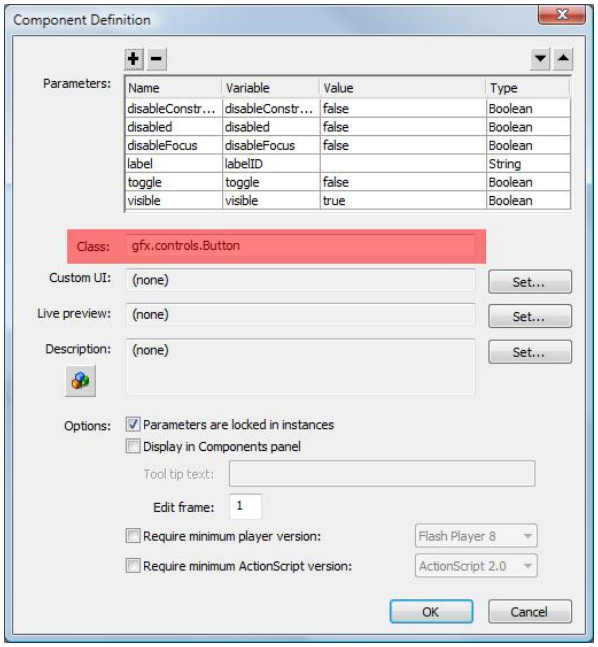

図 54: コンポーネント定義ダイアログ ([クラス] に入力する)

3.3 スキニングの例

Scaleform CLIK の大きな利点は、表示レイヤーと機能レイヤーを分けていることです。この分割により大部分は、プログラマとデザイナーがお互いに独立して作業できます。楽に外観と操作感をカスタマイズできることは、この分割の主要な利点の 1 つです。

プリビルド CLIK コンポーネントは標準の外観と操作感を備えていますが、ユーザーが自分のニーズに合わせてカスタマイズするように作られています。これ以降の章で、このようなプリビルド コンポーネントをカスタマイズする方法について説明します。

一連の CLIK コンポーネントは、標準の Flash シンボルと同じように簡単にスキニングされます。FLA の [ライブラリ] パネルでコンポーネントをダブルクリックし、そのコンポーネントのタイムラインにアクセスして以下のいずれかを行います:

- Flash でデフォルトのスキンを変更する。
- Flash でカスタムのスキンを始めから作成する。
- Photoshop や Illustrator®で作成したアート アセットを Flash にインポートする。

詳細なスキニングのチュートリアルについては、「Getting Started with CLIK」を参照してください。

3.3.1 StatusIndicator のスキニング

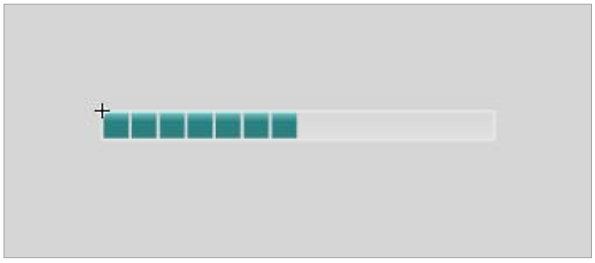

図 55: スキニング前の StatusIndicator

StatusIndicator は独特なコンポーネントで、Button をベースにしている他のほとんどのコンポーネントとは少し異なる方法が必要となります。StatusIndicator コンポーネントを開き、タイムラインを確認します。*indicator* と *track* という 2 つのレイヤーがあります。*track* は背景のグラフィックの視覚的な表示に使用されます。それ以外の目的には機能しません。*indicator* レイヤーは複数のキーフレームと、ビットマップか、ベクター グラフィックのいずれかを使用して、昇順で段階的にステータスを表現します。

このチュートリアルでは、1 つの Photoshop ファイルで作成した複数のビットマップを複数のレイヤーで使用して、ステータス インジケータを表現します。ここでは StatusIndicator をスキニングする 1 つの方法を説明していますが、ステージでグラフィックを作成し組み立てる方法は他にもあります。StatusIndicator が適切に動作するためには、最終的な結果は同じになる必要があります。

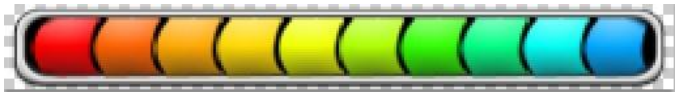

図 56: StatusIndicator の PSD ファイル

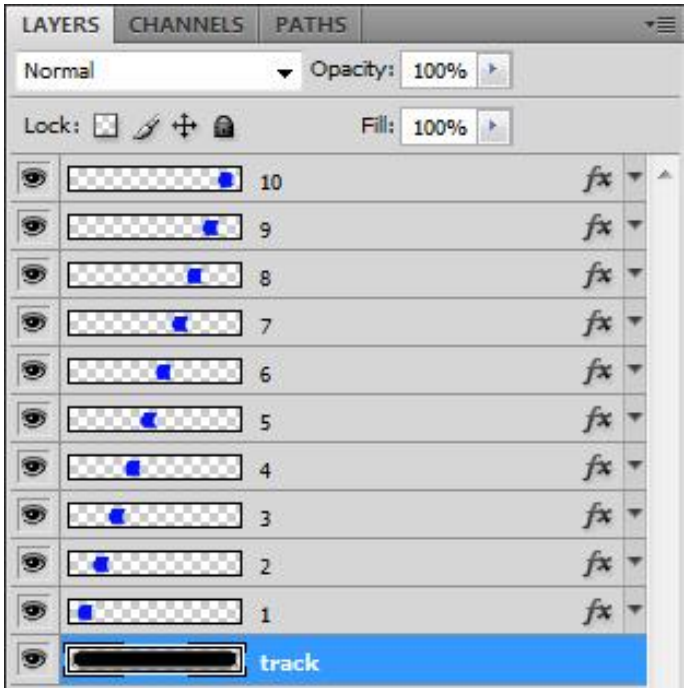

図 57: StatusIndicator PSD ファイルの Photoshop におけるレイヤー セットアップ

1. 上図のような StatusIndicator ビットマップを Photoshop で作成します。レイヤーの順序と番号の付け方、さらにレイヤー上の各イメージの位置に注意します。背景が透明であることを確認します。このチュートリアル用に同様のファイルを作成してください。
2. PSD として作ったファイルを保存します。
3. Flash でメニューの [ファイル] を選択し、[読み込み]->[ステージに読み込み] を選択します。
4. StatusIndicator のスキン PSD ファイル（先ほど保存しておいたもの）を参照して選択します。
5. [･･･に読み込む] ウィンドウで [レイヤーを次に変換] が [Flash レイヤー] に設定されていることを確認します。
6. [OK] をクリックします。
7. PSD ファイルの各レイヤーに対して、新規の Flash タイムライン レイヤーが作成されたはずです。上記のイメージの場合、PSD ファイルには 10 個のレイヤーがあるので、10 個のレイヤーが作成されました (フレームごとに 1 つ)。各レイヤーは 1 から 10 までのラベルが付き、1 が最下部にあり 10 が最上部にあります。[レイヤー1] を選択します。
8. ステージですべてのビットマップ イメージの周囲に、選択範囲を描画します。
9. イメージを以前のスキニング前のトラックの上に移動して、必要に応じてサイズ変更します。
10. [レイヤー1] の最初のキーフレームを選択して、フレーム 6 にドラッグします。

11. [レイヤー1] でフレーム 11 に新規のキーフレームを追加します。フレーム 11 で右クリックして [キーフレームの挿入] を選択します。
12. [レイヤー2] のビットマップを選択して、Ctrl+X キーを押してカットします。
13. [レイヤー1] のフレーム 11 の新規のキーフレームを選択して、Ctrl+Shift+V キーを押して [レイヤー1] で正確に同じ場所にそのビットマップを配置します。
14. 10 個のビットマップすべてを同じレイヤー (レイヤー1) に置くまで、正しいキーフレームでこの処理を繰り返します。後続の各ビットマップは、前回のキーフレームから 5 フレーム後のキーフレームにコピーされるはずです。[レイヤー2] のビットマップはキーフレーム 11 に、[レイヤー3] のビットマップはフレーム 16 に、[レイヤー4] のビットマップはフレーム 21 に、という具合にコピーされるはずです。10 個のイメージの PSD を使うと、最後のキーフレームはフレーム 51 になるはずです。
15. 空白のレイヤー (2–10) を削除してクリーンアップします。
16. 最後のビットマップ キーフレームの後にタイムライン上に追加のフレームがある場合、さらに 5 つのキーフレームを数えて、次に残りのフレームを選択し右クリックして [フレームの削除] を選択し削除します。このチュートリアルでは、最後のフレームはフレーム 55 になるはずです。
17. 以前の *indicator* レイヤーを選択して削除します。
18. 以前の *track* レイヤーを選択して削除します。インポートされた新規の *track* レイヤーを削除しないようにしてください。
19. [レイヤー1] を選択して、'indicator' という名前に変更します。
20. この *indicator* レイヤーと *track* レイヤーを *actions* レイヤーの下にドラッグします。*track* レイヤーが必ず *indicator* レイヤーの下になるようにします。

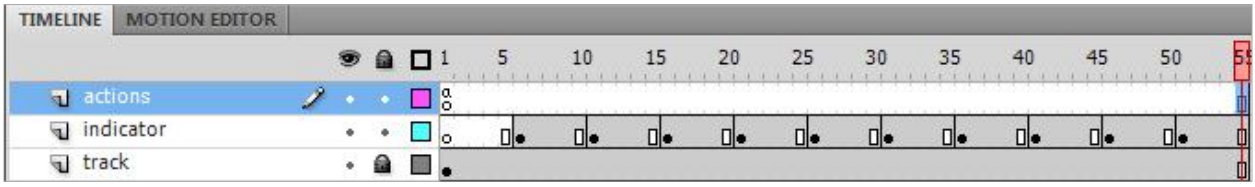

図 58: 最終的なタイムライン (キーフレームの位置と最後のフレームの位置に注意する)

21. StatusIndicator のタイムラインを終了します。
22. 検証可能パラメータの値を 1 から 10 までの任意の数に設定します。
23. このファイルを保存します。
24. ファイルをパブリッシュして、新規にスキニングされたインジケータを確認します。

図 59: スキニングされた StatusIndicator

3.4 フォントとローカライズ

この章では、Scaleform CLIK コンポーネントのフォントの使用について説明します。

3.4.1 はじめに

フォントを Scaleform で正しく表示するためには、必要なグリフ (フォントの文字) を SWF に埋め込むか、または Scaleform のローカライズ システムを使ってセットアップする必要があります。これ以降の章では、Flash UI でフォントを管理する方法について説明します。

3.4.2 フォントの埋め込み

Flash Player とは異なり、Scaleform でフォントを適切に表示するには、そのフォントを埋め込む必要があります。フォントが埋め込まれていない場合、それらのフォントがシステムに存在するとしても、Scaleform Player は空白の四角形を表示します。この表示の利点は、フォントが Scaleform に正しく埋め込まれていることを簡単に確認できることです。プリビルド コンポーネント セットでは、Slate Mobile が各 textField インスタンスに埋め込まれています。Slate Mobile には中国語、日本語、韓国語 (CJK) の文字、またはグリフは一切含まれていません。プリビルド コンポーネントは ASCII グリフしか埋め込んでいないので注意してください。Slate Mobile を CJK グリフを含むフォントに置き換えて、適切な埋め込みオプションを設定して変更することができます。

フォントを直接 textField に埋め込むことは、Scaleform のローカライズ システム (3.4.4 章を参照してください) とは互換性がないので注意してください。Scaleform は実際には、Scaleform のローカライズ システムを使ってフォントをセットアップすることをお勧めしています。直接埋め込むよりも多くの利点を備えているからです。ただし、ローカライズが必要ない場合など、フォントの埋め込みのほうが優れている場合もあります。UI を管理するのと同様に、フォントの管理もローカライズやメモリの管理など、複数の考慮点があります。

3.4.3 textField にフォントを埋め込む

フォントを埋め込む必要があるのは、ダイナミック、またはインプット textField だけです。スタティック テキストは、コンパイルすると自動的にアウトライン (未加工のベクター シェイプ) に変換されます。フォントをダイナミック textField に埋め込むには、ステージでその textField を選択して、[プロパティ インスペクタ] ([ウィンドウ]->[プロパティ インスペクタ]) の [埋め込み] ボタンをクリックします。埋め込む必要のある文字、または文字セットを選択して、[OK] を選択します。これらの手順を完了すると、そのフォントは FLA で使用できるようになります。同じフォントが同じ FLA の複数の textField で使用される場合は、この方法でそのフォントを埋め込むのは一度だけです。逆に、何回もそのフォントを埋め込んでもフ

ァイル サイズやメモリの使用量は増えません。中国語など多くのグリフを含む文字セットは、ロードされたときに大量のメモリを使用する可能性があるので注意してください。

3.4.4 Scaleform のローカライズ システム

Scaleform のローカライズ システムは現在のロケールが変更になるたびに、フォント ライブラリをロード/アンロードするホット スワップ メカニズムを使用しています。このようなフォント ライブラリ自体は、コンテンツ SWF が使用する埋め込まれたフォント グリフを含んだ SWF ファイルです。また、Scaleform はこのフォント ライブラリのグリフを使って、ダイナミックにオンデマンドでフォントを変更する効果的な方法を提供します。

Scaleform のローカライズ システムは、textField が直接グリフを埋め込んでいる場合はフォントを交換できないので、その textField はインポートされたフォント シンボルを使用する必要があります。通常、フォント シンボルは gfxfontlib.fla というファイルに作成され、コンテンツ SWF にインポートされます。このインポートされたフォントは、フォントの交換をサポートするすべての textField 上に設定されます。Scaleform は、このフォント シンボル リンクを乗っ取って、現在のロケールに基づいた別のフォントをロードできるようになっているのです。

フォント ライブラリはフォントをエクスポートする必要はありません。該当するフォントを埋め込む必要があるだけです (フォントを埋め込む方法について、前の章を参照してください)。Scaleform のローカライズ システムは fontconfig.txt ファイルを使って、ロケールごとに使用するフォント ライブラリを定義し、さらに textField が使用するフォント シンボルとフォント ライブラリに埋め込まれている物理フォント間のフォント マッピングを定義します。

また、fontconfig.txt ファイルには変換マップも含まれています。Scaleform のローカライズ システムはオンザフライで、ステージ上のすべてのダイナミック、またはインプット textField に表示されるテキストの変換を実行します。たとえば、textField に '$TITLE' というテキストが含まれていて、変換マップには $TITLE=Scaleform などのマッピングがある場合、この textField は自動的に 'Scaleform' と表示します。

この章で提示している情報は、Scaleform フォントとローカライズ システムの大まかな概要を示したものです。Scaleform におけるフォントとローカライズを完全に理解するには、「Font Overview」を参照してください。

4 プログラミングの詳細

この章では、フレームワークの要点を説明し、各サブシステムに焦点を当てて Scaleform CLIK コンポーネントのアーキテクチャを大まかに理解できるようにします。

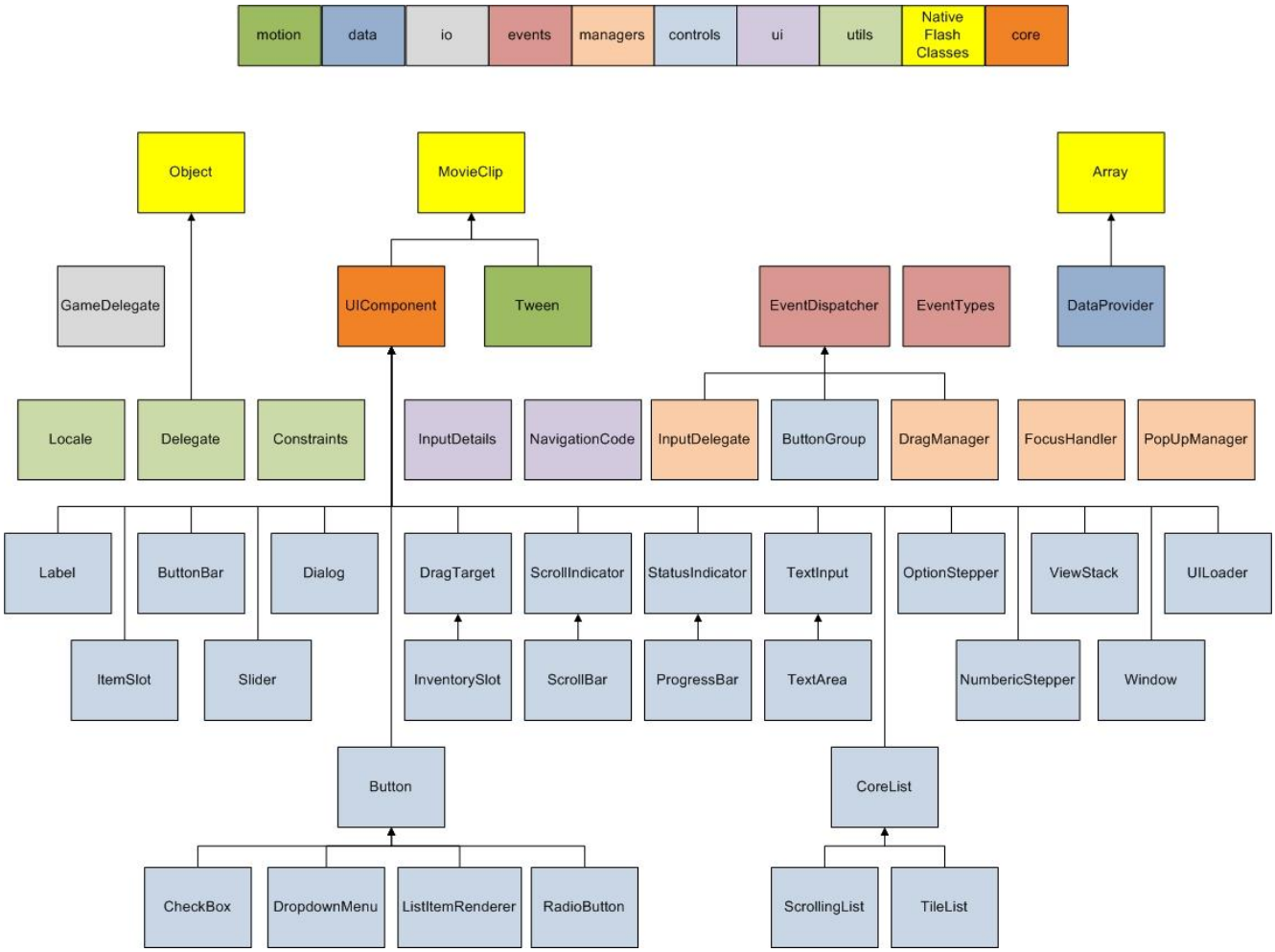

図 60: Scaleform CLIK クラスの階層

4.1 UIComponent

「プリビルド コンポーネント」の章では、Scaleform CLIK にバンドルされた具体的なコンポーネントが使用するクラスについて説明しました。それらのコンポーネントはすべて、UIComponent クラス (gfx.core.UIComponent) からコア機能を継承しています。この UIComponent クラスはすべての CLIK コンポーネントの基礎となっていますので Scaleform は、カスタムのコンポーネントを作成される場合には、UIComponent からサブクラス化することをお勧めします。

UIComponent クラス自体は Flash 8 MovieClip クラスから派生したものなので、標準の Flash MovieClip のプロパティやメソッドをすべて継承しています。UIComponent クラスは以下のカスタムの読み取り/書き込みプロパティをサポートします:

- *disabled;*
- *visible;*
- *focused;*
- *width;*
- *height;*
- *displayFocus :* コンポーネントが自らの focused ステートを表示する必要がある場合、このプロパティを true に設定します。詳細は「フォーカスの処理」のセクションを参照してください。

UIComponent にはサブクラスが実装する予定の以下のような空のメソッドが含まれています:

- *configUI :* コンポーネントの構成を行うために呼び出されます。
- *draw :* コンポーネントが無効になった場合に呼び出されます。詳細は「無効化」の章を参照してください。
- ***changeFocus :*** コンポーネントにフォーカスが適用されたとき、またはフォーカスを失ったときに呼び出されます。
- ***scrollWheel :*** カーソルがコンポーネント上にあるときに、マウス ホイールを使用すると呼び出されます。

UIComponent は Mix-in を使用して EventDispatcher クラスを継承し、イベントのサブスクライブとディスパッチをサポートします。したがって、すべてのサブクラスはイベントのサブスクライブとディスパッチをサポートします。
詳細は「イベントモデル」の章を参照してください。

4.1.1 初期化

このクラスは以下の初期化の手順を、onLoad() イベント ハンドラの内部で行います:

- MovieClip のサイズにデフォルト サイズを設定する。

- コンポーネントの構成を行う。この手順は configUI() メソッドを呼び出します。
- コンテンツを描画する。この手順は validateNow() メソッドを呼び出します。このメソッドは再描画を直ちに行います。つまり、draw() を呼び出します。

4.2 コンポーネント ステート

ほぼすべての Scaleform CLIK コンポーネントは視覚的なステートをサポートします。ステートは、そのコンポーネントのタイムラインで特定のキーフレームに移動する、またはステート情報をサブエレメントに渡すことで設定されます。以下のような 3 種類の一般的なステートの設定があります。

4.2.1 ボタン コンポーネント

マウスの操作に反応したり、オプションで選択可能であったりする、ボタンのように動作するコンポーネントはすべてこのカテゴリーに分類されます。このスキーマを使った CLIK コンポーネントの例は、Button とそのバリエーションである ListItemRenderer、RadioButton、CheckBox です。ScrollBar などの複合コンポーネントのサブエレメントもこのボタン コンポーネントです。このカテゴリーに分類される CLIK コンポーネントは、直接 Button クラス (gfx.controls.Button) を使用するか、または Button クラスから派生したクラスを使用するか、のいずれかであることになります。

ボタン コンポーネントがサポートする基本ステートは以下のとおりです:

- *up* またはデフォルトのステート
- マウス カーソルがコンポーネント上にあるとき、またはフォーカスが適用されているときの *over* ステート
- マウス ボタンがコンポーネント上で押されたときの *down* ステート
- コンポーネントが無効になったときの *disabled* ステート

また、ボタン コンポーネントはプレフィックスのステートもサポートします。これは他のプロパティの値に応じて設定することができます。デフォルトでは、コア CLIK Button コンポーネントは "selected_" というプレフィックスしかサポートしません。このプレフィックスはコンポーネントが、選択された状態のときにフレーム ラベルに付加されます。

ボタン コンポーネントがサポートする基本ステートは、selected ステートも含めると、以下のとおりになります:

- *up* またはデフォルトのステート
- マウス カーソルがコンポーネント上にあるとき、またはフォーカスが適用されているときの *over* ステート
- マウス ボタンがコンポーネント上で押されたときの *down* ステート
- コンポーネントが無効になったときの *disabled* ステート

- *selected_up* またはデフォルトのステート
- マウス カーソルがコンポーネント上にあるとき、またはフォーカスが適用されているときの *selected_over* ステート
- ボタンが押下されたときの *selected_down* ステート
- *selected_disabled* ステート

注意: この章で説明しているステートは、CLIK ボタン コンポーネントがサポートする全ステートのほんの一握りです。ステートの全リストについては、「Getting Started with CLIK Buttons」を参照してください。

CLIK Button クラスは getStatePrefixes() メソッドを提供します。このメソッドを使うと、開発者はコンポーネントのプロパティに応じてプレフィックスのリストを変更することができます。このメソッドは以下のように定義されます:

```
private function getStatePrefixes():Array {
      return (_selected) ? ["selected_",""] : [""];
}
```

前頁に記載しているとおり、デフォルトでは CLIK のボタンは "selected_" というプレフィックスしかサポートしていません。getStatePrefixes() メソッドはその selected プロパティに応じて、別の配列のプレフィックスを返します。このプレフィックスの配列は、適切なステート ラベルと組み合わせて内部で使用され、再生するフレームを決定します。

ステートが内部で設定されると、たとえばマウスのロールオーバーのときは、ルックアップテーブルが、フレーム ラベルのリストのために照会されます。Button クラスの stateMap プロパティは、フレーム ラベルのマッピングにステートを定義します。以下は CLIK Button クラスで定義されたステート マッピングです:

```
private var stateMap:Object = {
      up:["up"],
      over:["over"],
      down:["down"],
      release: ["release", "over"],
      out:["out","up"],
      disabled:["disabled"],
      selecting: ["selecting", "over"],
      kb_selecting: ["kb_selecting", "up"],
      kb_release: ["kb_release", "out", "up"],
      kb_down: ["kb_down", "down"]
}
```

各ステートは複数のターゲット ラベルを備えている場合もあります。このステート マップから返される値を getStatePrefixes() が返すプレフィックスと組み合わせて、再生されるターゲット フレームのリストを作成します。以下の図は、再生する正確なフレームを決定する場合に使用される全プロセスを表したものです:

図 61: 再生される正しいキーフレームの決定に使用されるプロセス

再生ヘッドは常に最後の使用可能なフレームにジャンプするので、プレフィックスを持つ特定のフレームが使用できない場合、必然的に、コンポーネントは前回リクエストされたフレームをデフォルトにします。開発者はこのステート マップをオーバーライドして、カスタムの動作を作成することができます。

4.2.2 ボタン以外のインタラクティブ コンポーネント

このカテゴリーはインタラクティブで、フォーカスを適用することはできてもマウス イベントには応答しないコンポーネントをすべて含みます。このスキーマを使用する CLIK コンポーネントの例として、ScrollingList、OptionStepper、Slider、TextArea が挙げられます。このようなコンポーネントには、マウス イベントに応答する子エレメントが含まれている場合があります。ボタン以外のインタラクティブ コンポーネントがサポートするステートは以下のとおりです:

- *default* ステート
- *focused* ステート
- *disabled* ステート

4.2.3 非インタラクティブ コンポーネント

インタラクティブではなくても、無効にすることができるコンポーネントはすべてこのカテゴリーに含まれます。Label コンポーネントは、ステートをサポートするデフォルト コンポーネント セットの唯一の非インタラクティブ コンポーネントです。非インタラクティブ コンポーネントがサポートするステートは以下のとおりです:

- *default* ステート

- *disabled* ステート

4.2.4 特殊なケース

上記のステートルールに従わないコンポーネントもあります。StatusIndicator とそのサブクラスの ProgressBar はタイムラインを使ってコンポーネントの値を表示します。再生ヘッドはコンポーネントのパーセント値を表すフレームに設定されます。たとえば、最小値 0、最大値が 10 で現在の値が 5 (50%) に設定されている StatusIndicator の 50 フレームのタイムラインは、フレーム 25 (50 フレームの 50%) に gotoAndStop() されます。これらのコンポーネントを拡張して、表示をプログラム的に管理するのは非常に簡単です。この動作を変えるように updateValue() メソッドを変更、またはオーバーライドすることができます。

或る場合には、コンポーネントは、デフォルトの動作に加えて、追加のコンポーネント ステートをサポートするような特別なモードを持っていることもあります。TextInput と TextArea コンポーネントでは、***over*** と ***out*** ステートが有効になる場合があります。これは、TextInput または TextArea コンポーネントの actAsButton プロパティが、フォーカスされていない時にマウスカーソルのロールオーバーとロールアウトをサポートするように設定されている場合です。

4.3 イベント モデル

Scaleform CLIK コンポーネント フレームは「イベント モデル」と呼ばれる通信パラダイムを使用します。コンポーネントは変更、または操作されるときに、イベントを「ディスパッチ」して、コンテナ コンポーネントは別のイベントにサブスクライブすることができます。これにより、1 つだけでなく複数のオブジェクトに変更が通知されます。これが ActionScript 2 コールバックが動作する方法です。

EventDispatcher クラス (gfx.events.EventDispatcher) は、このイベント モデルをサポートする使いやすい API を提供します。CLIK コンポーネントはこのクラスを拡張するか、または EventDispatcher.initialize() メソッドを使うかのいずれかでその動作を組み合わせることができます。ただし、CLIK コンポーネントが UIComponent から派生した場合、そのコンストラクタで super() を呼び出すだけです。UIComponent はすでに EventDispatcher との組み合わせを行っているからです。EventDispatcher サブクラス、または組み合わせは、イベントのサブスクライブとディスパッチをサポートします。

4.3.1 使用法とベスト プラクティス

4.3.1.1 イベントのサブスクライブ

イベントをサブスクライブするには、リッスンする操作のタイプを指定するタイプ パラメータと共に addEventListener() メソッドを使用します。逆に、removeEventListener() はイベントからアンサブスクライブします。複数のリスナーが正確に同じパラメータと共に追加さ

れる場合、1 つのイベントのみが発生します。また、このようなメソッドはそれぞれ、リッスン オブジェクトであるスコープ パラメータと callBack も必要とします。callBack はイベントがディスパッチされるときに呼び出される関数の String 名です。

EventTypes クラス (gfx.events.EventTypes) には共通で使用されるイベントの一覧が含まれます。これはイベント タイプを表す文字列の代わりに使用できます ("show" ではなく EventTypes.SHOW)。

```
buttonInstance.addEventListener("itemClick", this, "callBack");
function callBack(eventObj:Object):Void {
      buttonInstance.removeEventListener("itemClick", this, "callBack");
}
```

この例では buttonInstance は "itemClick" イベントをリッスンして、そのイベントを受け取ったときに "callback" 関数を実行するように設定されています。この callback 関数は次にそのイベント リスナーを削除します。

event Object のプロパティは、その起源とタイプによって異なります。CLIK コンポーネントが作成するイベント オブジェクト (およびそのプロパティ) の全リストについては、「プリビルド コンポーネント」を参照してください。

4.3.1.2 イベントのディスパッチ

サブスクライブされたリスナーに変更や操作を通知したい CLIK コンポーネントは、dispatchEvent() メソッドを使用します。このメソッドは 1 つの引数を必要とします。それは、ディスパッチされるイベントのタイプを指定する必須タイプ プロパティを含めて、関連データが含まれたオブジェクトです。コンポーネント フレームワークは自動的に target プロパティを追加します。これはそのイベントをディスパッチするオブジェクトへのリファレンスですが、手動でカスタム ターゲットでオーバーライドするように設定することもできます。

```
dispatchEvent({type:"itemClick", item:selectedItem});
```

4.4 フォーカスの処理

Scaleform CLIK コンポーネントはカスタムのフォーカス処理フレームワークを使用します。これはほとんどのコンポーネントに実装され、非フレームワーク コンポーネントやシンボルとうまく動作するはずです。すべてのフォーカスの変更は Scaleform Player レベルで、マウス、またはキーボード (コントローラ) によるフォーカスの変更、あるいは ActionScript で Selection.setFocus(instance) を呼び出すことのいずれかで行われます。FocusHandler マネージャ (gfx.managers.FocusHandler) は、1 つのコンポーネント クラスが作成されるとすぐにインスタンス化されます。FocusHandler を直接インスタンス化する必要はありません。

4.4.1 使用法とベスト プラクティス

現在、フォーカスは CLIK コンポーネント インスタンス上に設定する必要があります。そうでない場合、フォーカスのデフォルトは Player になります。設定されていない場合、フォーカス管理システムは正しく動作しません (たとえば、Tab キーはフォーカスを切り替えなくなります)。フォーカスは、コンポーネント上の focused プロパティを true に設定することで適用することができます。非コンポーネント エレメントでも動作するフォーカスの適用を行う他の方法は以下のとおりです:

```
Selection.setFocus(firstComponentOrMovieClip);
```

マウス ハンドラのない MovieClip (onRelease、onRollOver など) は、Scaleform や Flash Player でフォーカスを適用できず、フォーカスの変更イベントを作成しません。ボタン コンポーネントは自動的にマウス ハンドラを追加します。他のコンポーネントは、標準の MovieClip focusEnabled と tabEnabled プロパティの組み合わせを設定します。

ScrollBar などのフォーカス可能なサブエレメントを持つコンポーネントは、自らの focusTarget プロパティを使って、自身のオーナー コンポーネントにフォーカスを渡します。フォーカスがコンポーネントに変更されると、FocusHandler は再帰的に focusTarget チェーンを検索して、focusTarget を返さない最後のコンポーネントにフォーカスを適用します。

場合によっては、コンポーネントが実際には Player やアプリケーションのフォーカスではないときに、そのコンポーネントをフォーカスが適用されているように表示する必要があります。displayFocus プロパティを true に設定して、フォーカスが適用されているかのように動作することを、コンポーネントに指示することができます。たとえば、Slider にフォーカスが適用されているとき、Slider のトラックもフォーカスが適用されているように見える必要があります。UIComponent.changeFocus() メソッドが、focused プロパティが変更されると、コンポーネント上に呼び出されるので注意してください。

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

逆に、場合によってドラッグ可能なパネル、またはマウスのみのコントロールが有効なその他のコンポーネントなど、コンポーネントはクリック可能でもフォーカスが適用できないようにする必要があります。この場合、tabEnabled プロパティを false に設定します。

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

4.4.2 複合コンポーネントのフォーカスの取得

複合コンポーネントは、ScrollingList、OptionStepper、または ButtonBar など、異なるコンポーネントで構成されたコンポーネントのことです。このコンポーネント自体はサブコンポーネントにマウス ハンドラを備えていても、自分自身のマウス ハンドラはいっさい持っていないことがよくあります。つまり Flash や Scaleform の Selection エンジンはアイテムにフォーカスを適用できず、内蔵のナビゲーション サポートはコンポーネントの識別に問題があり、誤ってその子コンポーネントを検証してしまうということになります。

複合であるにもかかわらず、単独のエンティティとしてそのコンポーネントに動作させるには、以下の手順を行います:

1. マウス ハンドラを持つすべてのサブコンポーネント (OptionStepper の矢印ボタンなど) で tabEnabled = false と設定します。
2. マウス ハンドラを持つすべてのサブコンポーネントで focusTarget プロパティを、コンテナ コンポーネントに設定します。必ず、コンテナ コンポーネントで focusEnabled = true と設定します。
3. 必要な場合、サブコンポーネントの displayFocus プロパティを、コンテナ コンポーネントの changeFocus メソッド内で、true に設定します。

これで、サブコンポーネントにフォーカスが適用されると、そのフォーカスはコンテナ コンポーネントに移動するようになります。

4.5 入力処理

Scaleform CLIK コンポーネントは Flash 8 MovieClip クラスから派生しているので、ユーザーの入力を受け取ると、他の MovieClip インスタンスとほぼ同じように動作します。コンポーネントがマウス ハンドラを備えていれば、マウス イベントはコンポーネントがキャッチします。ただし、キーボードや同等のコントローラ イベントの処理方法に関しては、CLIK には大きな概念上の変更があります。

4.5.1 使用法とベスト プラクティス

マウス以外の入力はすべて、InputDelegate マネージャ クラス (gfx.managers.InputDelegate) がインターセプトします。InputDelegate は内部で、またはゲーム エンジンから入力コマンドの値をリクエストするかのいずれかで、入力コマンドを InputDetails オブジェクト (gfx.ui.InputDetails) に変換します。後者には、InputDelegate.readInput() メソッドをターゲット アプリケーションをサポートするように変更することが関係します。いったん InputDetails が作成されると、入力イベントは InputDelegate からディスパッチされます。

InputDetails は以下のプロパティで構成されます:

- 「キー」などのタイプ
- 押下されたキーのキー コードなどのコード
- ボタンのベクトルやキー押下のタイプなど入力についての追加情報である値。キーイベントは、キーが離される（key up）とキーが押される（key down）というアクションのどちらにも生成されます。また、相応する InputDetails のパラメータ値は、それぞれ、“keyUp”または“keyDown”となります。
- 「上」、「下」、「左」、「右」などのマッピングが可能な場合、そのような人が解読できる移動方向を定義する navEquivalent (ナビゲーション相当操作)。NavigationCode クラス (gfx.ui.NavigationCode) は、一般的なナビゲーション相当操作の便利な一覧を提供します。

FocusHandler は InputDelegate の入力イベントをリッスンして、そのようなイベントをフォーカス パスを通じてコンポーネントに渡します。フォーカスが適用されたコンポーネントを使って、フォーカス パスを決定します。このパスは handleInput() メソッドを実装した、表示リスト階層の上から順に並んだコンポーネントのリストです。

このメソッドはフォーカス チェーンの最上位のコンポーネントで呼び出され、InputDetails と pathToFocus 配列がパラメータとして渡されます。

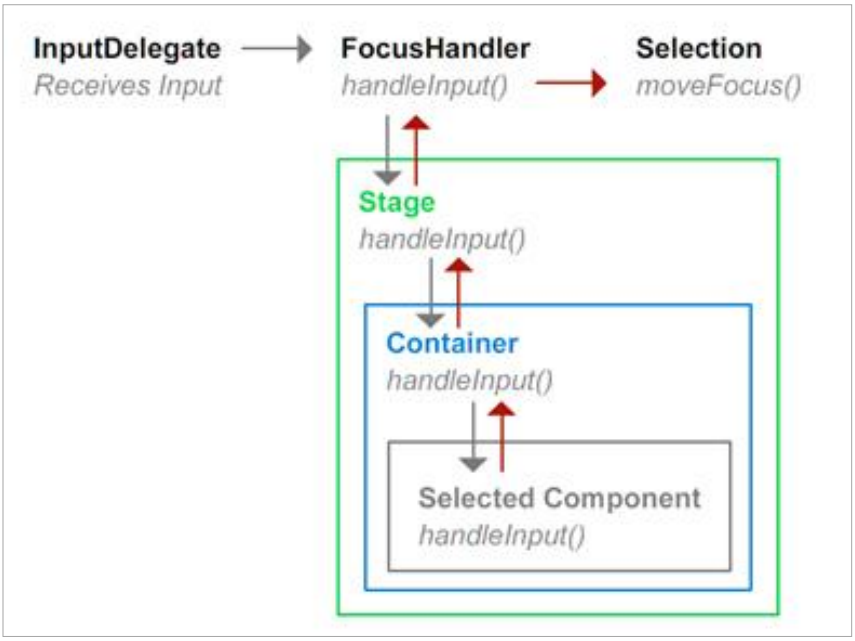

図 62: handleInput() チェーン

FocusHandler は handleInput() 呼び出しに対してブール値の応答を期待します。この値はそのコンポーネント、またはフォーカス パスの任意のコンポーネントが入力を処理したかどうかを表します。false という応答があり、その入力が null ではない navEquivalent を持っている場合、その入力は Scaleform Player に渡されます。

```
function handleInput(details:InputDetails, pathToFocus:Array):Boolean {
```

```
            if (details.navEquivalent == "left")
      {     // または NavigationCode.LEFT
                  doSomething();
                  return true;
            }
            return false;
      }
```

イベントを pathToFocus 配列の次のコンポーネントにバブルして、入力が処理された場合に true か false を返すのは、その入力を処理する各コンポーネント次第です。フォーカス パスの次のコンポーネントが handleInput を正しく実装すると仮定しないほうが良いので、このメソッドはバブルされた入力の戻り値だけでなく、ブール値を返すことを保証するべきです。

```
      function handleInput(details:InputDetails, pathToFocus:Array):Boolean {
            var nextItem:MovieClip = pathToFocus.shift();
            var handled:Boolean = nextItem.handleInput(details, pathToFocus);
            if (handled) { return true; }
            // カスタムの処理コード
            return false; // または、処理された場合は true
      }
```

また、入力はフォーカス パスにはないコンポーネントに渡すこともできます。たとえば、pathToFocus 配列にないとしても、DropdownMenu では handleInput はドロップダウン リスト コンポーネントに渡されます。これにより、そのリストはキー コマンドに応答することができます。

```
      function handleInput(details:InputDetails, pathToFocus:Array):Boolean {
            var handled:Boolean = dropdown.handleInput(details);
            if (handled) { return true; }
            // カスタムの処理コード
            return false; // または、処理された場合は true
      }
```

また、イベント リスナーを InputDelegate.instance に追加することで、手動で入力イベントをリッスンし、入力を以下のように処理することも可能です。入力はそれでも FocusHandler に取得され、フォーカス階層を下に移動するので注意してください。

```
      InputDelegate.instance.addEventListener("input", this, "handleInput");
      function handleInput(event:Object):Void {
            var details:InputDetails = event.details;
            if (details.value = Key.TAB) { doSomething(); }
      }
```

InputDelegate は予想される入力を管理するために、ゲームに合わせて調整する必要があります。デフォルトの InputDelegate はキーボード コントロールを処理して、矢印キー、さら

に一般的なゲームのナビゲーション キーである W、A、S、D を方向性のあるナビゲーションの相当操作に変換します。

4.5.2 複数のマウス カーソル

Wii などの一部のシステムは複数のカーソル デバイスをサポートします。CLIK コンポーネント フレームワークは複数のカーソルをサポートしますが、1 つの SWF では複数のコンポーネントにフォーカスを適用することはできません。二人のユーザーが別々のボタンをクリックした場合、最後にクリックされたアイテムにフォーカスが適用されます。

このフレームワークでディスパッチされたすべてのマウス イベントは、mouseIndex プロパティを含んでいます。これはそのイベントを作成した入力デバイスのインデックスです。

```
myButton.addEventListener("click", this, "onCursorClick");
function onCursorClick(event:Object):Void {
      switch (event.mouseIndex) {
      …
      }
}
```

4.6 無効化

無効化は、複数のプロパティが変更になったときに、コンポーネントが再描画する回数を制限するメカニズムです。コンポーネントが無効化されると、自らを次のフレームに再描画します。これにより、開発者は一度に多くの変更をコンポーネントに適用して、一度だけコンポーネントを更新させることができます。コンポーネントが直ちに再描画する必要がある場合などわずかに例外もありますが、ほとんどの場合、無効化で十分です。

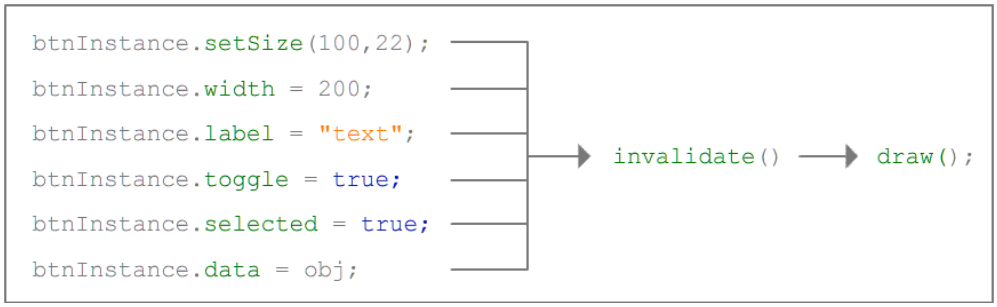

図 63: 特定のプロパティが変更になったときに、CLIK コンポーネントは自動的に無効化する。

4.6.1 使用法とベスト プラクティス

内部のコンポーネントの変更後に (通常 setter 関数が行います)、UIComponent.invalidate() を呼び出す必要があります。これは最終的にコンポーネントの UIComponent.draw() を呼び出します。invalidate() メソッドは、タイマー ベースの遅延を

使って draw() 呼び出しを生成して、不要な更新を避けます。既存のコンポーネントを使用している開発者は無効化を使う必要はありませんが、少なくとも認識しておくべきです。

直ちに再描画する必要がある場合、開発者は UIComponent.validateNow() メソッドを使用することができます。

4.7 コンポーネントのスケーリング

Scaleform CLIK コンポーネントは以下の 2 つの方法でスケーリングを行います:

1. リフロー レイアウトを使用する。この場合コンポーネントのスケールはリセットされ、そのエレメントはオリジナルのサイズに合うようにサイズ変更されます。サブエレメントを含み、バックグランドのないコンポーネントはこの方法を使用します。
2. アスペクト比を維持するようにエレメントを逆にスケーリングする。この場合コンポーネントはスケーリングされたままですが、そのエレメントは逆にスケーリングされ、スケーリングされていないように見せます。この方法はグラフィック バックグラウンドと scale9grid を持つコンポーネントを拡大して、歪めることなく、コンテキストを内部でスケーリングすることができます。

一般的にコンポーネントには、Flash のスケール 9 グリッドに制約があることから、リフローの手法を使います。このため、開発者は Flash や Flex のコンポーネントと同じように、“スキン”のシンボルを作らなければなりません。リフローは、コンポーネントが同じようにスケールするサブコンポーネントを持っていれば上手く働きます。つまり、リフローは、コンテイナーやこれに似たエンティティに向いているということになります。

逆スケーリングの手法は、Scaleform が拡張的なスケール 9 グリッド機能を持っていることから、特に CLIK のためのようなものです。これは、ある種のコンポーネントに使われているようなレイヤーに集中したアプローチではなく、フレーム ステートを使った単一アセットを持つコンポーネントの作成に使うことができます。基本の CLIK コンポーネントは、もとより必要最小限に作られていて、通常は、背景一つ、ラベル一つ、オプションのアイコンまたはサブ・ボタン一つを持っています。そこで、逆スケーリングの手法にはぴったりなのです。しかしながら、逆スケーリングはコンテイナー的な組み立て（パネル レイアウト等）には適していません。こういった場合には、残りのサブエレメントと律則関係にあるような背景を一つ持ったリフロー手法をお薦めします。

コンポーネントは Flash IDE のステージでスケーリングすることも、width と height プロパティ、または setSize() メソッドを使ってダイナミックにスケーリングすることもできます。スケーリングされたコンポーネントの外観は、Flash IDE では正確に見えない場合もあります。これはコンポーネントを、LivePreview のない未コンパイルの MovieClip として渡してしまうことの限度です。スケーリングされたコンポーネントが、どのようにゲーム内で表示されるか正確な表現を得る唯一の方法は、、Scaleform Player でこのムービーをテストすることです。CLIK エクステンションは、ワークフローのこの部分を向上する Launcher パネルをバンドルしています。

4.7.1 Scale9Grid

ほとんどのコンポーネントは 2 つ目のスケーリング方法を使用します。Scaleform Player では Scale9Grid 内部の MovieClip アセットは、大部分 scale9grid に従っていますが、Flash は MovieClip が含まれていればそのグリッドを破棄します。つまり、scale9grid が Flash Player で動作しない場合でも、Scaleform では完璧に動作する場合があるということです。

1 つ注意すべき点は、MovieClip に scale9grid が含まれている場合、サブエレメントもそのルールでスケーリングされるということです。Scale9grid をそのサブエレメントにも追加すれば、サブエレメントは親のグリッドを無視して、通常どおり描画するようになります。

4.7.2 Constraints

Constraints ユーティリティ クラス (gfx.utils.Constraints) は、スケーリングされるコンポーネント内のアセットのスケーリングと配置を支援します。このクラスを使うと開発者はステージ上にアセットを配置することはもちろん、アセットが親コンポーネントの端からの距離を維持するように保つことができます。たとえば、ScrollBar コンポーネントはトラックと下向き矢印ボタンを、ステージ上に置かれた位置に応じてサイズ変更し配置します。Constraints はコンポーネントで使用される両方のスケーリング メソッドで動作します。

以下のコードは ScrollBar アセットを configUI() メソッドの constraints オブジェクトに追加して、下向き矢印ボタンを最下部に揃え、トラックをスケーリングして親コンポーネントと共に拡大します。draw() メソッドにはこの constraints を更新するコードと、その結果登録されるエレメントが含まれています。この更新は draw() 内部で行われます。このメソッドはコンポーネントの無効化後、通常コンポーネントのサイズが変わった後で呼び出されるからです。

```
    private function configUI():Void {
        …
        constraints = new Constraints(this);
        // upArrow はすでに左上に吸着している。
        constraints.addElements(downArrow, Constraints.BOTTOM);
        constraints.addElenent(track, Constraints.TOP|Constraints.BOTTOM);
        …
    }

    private function draw():Void {
        …
       constraints.update(__width, __height);
        …
    }
```

4.8 コンポーネントとデータのセット

データのリストが必要なコンポーネントは dataProvider による方法を使用します。dataProvider はデータの保管と取得のオブジェクトで、Scaleform CLIK IDataProvider クラス (gfx.interfaces.IDataProvider) で定義された API のすべて、または一部を公開します。

dataProvider による方法は直接プロパティにアクセスするのではなく、コールバックを伴なうリクエスト モデルを使用します。これにより dataProvider は必要に応じて、データのためにゲーム エンジンに接触することができます。これはメモリ上の利点と、大きなデータ セットを細かく、管理しやすいものに分ける機能を提供します。

dataProvider を使用するコンポーネントは、CoreList (ScrollingList、TileList)、OptionStepper、DropdownMenu を拡張するものが含まれています。CLIK フレームワークのどのコンポーネントも IDataProvider インターフェイスを使用する必要はありません。その API のメソッドを含めるだけです。IDataProvider クラスは参照用にに提供されているだけです。

4.8.1 使用法とベスト プラクティス

フレームワークに含まれる DataProvider クラス (gfx.data.DataProvider) は静的 initialize() メソッドを使って、dataProvider メソッドを任意の ActionScript 配列に追加します。dataProvider を使用するコンポーネントは自動的に配列を初期化して、dataProvider による方法を使って自身にアクセスできるようにします。つまり、以下のシンタックスは静的に宣言された配列を、IDataProvider に記述されているメソッドで、完全に機能する dataProvider として初期化するという意味です:

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

dataProvider が実装するべきメソッドは以下のとおりです:

- *length :* プロパティ、または getter 関数のいずれかとして、データ セットの長さを返します。
- *requestItemAt :* dataProvider の特定の項目をリクエストします。OptionStepper など、常に 1 つの項目を表示するリスト コンポーネントが、通常は使用します。
- *requestItemRange :* 開始と終了インデックスで dataProvider の項目の範囲をリクエストします。ScrollingList など、複数の項目を表示するリスト コンポーネントが、通常は使用します。
- *indexOf :* 項目のインデックスを返します。
- *invalidate :* 変更があると、dataProvider にフラッグを付け、データの新規の長さを提供します。また、このメソッドは "dataChange" イベントをディスパッチして、データが更新されたことをコンポーネントに通知しなければなりません。dataProvider

は、公的にアクセス可能で、データ サイズを反映する length プロパティをサポートしなければなりません。

データの短いリストを必要とするインスタンスは、dataProvider として配列を使用することができます。開発者は ActionScript 2 のデータの保管が、アプリケーションにネイティブで保管するよりも、メモリとパフォーマンスの点では非常に負荷が高いということに気付くべきです。大型のデータ セットの場合、dataProvider を、ExternalInterface.call クラスにつなぎ、必要に応じてゲーム エンジンからデータをポーリングすることをお勧めします。

4.9 ダイナミック アニメーション

Scaleform CLIK には、Flash 8 Tween クラスに似た動作を備えていますが、完全に Scaleform と互換性があるカスタムの Tween クラス (gfx.motion.Tween) があります。どちらの Tween クラスも mx.transitions.easing.*パッケージのものなど、同じタイプのイージング関数をサポートします。

しかし、CLIK Tween クラスには Flash のものとは大きな相違点があります。まず、このクラスはトゥイーン メソッドを Flash 8 MovieClip クラスのプロトタイプにインストールします。これによりトゥイーンの機能は、CLIK コンポーネントだけでなくすべての MovieClip に公開されます。次に、CLIK Tween クラスは onEnterFrame を使用するので、MovieClip 毎に一度に 1 つのトゥイーンしかアクティブすることはできません。新規のトゥイーンをすでにアクティブなトゥイーンを備えた MovieClip に対して作成した場合、この新規のトゥイーンは前のトゥイーンを停止します。ただし、これらのトゥイーン メソッドは MovieClip 毎に複数のプロパティをサポートして、同じオブジェクトのさまざまなプロパティを同時に変更できるようにします。次の章の例は、同時に複数のプロパティに影響するトゥイーンの作成方法を示しています。

4.9.1 使用法とベスト プラクティス

Tween クラスは MovieClip の 2 つの主要メソッド tweenTo() と tweenFrom() を公開します。tweenTo() メソッドは MovieClip の現在のプロパティ値を、関数パラメータ リストで指定されたものにトゥイーンします。tweenFrom() メソッドはそのパラメータ リストで指定されたプロパティ値から、現在の値にトゥイーンします。

使用する前に、これらのトゥイーン メソッドを MovieClip プロトタイプにインストールする必要があります。この Tween クラスはヘルパー静的関数 Tween.init() を提供してこの動作を行います:

```
import mx.transitions.easing.*;
import gfx.motion.Tween;
Tween.init();     // トゥイーン メソッドを MovieClip プロトタイプにインストールする
// 現在の水平方向の位置とアルファ値から、指定されたものまでの 1 秒間のトゥイーンを行う
myMovieClip.tweenTo(1, {_x: 200, _alpha: 0}, Strong.easeIn);
```

このトゥイーンが完了したときに通知するには、単純にトゥイーンされているオブジェクトに onTweenComplete() 関数を作成します。

```
import mx.transitions.easing.*;
import gfx.motion.Tween;
Tween.init();
mc.onTweenComplete = function() {
      trace("Tween has finished!");
}
mc.tweenTo(5, {_rotation: 20},   Bounce.easeOut);
```

*4.10*ポップアップのサポート

Scaleform CLIK には PopUpManager クラス (gfx.managers.PopUpManager) が含まれていて、ダイアログやツールチップなどのポップアップをサポートします。Dialog コンポーネントはこの PopUpManager を使って自らのコンテンツを表示します。

4.10.1　使用法とベスト プラクティス

PopUpManager は複数の静的メソッドを備えており、ポップアップの作成と管理を支援します。createPopUp() メソッドは任意の MovieClip シンボル (CLIK コンポーネントを含みます) へのリンケージ ID を使って、ポップアップを作成する場合に使用できます。context パラメータを使ってポップアップのインスタンスを作成し、relativeTo パラメータを使ってポップアップを配置します。どちらのパラメータも MovieClip になります。

```
import gfx.managers.PopUpManager;
PopUpManager.createPopUp(context, "MyToolTipLinkageID",
                                  {_x: 200, _y:200}, relativeTo);
```

Scaleform エクステンション topmostLevel を使って、ポップアップが常にステージ上で他のすべてのエレメントの上に表示されることが保証されるようにします。このスキームは、ライブラリ スコープに関する問題のため、ルート レベルでポップアップを作成する代わりに採用されました。子 SWF が親の内部にロードされていて、親のコンテキストに存在するパスのライブラリで定義されたシンボルのインスタンスを作成しようとした場合、親は子のライブラリにアクセスできないので、シンボル ルックアップ エラーが発生します。残念ですが、この問題をうまく回避する方法はなく、topmostLevel スキームが最善の方法です。

topmostLevel はポップアップ以外のものにも適用できます。ステージのこのようなエレメントの Z オーダーが、維持されます。したがって、ポップアップの上にカスタムのカーソルを描画するために、そのカーソルを最も高い深度、またはレベルで作成して、その topmostLevel プロパティを true に設定することができます。

*4.11*ドラッグ&ドロップ

DragManager クラス (gfx.managers.DragManager) は、ドラッグ操作の開始と管理をサポートします。このクラスはシングルトンと同様に動作し、唯一のオブジェクトにアクセスするインスタンス プロパティを提供します。このオブジェクトを使用すると、開発者はドラッグ操作の開始/停止を行うことができます。

startDrag() メソッドを呼び出して、ステージ上の MovieClip、またはシンボル ID いずれかを使ってドラッグ操作を開始することができます。これはドラッグするためのシンボルのインスタンスを作成します。stopDrag() メソッドはドラッグ操作を終了し、さらにすべてのリスナーに通知します。また、DragManager は自らを InputDelegate に登録して、キーボードや同等のコントローラを使ったドラッグ操作のキャンセルも許可します。

4.11.1　使用法とベスト プラクティス

図 64: 動作中のドラッグ デモ

DragManager はドラッグの管理を行うだけです。この DragManager を使って、本当の意味でドラッグ&ドロップをサポートするコンポーネントを作成するかどうかは開発者次第です。Scaleform CLIK は DragTarget (gfx.controls.DragTarget) というサンプルのドロップ ターゲット クラスを含んでいます。この DragTarget は UIComponent を拡張するので、CLIK コンポーネントとして分類されます。このクラスは DragManager を補完する固有の機能を含んでいます。

DragTarget にはドラッグ タイプのコンセプトが含まれています。このようなドラッグ タイプは、、コンポーネントがドロップ操作を許可/拒否できるように DragTarget ごとに構成することが可能です。このようなドラッグ タイプを補完するためには、DragTarget は dragBegin と dragEnd のイベント リスナーも DragManager にインストールします。これにより DragTarget は、ドロップ操作を受け入れるか否かに関わらず、表示することができます。

CLIK にバンドルされている Inventory Demo は、DragTarget をサブクラス化または作成する 2 つのコンポーネントを使って、DragManager と DragTarget を使ったドラッグ&ドロップ操作を実演しています。この 2 つのコンポーネントとは InventorySlot と ItemSlot です。この 2 つのコンポーネント クラスは CLIK フレームワークの一部として提供されており、何ができるかを示すサンプルとして用意されていますが、すべての使用例に対する究極のソリューションではありません。

InventorySlot クラス (gfx.controls.InventorySlot) は、ロール プレイング ゲームでよく目にするような、インベントリー項目を表す一連のアイコンの表示をサポートしています。このクラスは DragTarget クラスをサブクラス化してドロップをサポートし、DragManager を使ったドラッグ操作を作成するコードを含んでいます。ドラッグ操作が始まると、アイコンのコピーが作成されてカーソルに合わせて移動します。有効なドロップ操作では、ターゲットの InventorySlot はそのアイコンをドラッグされたものに変更します。

ItemSlot クラス (gfx.controls.ItemSlot) は InventorySlot に非常に似ていますが、クリックなどのマウス イベントをサポートする追加機能を含んでいます。サブクラス化する代わりに、ItemSlot は DragTarget のインスタンスを含んでいます。また、CLIK Button のインスタンスも含んでいます。これらの 2 つのサブエレメントは ItemSlot に必要な機能を提供して、ドラッグ&ドロップとボタンの操作の両方をサポートします。

*4.12*その他

4.12.1 Delegate

Delegate クラス (gfx.utils.Delegate) は、正しいスコープを持った関数デリゲートを作成する静的メソッドを提供します。これは Scaleform CLIK コンポーネントでは使用されません。主に DragManager が使用します。使用例:

```
eventProvider.onMouseMove = Delegate.create(this, doDrag);
```

4.12.2 Locale

Locale クラス (gfx.utils.Locale) はカスタムのローカライズ スキームにインターフェイスを提供します。Scaleform のローカライズ スキームは内部でトランスレーションのルックアップを行うので、ActionScript からの明確なルックアップは必要ありません。ただし、カスタ

ムのトランスレーション プロバイダがこのルックアップを必要とする場合もあります。Button、Label、TextInput コンポーネントとそのサブクラスはすべて、Locale.getTranslatedString() 静的メソッドを使ってローカライズされた文字列をクエリーします。デフォルトのインプリメンテーションは単純に同じ値を返します。この Locale クラスをカスタムのローカライズ スキームをサポートするように変更するかどうかは、開発者次第です。

5 使用例

これ以降の章には、Scaleform CLIK コンポーネントを使って実装された使用例のサンプルが含まれています。

5.1 基礎編

以下のサンプルは範囲と複雑さにおいては単純ですが、一般的なタスクを実行するための Scaleform CLIK フレームワークの使い勝手の良さを表しています。

5.1.1 2 つの textField を備えた ListItem Renderer

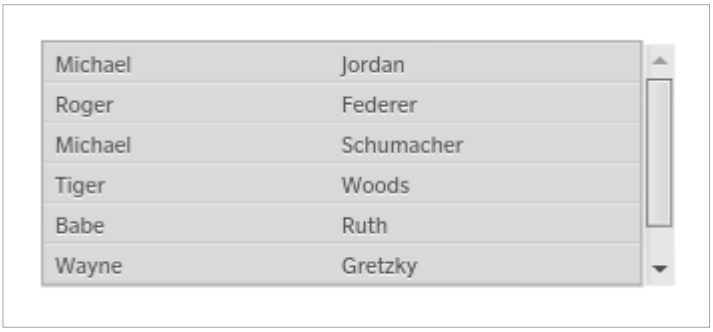

図 65: 2 つのラベルを含むリスト項目を表示した ScrollingList

この ScrollingList コンポーネントは、デフォルトで ListItemRenderer を使って行内容を表示します。ただし、ListItemRenderer は 1 つの textField しかサポートしません。リスト項目が複数の textField を表示するケースはいくつもあります。さらにアイコンなどの textField 以外のリソースを表示することもあります。このサンプルは 2 つの textField をリスト項目に追加する方法を表しています。

まず、必要条件を定義します。目的は 2 つの textField をサポートするカスタムの ListItemRenderer を作成することです。また、このカスタムの ListItemRenderer は ScrollingList と互換性があるべきです。計画ではリスト項目に付き 2 つの textField を持つ予定なので、データも単一文字列のリストだけではないはずです。このサンプルに以下の dataProvider を使用してみます:

```
    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"}];
```

この dataProvider には 2 つのプロパティ fname と lname を持つオブジェクトが含まれています。これらの 2 つのプロパティは 2 つのリスト項目 textField に表示されます。

デフォルトの ListItemRenderer は 1 つの textField しかサポートしないので、2 つの textField をサポートする機能が必要となります。これを実現する最も簡単な方法は、ListItemRenderer クラスをサブクラス化することです。以下は MyItemRenderer というクラスに対するソース コードです。これは ListItemRenderer をサブクラス化して、2 つの textField の基本的なサポートに追加します。(このコードを作業中の FLA と同じディレクトリの *MyItemRenderer.as* というファイルに追加します:

```
import gfx.controls.ListItemRenderer;

class MyItemRenderer extends ListItemRenderer {

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

	public function MyItemRenderer() { super(); }

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

	private function updateAfterStateChange():Void {
		textField1.text = data ? data.fname : "";
		textField2.text = data ? data.lname : "";
	}
}
```

ListItemRenderer の setData と updateAfterStateChange メソッドは、このサブクラスでオーバーライドされます。setData メソッドは、リスト項目がそのコンテナ リスト コンポーネント (ScrollingList、TileList など) から項目のデータを受け取るときは常に呼び出されます。ListItemRenderer では、このメソッドは 1 つの textField の値を設定します。MyItemRenderer は代わりに両方の textField の値を設定して、内部でその項目オフジェクトへのリファレンスも保管します。この項目オブジェクトは updateAfterStateChange で再利用されます。このメソッドは ListItemRenderer のステートが変更になると常に呼び出されます。このステートの変更によって、textField は自らの値の更新が必要になる場合があります。

ここまでは、複雑なリスト項目を持つ dataProvider と、そのリスト項目のレンダリングをサポートする ListItemRenderer を定義してきました。すべてをこのリスト コンポーネントと組み合わせるために、シンボルを作成してこの新規の ListItemRenderer クラスをサポートしなければなりません。このサンプルの場合、これを実現する最速の方法は、ListItemRenderer シンボルを 'textField1' と 'textField2' の 2 つの textField を持つように

変更することです。次に、このシンボルの識別子とクラスを MyItemRenderer に変更する必要があります。このリストで MyItemRenderer コンポーネントを使用するために、リスト インスタンスの itemRenderer 検証可能プロパティを 'ListItemRenderer' から 'MyItemRenderer' に変更します。

この FLA を実行します。リストは、dataProvider に設定された fname と lname プロパティという 2 つのラベルを表示するリスト エレメントを含んで表示されるはずです。

5.1.2 ピクセル単位のスクロール表示

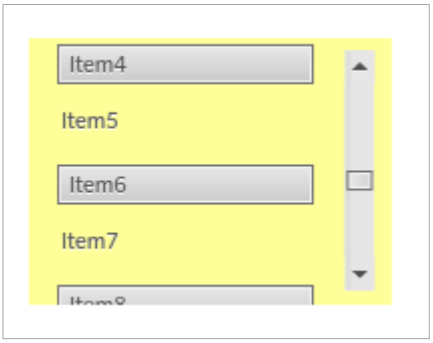

図 66: CLIK エレメントを含むコンテンツを持ったピクセル単位のスクロール表示

ピクセルごとのスクロールは、複雑なユーザー インターフェイスでは一般的な使用事例です。このサンプルは、CLIK ScrollBar コンポーネントを使ってこれを簡単に実現する方法を示しています。

まず始めにスクロール表示の新規のシンボルを作成します。これは、コンテンツのオフセットの算出に必要とされる計算を簡素化する便利なコンテナを提供します。このスクロール表示コンテナ内で、以下のレイヤーを上から下への順序でセットアップします。これらのレイヤーは必須ではありませんが、分りやすくするためにお薦めします:

- *actions :* スクロール表示を動作させる ActionScript コードが含まれています。
- *scrollbar :* CLIK ScrollBar のインスタンスが含まれています。このインスタンスを 'sb' とします。
- *mask :* 四角形、または MovieClip で定義されたマスク レイヤーです。レイヤー プロパティもマスクに設定します。
- *content :* スクロールするコンテンツが含まれます。
- *background :* マスクされていない領域を強調表示するバックグラウンドを含んだオプションのレイヤーです。

コンテンツ レイヤーに適切なエレメントを追加して、そのエレメントをすべて保有するシンボルを作成します。コンテンツをすべて保有するシンボルを作成することで、コンテンツをスクロールするというタスクがさらに容易になります。このコンテンツ インスタンスを 'content' として、その y 値を 0 に設定します。これはこのスクロール ロジックがさまざま

なオフセットを明らかにする必要がないということを保証します。ただし、このようなオフセットが、作成するスクロール表示の複雑さに応じて、必要になる場合もあります。

ここまでは、簡単なスクロール表示の作成に必要な構造と、合わせて組み込む必要のある指定エレメントを定義してきました。*code* レイヤーの最初のフレームに以下の ActionScript コードを加えます:

```
// 133 は表示サイズ （マスクの高さ） である
sb.setScrollProperties(1, 0, content._height - 133);
sb.position = 0;

sb.addEventListener("scroll", this, "onScroll");
function onScroll() {
      content._y = -sb.position;
}
```

このスクロールバーは別のコンポーネントには接続されていないので、手動で構成する必要があります。setScrollProperties メソッドを使って、このスクロールバーをある位置単位でスクロールし、完全にコンテンツを表示するための最小値と最大値を設定します。この場合のスクロールバーの位置はピクセル単位です。このファイルを実行します。スクロール表示は、スクロールバーで操作することで変化するようになっているでしょう。

Scaleform で大量にマスクを使用すると、特に HUD の場合、著しくパフォーマンスが低下することがあるので注意してください。Scaleform は開発者が、コンテンツをプロファイリングして、パフォーマンス メトリクスをとることをお勧めします。

5.2 上級編

これ以降の章では、Scaleform CLIK にバンドルされている複合デモについて説明します。高度なインプリメンテーションの詳細のみを提供しています。これ以降の章では、ほとんど完璧な Flash と ActionScript 2 の知識が必要なので注意してください。

ScrollingList を使った TreeView

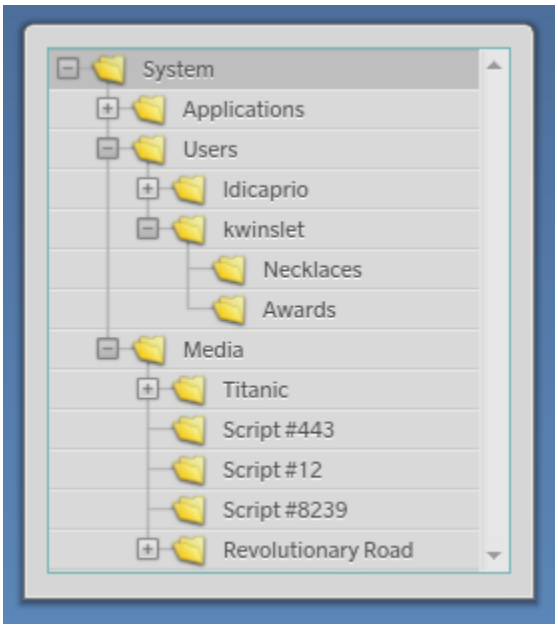

図 67: TreeView のデモ

TreeView Demo (*Resources/AS2/CLIK/demos/TreeViewDemo.fla*) はカスタムの ListItemRenderer と DataProvider の他に、ベースとして ScrollingList コンポーネントを使ったツリー表示のコントロールを実装しています。

標準の ScrollingList とツリー表示の違いは以下のとおりです:

- ツリー表示のエレメント数はその項目の階層状況に応じて変更することができますが、それでも直線的なエレメントのシーケンスです。
- ツリー表示の項目は視覚的なインジケータを使って、その階層ツリー内の自分の深度を表示しなければなりません。ほとんどの場合、この深度は「タブが設定された」ように見えるスペースで表現されます。これは ListItemRenderer を、その内容に「タブが設定された」ように変更することで簡単に行うことができます。
- ツリー表示は、ユーザーが各項目の展開や縮小を行えるメカニズムを含んでいなければなりません。再度、ListItemRenderer をこの機能を提供するボタン サブエレメントを含むように変更することができます。

ScrollingList は一から作成されたツリー表示の代理を効率的に務めます。このデモの場合、TreeViewItemRenderer というカスタムの ListItemRenderer と TreeViewDataProvider というカスタムの DataProvider が定義されています。また、このデモは TreeViewConstants という小さなユーティリティ クラスを使って、共通の一覧を提供しています。このようなクラスはすべて *Resources/AS2/CLIK/demos/com/scaleform* ディレクトリにあります。

ツリー表示を実装するときに最初に決めることは、そのデータの表現です。ActionScript 2 Object は本質的にハッシュ テーブルで、このデモはそのプロパティを使用しています。ツリーは以下のような Object のツリーを使って構成されています:

```
var root:Object =
 {
       label: "System",
       nodes:[
                {label: "Applications",
                 nodes:[
                            {label: "CGStudio",
                             nodes:[
                                        {label: "Data"},
                                        {label: "Samples"},
                                        {label: "Config"}
                                      ]
                              },
                            {label: "Money Manager"},
                            {label: "Role Call v6"},
                            {label: "Super Draft",
                             nodes:[
                                        {label: "Templates"},
  …
```

ツリーの各項目は label と nodes という 2 つのプロパティを持つ Object として表現されています。これは子ノードを含む可能性がある Array です。ScrollingList とこの階層表現をつなぐために、ツリー構造を解析して維持し、該当するデータをリストに公開するカスタムの DataProvider が必要となります。

TreeViewDataProvider はそのコンストラクタでこのツリーオブジェクトを取り入れ、各項目をその DataProvider とカスタムの ListItemRenderer だけに有効な特殊なプロパティで、各項目に注釈を付ける前処理を実行します。これらのプロパティには項目のタイプ、その展開/縮小ステート、親/兄弟リンク、レンダリン中に使用する線図形処理を記述する線図形記述子の配列が含まれます。TreeViewDataProvider は CLIK のデータ プロバイダが必要とする適切なパブリック メソッドを実装していますが、項目取得作業のほとんどを複数のヘルパー関数に委ねています。たとえば、項目の範囲の取得は、まずリストの最上部の項目インデックスを使ってツリー オブジェクトをさがし、次にその最上部の項目から始まる直線的なシーケンスを、ツリー走査を使って収集します。

TreeViewItemRenderer はそのデータを ScrollingList 経由で TreeViewDataProvider から受け取ります。取得したデータ オブジェクトには、ツリーの構造の知識がまったくなくても、その項目を正しく表示するために必要なすべてのメタ データが含まれています。未変更の ListItemRenderer コンポーネント シンボルが、TreeViewItemRenderer へのクラス リンケージ セットと共にこのデモで再利用されています。このリスト項目のフォルダ アイコンとライン コネクタのグラフィックは、ダイナミックに作成されています。このようなグラフィック エレメントはあらかじめキャッシュされ、同じ「タブ」スロットで同じシンボルのインスタンス化の重複を避けます。ListItemRenderer の textField エレメントは、項目の深度に基づいて、右にシフトされます。

TreeViewConstants クラスは以下の一覧を含んでいます:

- 階層中のノードのタイプ：開いているノード、閉じているノード、リーフ ノード
- フォルダ アイコンのタイプ
- ライン コネクタのタイプ

下の 2 つの一覧は表示目的のみに使用されます。このデモはフォルダ アイコンと線図形を使って、深度に関係なく項目間をシームレスに接続しています。この効果を実現するには、フォルダ アイコンと線図形は展開表示や縮小表示のさまざまな構成をすべて対象としなければなりません。

このツリー表示は ScrollingList をベースにしているので、ScrollBar コンポーネントを接続することができます。スクロールバーはツリー階層のステートに応じてその外観を変えます。ツリーの展開/縮小ステートはスクロールバーのツマミのサイズに影響します。スクロールバーを操作すると ScrollingList 表示が予想通りに変更されます。

5.2.1 再利用可能なウィンドウ

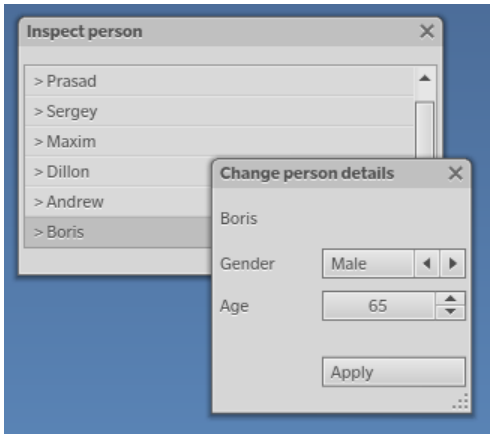

図 68: Window のデモ

Window Demo (*Resources/AS2/CLIK/demos/WindowDemo.fla*) は、任意の内容を表示できる簡単で再利用可能なウィンドウ コンポーネントを提供しています。Window コンポーネント クラスは *Resources/AS2/CLIK/demos/com/scaleform* にあります。

この Window コンポーネントは、コア UIComponent クラスをサブクラス化するので、CLIK コンポーネントに分類されます。このコンポーネントは、コア フレームワーク クラスには存在しません。汎用の Window コンポーネントは定義が難しく、まして効率的に実装するのは困難だからです。コンポーネントに機能を追加するほど、メモリとパフォーマンスの点では、使い難くなります。

したがって、このクラスは以下のような、Window コンポーネントに期待される一般的な機能のコア セットを実装するだけです:

- 変更可能なタイトル バー
- ドラッグ可能なタイトル バーとウィンドウの背景
- 閉じるボタン
- サイズ変更可能な境界線 (右下のみ)
- 最小サイズと最大サイズ

Window クラスは複数の検証可能プロパティを公開しています:

Title	その Window のタイトルです。
formPadding	Window の境界線とそのコンテンツ間の余白の量です。
formType	ロードするコンテンツのタイプです。この値は 'symbol' か 'swf' のいずれかです。'symbol' の場合、コンテンツはライブラリからロードされます。'swf' の場合、外部の SWF ファイルがロードされます。
formSource	コンテンツのソースです。これはシンボル名 (*formType* が 'symbol' の場合)、または SWF ファイル名 (formType が 'swf' の場合) のいずれかです。
allowResize	サイズ変更ボタンを表示/非表示にします。
minWidth, maxWidth, minHeight, maxHeight	ウィンドウのサイズ変更の寸法です。max 値が負数に設定された場合、対応する min 設定に固定されます。両方の max 値を負数に設定することは、サイズ変更ボタンを非表示にすることと同じ効果になります。これは完全にサイズ変更を不可にします。max 値が 0 に設定された場合、コンポーネントは上限なしにサイズ変更することができます。min 値は常にコンテンツの初期サイズです。

検証可能プロパティをカスタムのコンポーネント クラスに追加するのは簡単です。パブリック プロパティがプロパティの取得、または設定時に実行するコードを必要としない場合、以下のように宣言することができます:

```
    [Inspectable(name="formType", enumeration="symbol,swf"]
    private var _formType:String = "symbol";
```

Inspectable メタデータ シンタックスの詳細については、Flash の技術資料を参照してください。Inspectable メタデータは実際のプロパティの別名をサポートします。この場合 _formType プロパティは、読みやすくするための formType の別名です。

プロパティが値の設定、または取得後に実行するコードを必要とする場合、プロパティは getter 関数と setter 関数を定義する必要があります。

```
[Inspectable(defaultValue="Title")]
public function get title():String { return _title; }
public function set title(value:String):Void {
      _title = value;
      invalidate();
}
```

この Window コンポーネントは本質的に、タイトル ボタン、閉じるボタン、サイズ変更ボタンなどの複数のサブエレメントをサポートしています。これらのボタンはそれぞれ CLIK Button コンポーネントです。このようなボタンはこのクラスに必要とされるので、コンポーネント シンボルは該当するサブエレメントを含めることで、この要求を反映しなければなりません。

Window クラスは UIComponent クラスをサブクラス化するので、configUI() と draw() の 2 つのメイン メソッドはオーバーライドされます。configUI() メソッドは、ボタン リスナーとサイズ変更のための constraints オブジェクトをセットアップします。draw() メソッドは一般的にコンテンツのレイアウトのリフローを手動で、または constraints オブジェクトを使用するかのいずれかで行います。ただし、シンボル名、またはファイル パスを使ってコンテンツをロードするロジックも含んでいます。プライベート メソッドのロード後に、configForm() が呼び出され、コンテンツのセットアップを行います。

ファイルのロードは Flash 8 MovieClipLoader で行います。ファイルのロードはスレッドのサポートが有効なときに、Scaleform で非同期に行われるので、コンポーネントは、ファイルのロードが完了したときにトリガされる MovieClipLoader.onLoadComplete() コールバック内から configForm() を呼び出します。

シンボル、またはファイル パスのいずれかを使ってロードされるフォーム コンテンツが、内部の constraints オブジェクトに追加されます。Window コンポーネントのサイズが変わるときは常に、そのオブジェクトは無効化されます。これにより draw() メソッドが呼び出されます。このメソッドは結果的にその constraints オブジェクトの更新を新規のサイズで行います。フォーム コンテンツは constraints オブジェクトに登録されているので、validateNow() メソッドで変更を通知されます。

フォーム コンテンツが UIComponent から派生している場合、独自の draw() メソッドが呼び出されます。そのフォーム コンテンツのレイアウト管理ロジックを、draw() メソッドが呼び出されたときに実行することができます。フォーム コンテンツが UIComponent に由来していない場合、validateNow() メソッドを定義して、独自のエレメントをリフローすることができます。Window Demo には、draw() メソッドと validateNow() メソッドを使ってフ

ォームのレイアウトを維持する、シンボルと SWF ファイルの複数のサンプルが含まれています。

また、Window コンポーネントには、マウス カーソルで押したウィンドウを前面に移動する特殊なロジックも含まれています。このロジックは単純に、すべての Window コンポーネント インスタンスは同じレベルに存在すると仮定して、フォーカスが適用されているコンポーネントを 2 番目に高い深度にバブルします。

```
    private function onMouseDown() {
          var targetObj:Object = Mouse.getTopMostEntity();
          while (targetObj != null && targetObj != _root)
          {
                if (targetObj == this) {
                      swapDepths(_parent.getNextHighestDepth());
                      return;
                }
          targetObj = targetObj._parent;
          }
    }
```

これはこのような動作を実装する 1 つの方法です。別の方法として、WindowManager を作成して、ステージ上のすべてのウィンドウ コンポーネントのインデックスを維持することがあげられます。この WindowManager による方法は実際には、このサンプル Window コンポーネントで実装した方法よりも、さらに自由度が高くなります。