

ACTIONSCRIPT™ 2.0

コンポーネントガイド

ActionScript 2.0 コンポーネントガイド

本マニュアルがエンドユーザー使用許諾契約を含むソフトウェアと共に提供される場合、本マニュアルおよびその中に記載されているソフトウェアは、エンドユーザー使用許諾契約にもとづいて提供されるものであり、当該エンドユーザー使用許諾契約の契約条件に従ってのみ使用または複製することが可能となるものです。当該エンドユーザー使用許諾契約により許可されている場合を除き、本マニュアルのいかなる部分といえども、Adobe Systems Incorporated (アドビ システムズ社) の書面による事前の許可なしに、電子的、機械的、録音、その他いかなる形式・手段であれ、複製、検索システムへの保存、または伝送を行うことはできません。本マニュアルの内容は、エンドユーザー使用許諾契約を含むソフトウェアと共に提供されていない場合であっても、著作権法により保護されていることにご留意ください。

本マニュアルに記載される内容は、あくまでも参照用としてのみ使用されること、また、なんら予告なしに変更されることを条件として、提供されるものであり、従って、当該情報が、アドビ システムズ社による確約として解釈されることはありません。アドビ システムズ社は、本マニュアルにおけるいかなる誤りまたは不正確な記述に対しても、いかなる義務や責任を負うものではありません。

新しいアートワークを創作するためにテンプレートとして取り込もうとする既存のアートワークまたは画像は、著作権法により保護されている可能性のあるものであることにご留意ください。保護されているアートワークまたは画像を新しいアートワークに許可なく取り込んだ場合、著作権者の権利を侵害することがあります。従って、著作権者から必要なすべての許可を必ず取得してください。

例として使用されている会社名は、実在の会社・組織を示すものではありません。

Adobe®、Flash®、Flash® Player、Flash® Video、および Macromedia® は、アドビ システムズ社の米国ならびに他の国における商標または登録商標です。

Macintosh® は、Apple Computer, Inc. の米国および他の国における登録商標です。Windows® は、米国およびその他の国における Microsoft Corporation の登録商標または商標です。All other trademarks are the property of their respective owners.

Portions of this product contain code licensed from Nellymoser.(www.nellymoser.com).



Sorenson™ Spark™ ビデオ圧縮および圧縮解除テクノロジーは、Sorenson Media, Inc. のライセンス供与によって提供されます。

Flash CS3 video is powered by On2 TrueMotion video technology. © 1992-2005 On2 Technologies, Inc. All Rights Reserved. <http://www.on2.com>.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA. Notice to U.S. Government End Users. The Software and Documentation are "Commercial Items," as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA. For U.S. Government End Users, Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

目次

はじめに.....	7
対象となる読者.....	8
システム要件.....	8
本マニュアルについて.....	8
表記規則.....	9
用語.....	9
他の情報源.....	9
第1章：コンポーネントについて.....	11
コンポーネントのインストール.....	12
コンポーネントファイルの格納場所.....	14
コンポーネントファイルの編集.....	14
コンポーネントを使用するメリット.....	15
コンポーネントのカテゴリ.....	16
バージョン2のコンポーネントアーキテクチャについて.....	17
バージョン2コンポーネントの機能.....	18
コンパイルされたクリップと SWC ファイルについて.....	19
アクセシビリティとコンポーネント.....	20
第2章：コンポーネントを使用したアプリケーションの作成.....	21
お詫びギフトショップのチュートリアルについて.....	21
メインページの作成.....	23
お勧めギフトの表示用データコンポーネントのバインディング.....	29
ギフトの詳細の表示.....	33
チェックアウトスクリーンの作成.....	38
アプリケーションのテスト.....	47
完成済みアプリケーションの参照.....	47
第3章：コンポーネントの利用.....	49
[コンポーネント] パネル.....	50
Flash ドキュメントへのコンポーネントの追加.....	50
[ライブラリ] パネル内のコンポーネント.....	54
コンポーネントパラメータの設定.....	54

コンポーネントのサイズ変更	56
Flash ドキュメントからのコンポーネントの削除	57
コードヒントの使用	58
スクリーンでの ActionScript の使用	58
スクリーンと ActionScript のやり取りの例	59
カスタムフォーカスナビゲーションの作成	61
ドキュメント内のコンポーネントの深度を管理する	62
ライブプレビュー内のコンポーネント	63
コンポーネントでのプリローダーの使用	64
コンポーネントのロードについて	65
バージョン 1 コンポーネントからバージョン 2 アーキテクチャへの アップグレード	66
 第 4 章：コンポーネントイベントの処理	 67
リスナーによるイベントの処理	68
イベントの委譲	76
イベントオブジェクトについて	79
on() コンポーネントイベントハンドラの使用	80
 第 5 章：コンポーネントのカスタマイズ	 81
スタイルによるコンポーネントのカラーとテキストの変更	82
コンポーネントへのスキンの適用について	96
テーマについて	108
スキンとスタイルの組み合わせによるコンポーネントのカスタマイズ	119
 第 6 章：コンポーネントの作成	 125
コンポーネントのソースファイル	125
コンポーネント構造の概要	126
初めてのコンポーネントの作成	127
親クラスの選択	136
コンポーネントムービークリップの作成	138
ActionScript クラスファイルの作成	143
複数の既存コンポーネントを組み込んだコンポーネントの作成	172
コンポーネントの書き出しと配布	180
コンポーネント開発の最後の手順	184

第7章：コレクションプロパティ	187
コレクションプロパティの定義	188
簡単なコレクションの例	189
コレクションアイテムのクラス定義	191
プログラムによるコレクション情報へのアクセス	192
コレクションが定義されているコンポーネントの SWC ファイルへの 書き出し	194
コレクションプロパティが定義されているコンポーネントの使用	195
索引	197

はじめに

Flash CS3 Professional は、インパクトの強い Web 体験を生み出すための標準オーサリングツールです。コンポーネントとは、そのような体験を生むリッチなインターネットアプリケーションを構築するブロックのような部品です。コンポーネントは、Flash でのオーサリング中に設定されるパラメータと、実行時にコンポーネントをカスタマイズするための ActionScript メソッド、プロパティ、およびイベントを備えたムービークリップです。開発者はコンポーネントを通じてコードを再利用および共有し、複雑な機能をカプセル化することができます。デザイナーは ActionScript を使わずに、それらを利用してカスタマイズすることができます。

コンポーネントは、Adobe Component Architecture のバージョン 2 に基づいているため、開発者は一貫した外観と操作性を備えた堅牢なアプリケーションをすばやく簡単に作成できます。本マニュアルでは、バージョン 2 コンポーネントを使用してアプリケーションを作成する方法について説明します。関連する『ActionScript 2.0 コンポーネントリファレンスガイド』では、各コンポーネントのアプリケーションプログラミングインターフェイス (API) について説明します。その中で、Flash バージョン 2 コンポーネントの使用方法和手順の例を示すと共に、コンポーネントの API をアルファベット順に説明していきます。

Adobe が作成したコンポーネントを使用することも、他の開発者が作成したコンポーネントをダウンロードすることも、独自のコンポーネントを作成することもできます。

この章では、次のセクションについて説明します。

対象となる読者	8
システム要件	8
本マニュアルについて	8
表記規則	9
用語	9
他の情報源	9

対象となる読者

本マニュアルは、Flash アプリケーションを作成しており、コンポーネントを使って開発を効率化したいと考えている開発者向けに書かれています。このため、Flash でのアプリケーションの開発と ActionScript の記述について既に知識があることが望まれます。

ActionScript を作成した経験が少ない方でも、コンポーネントをドキュメントに追加し、プロパティインスペクタや [コンポーネントインスペクタ] パネルでパラメータを設定し、[ビヘイビア] パネルでイベントを操作することができます。たとえば、[Web ページへ移動] ビヘイビアを Button コンポーネントに追加すると、ActionScript コードをいっさい記述しなくても、ボタンがクリックされると Web ブラウザの中で URL を開くことができます。

作成するアプリケーションの堅牢性を高めるには、コンポーネントを動的に作成します。ActionScript を使用して、実行時にプロパティを設定し、メソッドを呼び出します。また、リスナーイベントモデルを使用して、イベントを処理します。

詳細については、[49 ページ](#)、[第 3 章の「コンポーネントの利用」](#)を参照してください。

システム要件

Adobe コンポーネントには、Flash 以外のシステム要件はありません。

バージョン 2 コンポーネントを使用する SWF ファイルは、Flash Player 6 (6.0.79.0) 以降で表示する必要があります。さらに、ActionScript 2.0 を使ってパブリッシュする必要もあります ([ファイル]-[パブリッシュ設定]-[Flash] タブで設定できます)。

本マニュアルについて

本マニュアルでは、コンポーネントを使った Flash アプリケーションの開発について詳しく説明します。読者は Flash および ActionScript の全般的な知識を持っていることが前提になります。Flash および関連製品に関する資料は、個別に入手可能です。

本マニュアルは、PDF ファイルとオンラインヘルプの形式で提供されています。オンラインヘルプを表示するには、Flash を起動し、[ヘルプ]-[ActionScript 2.0 コンポーネントガイド] を選択します。

Flash の詳細については、次のマニュアルを参照してください。

- Flash ユーザーガイド
- ActionScript 2.0 の学習
- ActionScript 2.0 リファレンスガイド
- ActionScript 2.0 コンポーネントリファレンスガイド

表記規則

本マニュアルでは、次の表記規則を使用しています。

- `Code font` (コードフォント) は、メソッドやプロパティ名など、ActionScript コードを示します。
- *Code font italic* (イタリック体のコードフォント) は、置き換える必要があるコードアイテム (ActionScript パラメータなど) を示します。
- **Bold font** (ボールドフォント) は、ユーザーが入力する値を示します。

用語

本マニュアルでは、次の用語を使用しています。

実行時 Flash Player でコードを実行しているときです。

オーサリング時 Flash オーサリング環境で作業をしているときです。

他の情報源

Flash の最新情報、アドバンスドユーザーからのアドバイス、高度なテクニック、サンプル、ヒント、その他の更新情報については、Adobe DevNet の Web サイト (www.adobe.com/jp/devnet) を参照してください。このサイトは定期的に更新されます。Flash の最新ニュースと Flash を最大限に活用する方法を提供するこの Web サイトを頻繁にアクセスしてみてください。

テクニカルノート、マニュアルの更新情報、および Flash コミュニティ内の追加情報へのリンクについては、Adobe Flash サポートセンター (www.adobe.com/jp/support/flash) を参照してください。

ActionScript の用語、シンタックス、および使用方法については、『ActionScript 2.0 の学習』と『ActionScript 2.0 リファレンスガイド』を参照してください。

コンポーネントの使用の概要については、Adobe On Demand Seminar (www.adobe.com/events/main.jsp) の「Using UI Components」を参照してください。

コンポーネントについて

Flash CS3 Professional コンポーネントとは、外観やビヘイビアを変更するためのパラメータを持つムービークリップです。コンポーネントの中には、ラジオボタンやチェックボックスのような単純なユーザーインターフェイスコントロールもあれば、スクロールペインのようにコンテンツを含んでいるものもあります。また、アプリケーション内でフォーカスを取得するオブジェクトを制御する FocusManager のように、見えないコンポーネントもあり得ます。

コンポーネントを利用すれば、ActionScript に精通していないユーザーでも、Flash の複雑なアプリケーションを構築できます。ボタン、コンボボックス、リストは、[コンポーネント] パネルからドラッグしてアプリケーションに追加できるので、これらのコンポーネントをカスタムで作成する必要はありません。コンポーネントの外観は、デザイン上の都合に合わせて簡単にカスタマイズすることができます。

コンポーネントは、Adobe Component Architecture のバージョン 2 に基づいているため、開発者は一貫した外観と操作性を備えた堅牢なアプリケーションをすばやく簡単に作成できます。このバージョン 2 アーキテクチャには、すべてのコンポーネントの基礎となるクラス、コンポーネントの外観をカスタマイズするのに必要なスタイルやスキン、ブロードキャスター / リスナーイベントモデル、深度とフォーカスの管理、アクセシビリティの実装などが含まれます。

**×
rt**

バージョン 2 コンポーネントをパブリッシュする場合、ActionScript 2.0 を使用してパブリッシュするようパブリッシュ設定を行う必要があります ([ファイル]-[パブリッシュ設定]-[Flash] タブ)。ActionScript 1.0 または ActionScript 3.0 を使用してパブリッシュした場合、バージョン 2 コンポーネントは正常に動作しません。

各コンポーネントには定義済みのパラメータがあり、Flash でのオーサリング時にこれを設定することができます。各コンポーネントには、ActionScript メソッド、プロパティ、イベントの一意なセット、つまり API (Application Programming Interface) が用意されており、パラメータや追加のオプションを実行時に設定できるようになっています。


Flash に付属しているコンポーネントの一覧については、12 ページの「コンポーネントのインストール」を参照してください。また、Flash コミュニティのメンバーが作成したコンポーネントを、Adobe Exchange (www.adobe.com/jp/exchange/) からダウンロードすることもできます。

この章では、次のセクションについて説明します。

コンポーネントのインストール.....	12
コンポーネントファイルの格納場所.....	14
コンポーネントファイルの編集.....	14
コンポーネントを使用するメリット.....	15
コンポーネントのカテゴリ	16
バージョン 2 のコンポーネントアーキテクチャについて	17
バージョン 2 コンポーネントの機能	18
コンパイルされたクリップと SWC ファイルについて	19
アクセシビリティとコンポーネント.....	20

コンポーネントのインストール

Flash を最初に起動するときに、Adobe コンポーネントは既にインストールされています。それらのコンポーネントは [コンポーネント] パネルで表示できます。

	バージョン 2 コンポーネントを表示するには、ActionScript 2.0 を使用してパブリッシュするようパブリッシュ設定を行う必要があります ([ファイル]-[パブリッシュ設定]-[Flash] タブ)。デフォルトのパブリッシュ設定ではバージョン 2 コンポーネントが表示されません。
---	---

Flash には、次のコンポーネントが含まれています。

- Accordion
- Alert
- Button
- CheckBox
- ComboBox
- DataGrid
- DataHolder
- DataSet
- DateChooser
- DateField
- FLVPlayback
- Label
- List
- Loader
- Media

- Menu
- MenuBar
- NumericStepper
- ProgressBar
- RadioButton
- RDBMSResolver
- ScrollPane
- TextArea
- TextInput
- Tree
- UIScrollBar
- WebServiceConnector
- Window
- XMLConnector
- XUpdateResolver

Flash には DataBinding クラスもあります。このクラスにアクセスするには、[ウィンドウ]-[サンプルライブラリ]-[クラス] を選択します。

Flash コンポーネントを表示するには：

1. Flash を起動します。
2. ActionScript 2.0 を使用してパブリッシュするようパブリッシュ設定を行います。
[ファイル]-[パブリッシュ設定]-[Flash] タブを選択し、[ActionScript] ドロップダウンメニューから [ActionScript 2.0] を選択します。
3. [ウィンドウ]-[コンポーネント] を選択して、[コンポーネント] パネルを開きます。
4. [User Interface] を選択してツリーを展開すると、インストール済みのコンポーネントが表示されます。

Adobe Exchange (www.adobe.com/jp/exchange) からコンポーネントをダウンロードすることもできます。Exchange からダウンロードしたコンポーネントをインストールするには、www.adobe.com/jp/exchange/em_download/ から Macromedia Extension Manager をダウンロードしてインストールします。

コンポーネントはすべて、Flash の [コンポーネント] パネルに表示できます。Windows または Macintosh コンピュータにコンポーネントをインストールするには、次の手順を実行します。

Windows ベースのコンピュータまたは Macintosh コンピュータにコンポーネントをインストールするには：

1. Flash を終了します。
2. コンポーネントが含まれる SWC ファイルまたは FLA ファイルをハードディスクの次のフォルダに配置します。
 - Windows : C:\Program Files\Adobe\Adobe Flash CS3\言語\Configuration\Components
 - Macintosh : Macintosh HD/アプリケーション/Adobe Flash CS3/Configuration/Components
3. Flash を起動します。
4. ActionScript 2.0 を使用してパブリッシュするようパブリッシュ設定を行います。
[ファイル]-[パブリッシュ設定]-[Flash] タブを選択し、[ActionScript] ドロップダウンメニューから [ActionScript 2.0] を選択します。
5. [コンポーネント] パネルが開いていない場合は、[ウィンドウ]-[コンポーネント] を選択し、コンポーネントを表示します。

コンポーネントファイルの格納場所

Flash コンポーネントは、アプリケーションレベルの設定フォルダ内にあります。



これらのフォルダの詳細については、『Flash ユーザーガイド』の「Flash と共にインストールされる設定フォルダ」を参照してください。

コンポーネントは、次の場所にインストールされます。

- Windows 2000 または Windows XP : C:\Program Files\Adobe\Adobe Flash CS3\言語\Configuration\Components
- Mac OS X : Macintosh HD/ アプリケーション /Adobe Flash CS3/Configuration/Components

コンポーネントファイルの編集

コンポーネントの ActionScript ソースファイルは、次の場所に存在します。

- Windows 2000 または Windows XP : C:\Program Files\Adobe\Adobe Flash CS3\言語\First Run\Classes\mx
 - Mac OS X : Macintosh HD/ アプリケーション /Adobe Flash CS3/First Run/Classes/mx
- Flash を最初に起動したときに、「First Run」ディレクトリのファイルが「Documents and Settings」パスにコピーされます。「Documents and Settings」パスは次の場所にあります。
- Windows 2000 または Windows XP : C:\Documents and Settings\ユーザー名\Local Settings\Application Data\Adobe\Adobe Flash CS3\言語\Configuration\Classes\mx

- Mac OS X : < ユーザー名 > / Library / Application Support / Adobe / Adobe Flash CS3 / < 言語 > / Configuration / Classes / mx

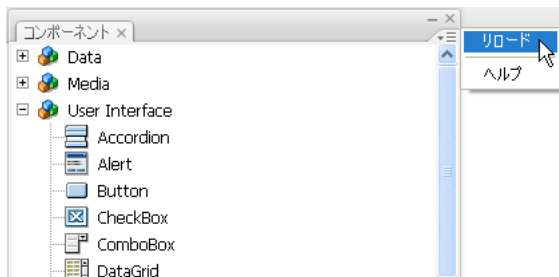
Flash の起動時に "Document and Settings" パスでファイルが見つからない場合、Flash は、"First Run" ディレクトリから "Documents and Settings" パスにファイルをコピーします。

×
リ
ActionScript ソースファイルを編集する場合、"Documents and Settings" パスのファイルを編集します。編集の結果、コンポーネントが "壊れた" 場合、Flash を終了して再び起動したときに、機能ファイルを "First Run" ディレクトリからコピーすることにより、Flash は機能を元の状態に戻します。ただし、"First run" ディレクトリのファイルを編集した結果、コンポーネントが "壊れた" 場合、ソースファイルを元の機能ファイルの状態に戻すには、Flash を再インストールする必要があります。

コンポーネントを追加した場合は、[コンポーネント] パネルを更新する必要があります。

[コンポーネント] パネルのコンテンツを更新するには :

- [コンポーネント] パネルメニューから [リロード] を選択します。



[コンポーネント] パネルからコンポーネントを削除するには :

- "Configuration" フォルダから MXP または FLA ファイルを削除します。

コンポーネントを使用するメリット

コンポーネントを使用すると、アプリケーションの設計プロセスとコーディングプロセスを切り離すことができます。さらに、自分が作成したコンポーネントのコードはもちろん、他の開発者が作成したコンポーネントをダウンロードおよびインストールして、そのコードを再利用することもできるようになります。

コンポーネントを利用すると、プログラマーが作成した機能を、デザイナーがアプリケーションに実装できます。開発者は、頻繁に利用される機能をコンポーネントにカプセル化することができ、デザイナーは、プロパティインスペクタまたは [コンポーネントインスペクタ] パネルのパラメータを変更することにより、コンポーネントの外観とビヘイビアをカスタマイズできます。

Flash 開発者は、Adobe Exchange (www.adobe.com/jp/exchange) を利用してコンポーネントを交換することができます。コンポーネントを利用すれば、複雑な Web アプリケーションで、さまざまな要素を一から作成する必要がなくなります。新しいアプリケーションを作成するには、必要なコンポーネントを探し、それらを Flash ドキュメントで統合するだけです。


バージョン 2 コンポーネントアーキテクチャに基づくコンポーネントは、スタイルやイベント処理、スキン、フォーカス管理、深度の管理などの中心的機能を共有しています。バージョン 2 コンポーネントを初めてアプリケーションに追加する際には、この中心的機能を提供する 25 KB 程度のデータがドキュメントに追加されます。この 25 KB 分のデータは、それ以降に追加するコンポーネントによっても利用されるので、結果的には、ドキュメントのサイズはそれほど大きくなりません。コンポーネントのアップグレードの詳細については、[66 ページの「バージョン 1 コンポーネントからバージョン 2 アーキテクチャへのアップグレード」](#)を参照してください。

コンポーネントのカテゴリ

Flash に含まれているコンポーネントは、次の 5 つのカテゴリに分類できます。なお、括弧内に示すように、ActionScript ソースファイルの分類もこのカテゴリにほぼ対応しています。

■ データコンポーネント (mx.data.*)

データコンポーネントは、データソースの情報をロードおよび処理するための機能で、WebServiceConnector コンポーネントと XMLConnector コンポーネントがこれにあたります。

	データコンポーネントのソースファイルは、Flash と共にインストールされません。ただし、サポート用 ActionScript ファイルの一部はインストールされます。
---	---

■ FLVPlayback コンポーネント (mx.video.FLVPlayback)

FLVPlayback コンポーネントを使用すると、Flash アプリケーションにビデオプレーヤーを追加して、Flash Video Streaming Service (FVSS) や Flash Media Server によりプログレッシブビデオストリーミングを HTTP 上で簡単に再生できます。

■ メディアコンポーネント (mx.controls.*)

メディアコンポーネントは、ストリーミングメディアを再生および制御するための機能で、MediaController、MediaPlayback、および MediaDisplay のコンポーネントがこれにあたります。

■ ユーザーインターフェイスコンポーネント (mx.controls.*)

ユーザーインターフェイスコンポーネント ("UI コンポーネント" と呼ばれる) は、ユーザーがアプリケーションとやり取りするための機能で、RadioButton、CheckBox、TextInput などのコンポーネントがこれにあたります。

■ マネージャコンポーネント (mx.managers.*)

マネージャは、アプリケーションでフォーカスや深度などの機能を管理するための非ビジュアルコンポーネントで、FocusManager、DepthManager、PopUpManager、StyleManager、および SystemManager コンポーネントがこれにあたります。

■ スクリーンコンポーネント (mx.screensens.*)

スクリーンカテゴリには ActionScript クラスが含まれており、Flash で利用できるフォームやスライドを制御できます。

コンポーネントの一覧については、『ActionScript 2.0 コンポーネントリファレンスガイド』を参照してください。

バージョン 2 のコンポーネントアーキテクチャについて

プロパティインスペクタまたは [コンポーネントインスペクタ] パネルを使ってコンポーネントパラメータを変更することで、コンポーネントの基本機能を利用できます。ただし、コンポーネントをよりよく制御するには、API を使い、コンポーネントがどのように作成されたかをある程度理解することが必要です。

Flash のコンポーネントは、Adobe Component Architecture のバージョン 2 を使って作成されています。バージョン 2 のコンポーネントは、Flash Player 6 (6.0.79.0) 以降および ActionScript 2.0 でサポートされています。これらのコンポーネントは、バージョン 1 アーキテクチャを使用して作成されたコンポーネント (Flash MX 2004 以前にリリースされたすべてのコンポーネント) とは互換性がない場合があります。また、Flash Player 7 は元のバージョン 1 コンポーネントをサポートしていません。詳細については、[66 ページの「バージョン 1 コンポーネントからバージョン 2 アーキテクチャへのアップグレード」](#)を参照してください。

×
中

Flash MX UI コンポーネントは、Flash Player 7 以降で動作するように更新されています。ただし、更新されたこれらのコンポーネントは従来どおりバージョン 1 アーキテクチャに基づいています。Adobe Flash Exchange (www.adobe.com/go/v1_components_jp) からこれらのコンポーネントをダウンロードできます。

バージョン 2 コンポーネントは、コンパイルされたクリップ (SWC) のシンボルとして [コンポーネント] パネルに含まれます。コンパイルされたクリップは、コードがコンパイルされたコンポーネントムービークリップです。コンパイルされたクリップを編集することはできませんが、他のコンポーネントと同様に、プロパティインスペクタおよび [コンポーネントインスペクタ] パネルでパラメータを変更できます。詳細については、[19 ページの「コンパイルされたクリップと SWC ファイルについて」](#)を参照してください。

バージョン 2 コンポーネントは、ActionScript 2.0 で記述されています。各コンポーネントはクラスであり、各クラスは ActionScript のパッケージに含まれます。たとえば、ラジオボタンコンポーネントは RadioButton クラスのインスタンスで、このクラスのパッケージ名は mx.controls です。パッケージの詳細については、『ActionScript 2.0 の学習』の「パッケージについて」を参照してください。

Adobe Component Architecture のバージョン 2 で作成されたほとんどの UI コンポーネントは、UIObject クラスおよび UIComponent クラスのサブクラスであり、これらのクラスのプロパティ、メソッド、イベントをすべて継承します。また、多くのコンポーネントは、別のコンポーネントのサブクラスでもあります。コンポーネントの継承パスは、『ActionScript 2.0 コンポーネントリファレンスガイド』の各コンポーネントのセクションに示されています。



クラス階層上の FlashPaper ファイルについては、Web サイトの Flash サンプルページ (www.adobe.com/go/learn_fl_samples_jp) を参照してください。

また、すべてのコンポーネントは同じイベントモデル、CSS ベースのスタイル、ビルトインのテーマおよびスキนมカニズムを使用します。スタイルとスキンの詳細については、[81 ページ](#)、[第 5 章の「コンポーネントのカスタマイズ」](#)を参照してください。イベント処理の詳細については、[49 ページ](#)、[第 3 章の「コンポーネントの利用」](#)を参照してください。

バージョン 2 のコンポーネントアーキテクチャの詳細については、[125 ページ](#)、[第 6 章の「コンポーネントの作成」](#)を参照してください。

バージョン 2 コンポーネントの機能

ここでは、コンポーネントを使って Flash アプリケーションを構築する開発者の立場から見た、バージョン 2 コンポーネントの機能について説明し、バージョン 1 コンポーネントと比較します。コンポーネントを構築する上でのバージョン 1 アーキテクチャとバージョン 2 アーキテクチャの相違の詳細については、[125 ページ](#)、[第 6 章の「コンポーネントの作成」](#)を参照してください。

[コンポーネントインスペクタ] パネルによって、Adobe Flash および Dreamweaver でのオーサリング時にコンポーネントパラメータを変更できます。詳細については、[54 ページの「コンポーネントパラメータの設定」](#)を参照してください。

リスナーイベントモデルによって、リスナーでのイベント処理ができます。詳細については、[67 ページ](#)、[第 4 章の「コンポーネントイベントの処理」](#)を参照してください。プロパティインスペクタ内に clickHandler パラメータはありません。イベントを処理するには ActionScript コードを記述する必要があります。

スキンプロパティによって、実行時に個々のスキン (上矢印と下矢印、チェックボックスのチェックマークなど) をロードできます。詳細については、[96 ページの「コンポーネントへのスキンの適用について」](#)を参照してください。

CSS ベースのスタイルによって、アプリケーション全体の外観に一貫性を持たせることができます。詳細については、[82 ページの「スタイルによるコンポーネントのカラーとテキストの変更」](#)を参照してください。

テーマによって、ライブラリからドラッグするだけで、あらかじめ定義された外観を一連のコンポーネントに適用できます。詳細については、[108 ページの「テーマについて」](#)を参照してください。

Halo テーマを、バージョン 2 コンポーネントのデフォルトのテーマとして使用します。詳細については、[108 ページの「テーマについて」](#)を参照してください。

Manager クラスを使うと、アプリケーションのフォーカスや深さを簡単に操作できるようになります。詳細については、[61 ページの「カスタムフォーカスナビゲーションの作成」](#)および [62 ページの「ドキュメント内のコンポーネントの深度を管理する」](#)を参照してください。

UIObject クラスおよび UIComponent クラスは、これらを継承するコンポーネントに対して、核となるメソッド、プロパティ、およびイベントを提供する基本クラスです。『ActionScript 2.0 コンポーネントリファレンスガイド』の「UIComponent クラス」および「UIObject クラス」を参照してください。

SWC ファイル形式のパッケージングによって、配布が容易になり、コードの秘匿が可能となります。[125 ページ、第 6 章の「コンポーネントの作成」](#)を参照してください。

ビルトインデータバインディングが、[コンポーネントインスペクタ] パネルを通じて利用できます。詳細については、『Flash ユーザーガイド』の「データの統合」を参照してください。

クラス階層が、ActionScript 2.0 での継承が容易な構造になりました。一意の名前空間の作成や、必要に応じたクラスの読み込みができます。また、容易にサブクラスを作成してコンポーネントを拡張できます。[125 ページ、第 6 章の「コンポーネントの作成」](#)と『ActionScript 2.0 リファレンスガイド』を参照してください。



Flash 8 以降には、9 スライス (“scale-9” と呼ぶ場合もある)、高度なアンチエイリアス、ビットマップキャッシュなど、v2 コンポーネントでサポートされていない機能が存在します。

コンパイルされたクリップと SWC ファイルについて

コンパイルされたクリップとは、プリコンパイル済みの Flash シンボルと ActionScript コードのパッケージです。変更の必要がないシンボルやコードの再コンパイルを回避するために使用されます。ムービークリップも、Flash で “コンパイル” して、コンパイル済みクリップに変換することができます。たとえば、多くの ActionScript コードを含む、あまり頻繁に変更されないムービークリップなどは、コンパイルされたクリップに変換しておく効果的です。コンパイルされたクリップは、コンパイル前の通常のムービークリップと同じように動作しますが、表示やパブリッシュの処理をはるかに高速に実行できます。コンパイルされたクリップを編集することはできませんが、そのプロパティはプロパティインスペクタと [コンポーネントインスペクタ] パネルに表示されます。

Flash に含まれているコンポーネントは FLA ファイルではありません。コンパイル済みクリップ (SWC) ファイルとしてパッケージ化されています。SWC はコンポーネントを配布するための Adobe ファイル形式であり、コンパイルされたクリップ、コンポーネントの ActionScript クラスファイル、およびコンポーネントを記述するその他のファイルが含まれています。SWC ファイルの詳細については、[180 ページの「コンポーネントの書き出しと配布」](#)を参照してください。

SWC ファイルを "First Run/Components" フォルダに格納すると、そのコンポーネントが [コンポーネント] パネルに表示されます。[コンポーネント] パネルからコンポーネントをステージに追加すると、コンパイルされたクリップのシンボルがライブラリに追加されます。

ムービークリップをコンパイルするには：

- [ライブラリ] パネルにあるムービークリップを右クリック (Windows) または Control キーを押しながらクリック (Macintosh) し、[コンパイルされたクリップに変換] を選択します。

SWC ファイルを書き出すには：

- [ライブラリ] パネルにあるムービークリップを右クリック (Windows) または Control キーを押しながらクリック (Macintosh) し、[SWC ファイル書き出し] を選択します。



Flash での FLA コンポーネントのサポートは継続されます。

アクセシビリティとコンポーネント

最近の傾向として、Web コンテンツは「アクセシブル」であること、つまり、さまざまな障害を持つユーザーにとっても利用しやすいものであることが求められています。Flash アプリケーションのビジュアルコンテンツはこの点に対応しており、スクリーンリーダーから画面の内容が音声出力されるように設定すれば、視覚障害のあるユーザーにもご利用いただけます。

コンポーネントを作成するとき、作者は、そのコンポーネントとスクリーンリーダーとの間で通信を可能にする `ActionScript` を作成します。こうしておけば、開発者がコンポーネントを使って Flash のアプリケーションを作成する際に、[アクセシビリティ] パネルを利用して、各コンポーネントのインスタンスを設定できます。

アドビ システムズ社が作成したコンポーネントの大部分はアクセシビリティを重視してデザインされています。コンポーネントがアクセス可能かどうかを調べるには、『`ActionScript 2.0` コンポーネントリファレンスガイド』の説明を参照してください。Flash でアプリケーションを作成するとき、各コンポーネントにコード (`mx.accessibility.ComponentNameAccImpl.enableAccessibility();`) を追加し、[アクセシビリティ] パネル内のアクセシビリティパラメータを設定する必要があります。コンポーネントのアクセシビリティは、Flash のすべてのムービークリップのアクセシビリティと同じように機能します。

アドビ システムズ社が作成したコンポーネントの大部分は、キーボードからも操作が可能です。『`ActionScript 2.0` コンポーネントリファレンスガイド』の各コンポーネントの説明は、キーボードでコンポーネントを操作できるかどうかを示します。

コンポーネントを使用したアプリケーションの作成

コンポーネントとは、Flash アプリケーションの作成時に使用できる作成済みのエレメントです。コンポーネントの種類としては、ユーザーインターフェイスコントロール、データへのアクセスと接続のメカニズム、およびメディア関連のエレメントなどがあります。すべてを一から作成する代わりにあらかじめ用意されているエレメントやビヘイビアを利用できるので、コンポーネントを使用すると、Flash アプリケーションの作成に必要な作業量を低減できます。

この章には、Flash CS3 Professional で使用可能なコンポーネントを使って Flash アプリケーションを作成する方法を示すチュートリアルがあります。これにより、Flash オーサリング環境でコンポーネントをどのように操作すればよいかがわかります。また、ActionScript コードを使用してコンポーネントをインタラクティブにする方法も学習できます。

お詫びギフトショップのチュートリアルについて

このチュートリアルでは、" お詫びの品 " ギフトサービスを提供する基本的なオンラインショッピングアプリケーションの作成手順について説明します。このサービスは、迷惑をかけた相手に贈る適切なお詫びの品を選ぶユーザーの手助けをします。アプリケーションは迷惑の度合いに見合うギフトのリストをフィルタ処理して表示します。ユーザーはそのリストから商品を選んでショッピングカートに入れ、チェックアウトのページに進んで、請求先、配送先およびクレジットカードの情報を入力できます。

この章では、次のセクションについて説明します。

お詫びギフトショップのチュートリアルについて	21
メインページの作成	23
お勧めギフトの表示用データコンポーネントのバインディング	29
ギフトの詳細の表示	33
チェックアウトスクリーンの作成	38
アプリケーションのテスト	47
完成済みアプリケーションの参照	47

このアプリケーションのインターフェイスは、ComboBox、DataGrid、TextArea、Button およびその他のコンポーネントを使用して作成します。インターフェイスのメインページは次のようになります。

what did you do?

Forgot to water your plants

gift ideas

Name	Price
The Sounds of Romance - CD	\$12.98 each
Diamond Engagement Ring	\$19124.99 each
Kitten	\$149.99 each
Bouquet of Flowers - Extreme	\$49.99 each
Cash (cold-hard)	\$100 stack

cart

Quantity	Product	Price	Subtotal
----------	---------	-------	----------

Clear Cart

Checkout

このアプリケーションでは、コンボボックスに表示する失敗内容のリスト (problems.xml) を取得するために、ActionScript の WebService クラスを使用して Web サービスに動的に接続します。また、ユーザーによるインタラクティブ操作の処理にも ActionScript を使用します。

インターフェイスと他のデータソースとの接続にはデータコンポーネントを使用します。ギフトリストの XML データファイル (products.xml) への接続に XMLConnector コンポーネントを使用し、データのフィルタ処理とデータグリッドでの表示に DataSet コンポーネントを使用します。

このチュートリアルを実施するには、Flash オーサリング環境をある程度使い慣れていることと、ActionScript の若干の使用経験が必要です。オーサリング環境で、パネル、ツール、タイムライン、およびライブラリを使用した経験が望まれます。このチュートリアルには、サンプルアプリケーションを作成するために必要なすべての ActionScript が提供されています。ただし、スクリプトの概念を理解して独自のアプリケーションを作成するには、ActionScript を記述した経験もあることが必要となります。

このアプリケーションの動作可能な完成済みバージョンについては、[47 ページの「完成済みアプリケーションの参照」](#)を参照してください。

サンプルアプリケーションは、アプリケーションの作成方法を説明するためのものであり、実用的なアプリケーションに求められる高いレベルの完成度を備えてはいません。

メインページの作成

基礎となる骨格のみのページに、次に示す手順に従ってコンポーネントを追加し、アプリケーションのメインページを作成します。次に、ActionScript コードを追加してそれらのコンポーネントをカスタマイズし、アプリケーションのコンポーネントを操作できるようにするための ActionScript クラスを読み込み、Web サービスにアクセスして迷惑をかけた行為のリストをコンボボックスに設定します。コードでは、その Web サービスの結果を受け取るようにコンボボックスの dataProvider プロパティを設定することにより、コンボボックスの内容を設定しています。

1. "first_app_start.fla" ファイルを開きます。サンプルの .fla ファイルについては、Flash サンプルページ (www.adobe.com/go/learn_fl_samples_jp) を参照してください。

このファイルには、次のような外観の基礎ページが含まれています。



"start_app.fla" ファイルには、3つのレイヤーが含まれています。それは、黒色の背景イメージとテキストタイトルから成る Background レイヤー、アプリケーションのセクションのテキストラベルから成る Text レイヤー、および 1 番目のフレーム (Home) と 10 番目のフレーム (Checkout) のラベルから成る Labels レイヤーです。

2. [ファイル]-[名前を付けて保存] を選択します。ファイル名を変更し、それをハードディスクに保存します。
3. タイムラインで Labels レイヤーを選択し、[レイヤーの追加] ボタンをクリックして上位に新規レイヤーを追加します。新規レイヤーに **Form** という名前を指定します。これはコンポーネントインスタンスを配置するレイヤーです。

4. Form レイヤーを選択します。[コンポーネント] パネル ([ウィンドウ]-[コンポーネント]) で、ComboBox コンポーネントを探します。ComboBox のインスタンスをステージにドラッグします。これを、[What Did You Do?] の文字列の下に配置します。プロパティインスペクタ ([ウィンドウ]-[プロパティ]-[プロパティ]) で、インスタンス名として「**problems_cb**」を入力します。幅として「**400**」(ピクセル)を入力します。x 座標として「**76.0**」を、y 座標として「**82.0**」を入力します。

× ❗	ComboBox コンポーネントのシンボルがライブラリに追加されています ([ウィンドウ]-[ライブラリ])。コンポーネントのインスタンスをステージにドラッグすると、コンポーネントのコンパイルされたクリップシンボルがライブラリに追加されます。Flash 内のすべてのシンボルと同じように、ライブラリシンボルをステージにドラッグすることによって、コンポーネントのインスタンスをさらに作成できます。
--------	---

5. DataGrid コンポーネントのインスタンスを、[コンポーネント] パネルからステージにドラッグします。これを [Gift Ideas] の文字列の下に配置します。インスタンス名として「**products_dg**」を入力します。幅として「**400**」(ピクセル)を、高さとして「**130**」を入力します。x 座標として「**76.0**」を、y 座標として「**128.0**」を入力します。
6. DataSet コンポーネントのインスタンスを、[コンポーネント] パネルからステージの端にドラッグします。DataSet コンポーネントは、実行時にアプリケーション内で表示されません。DataSet アイコンは、Flash オーサリング環境で操作する場合のプレースホルダに過ぎません。インスタンス名として「**products_ds**」を入力します。

XMLConnector コンポーネントのインスタンスを、[コンポーネント] パネルからステージの端にドラッグします。DataSet コンポーネントと同様、XMLConnector コンポーネントも実行時にアプリケーション内で表示されません。インスタンス名として「**products_xmlcon**」を入力します。プロパティインスペクタの [パラメータ] タブをクリックし、URL プロパティとして「**http://www.flash-mx.com/mm/firstapp/products.xml**」を入力します。direction プロパティの値をクリックしてコンボボックスをアクティブにし、下矢印をクリックして、リストから receive を選択します。

× ❗	コンポーネントのパラメータは、[コンポーネントインスペクタ] パネル ([ウィンドウ]-[コンポーネントインスペクタ]) で設定することもできます。プロパティインスペクタの [パラメータ] タブと [コンポーネントインスペクタ] パネルの [パラメータ] タブは同じ働きです。
--------	--

URL は、アプリケーションの [Gift Ideas] セクションに表示される商品データを含む外部 XML ファイルを指定します。後の手順では、データバインディングを使用し、XMLConnector、DataSet、および DataGrid の各コンポーネントをバインドします。DataSet コンポーネントは外部 XML ファイルのデータにフィルタを適用し、DataGrid コンポーネントはそれを表示します。

7. Button コンポーネントのインスタンスを、[コンポーネント] パネルからステージにドラッグします。これをステージの右下隅に配置します。インスタンス名として「**checkout_button**」を入力します。[パラメータ] タブをクリックし、label プロパティとして「**Checkout**」を入力します。x 座標に「**560.3**」、y 座標に「**386.0**」を入力します。

コンポーネントクラスの読み込み

各コンポーネントには、メソッドとプロパティを定義する **ActionScript** クラスファイルが関連付けられています。このセクションでは、アプリケーションの各コンポーネントに関連付けられているクラスを読み込む **ActionScript** コードを追加します。いくつかのコンポーネントについては、既にインスタンスがステージに追加されています。それ以外のものについては、後の手順で追加する **ActionScript** によりインスタンスを動的に作成します。

import ステートメントを使用するとクラス名への参照が作成され、該当するコンポーネントに関する **ActionScript** を容易に記述できます。クラスの参照時に、パッケージ名を含む完全な名前ではなくクラス名を指定すれば済むようになります。たとえば、**import** ステートメントで **ComboBox** クラスへの参照を作成した後は<インスタンス名>:ComboBox というシンタックスでコンボボックスのインスタンスを表すことができ、<インスタンス名>:mx.controls.ComboBox と記述する必要はありません。

パッケージとは、所定のクラスパスディレクトリ内にあって、クラスファイルを格納しているディレクトリのことです。ワイルドカード文字を使用すると、パッケージ内のすべてのクラスに対して参照を作成できます。たとえば、mx.controls.* シンタックスを指定すると、controls パッケージ内のすべてのクラスに対する参照が作成されます。ワイルドカード文字を使用してパッケージの参照を作成しても、使用しないクラスはコンパイル時にアプリケーションから除外されるため、サイズが余分に増えることはありません。

このチュートリアルアプリケーションでは、次に示すパッケージと個別クラスが必要です。

UI コンポーネントコントロールパッケージ このパッケージには、ComboBox、DataGrid、Loader、TextInput、Label、NumericStepper、Button、CheckBox など、ユーザーインターフェイスコントロールコンポーネントのクラスが含まれています。

UI コンポーネントコンテナパッケージ このパッケージには、Accordion、ScrollPane、Window など、ユーザーインターフェイスコンテナコンポーネントのクラスが含まれています。コントロールパッケージと同様に、このパッケージへの参照もワイルドカード文字を使用して作成できます。

DataGridColumn クラス このクラスを使用すると、DataGrid インスタンスに列を追加して、その外観を制御できます。

WebService クラス ComboBox インスタンスの内容として、迷惑をかけた行為のリストを設定します。このクラスのために、クラスのサンプルライブラリから **WebServiceClasses** アイテムも読み込む必要があります。このアイテムには、アプリケーションの SWF ファイルをコンパイルおよび生成するために必要なコンパイル済みクリップ (SWC) ファイルが含まれています。

Cart クラス このチュートリアルで用意されているカスタムクラスです。Cart クラスは、後で作成するショッピングカートの機能を定義します。Cart クラスファイルのコードを確認するには、アプリケーションの FLA ファイルおよび SWF ファイルが共に格納されている "component_application" フォルダにある "cart.as" ファイルを開きます。

これらのクラスを読み込むには、Actions レイヤーを作成し、メインタイムラインの最初のフレームに ActionScript コードを追加します。これ以降、このチュートリアルでアプリケーションに追加するコードはすべて Actions レイヤーに配置してください。

1. クラスのライブラリから **WebServiceClasses** アイテムを読み込むには、[ウィンドウ]-[サンプルライブラリ]-[クラス] を選択します。
2. **WebServiceClasses** アイテムを、クラスライブラリからアプリケーションのライブラリまでドラッグします。

クラスライブラリからアイテムを読み込む方法は、このライブラリにコンポーネントを追加する方法と似ています。具体的には、ライブラリにこのクラスの SWC ファイルを追加します。アプリケーションでこのクラスを使用するには、SWC ファイルがライブラリ内にある必要があります。

3. タイムラインで、Form レイヤーを選択し、[レイヤーの追加] ボタンをクリックします。新規レイヤーに **Actions** という名前を指定します。
4. Actions レイヤーを選択した状態で、フレーム 1 を選択し、F9 キーを押して [アクション] パネルを開きます。
5. [アクション] パネルで次のコードを入力して、再生中にアプリケーションがループすることを回避する `stop()` 関数を作成します。

```
stop();
```

6. Actions レイヤーでフレーム 1 を選択した状態で、[アクション] パネルに次のコードを追加し、クラスを読み込みます。

```
// 必須クラスを読み込む。
import mx.services.Webservice;
import mx.controls.*;
import mx.containers.*;
import mx.controls.gridclasses.DataGridColumn;
// Cart カスタムクラスを読み込む。
import Cart;
```

コンポーネントインスタンスのデータ型の設定

ここまでの手順でステージにドラッグした各コンポーネントインスタンスに、データ型を割り当てます。

ActionScript 2.0 では厳密な型指定を使用するため、データ型は変数を作成するときに割り当てます。厳密な型指定によって、[アクション] パネル内の変数に対するコードヒントが表示されるようになります。

- [アクション] パネルに、これまでに作成した 4 つのコンポーネントにデータ型を割り当てる次のコードを追加します。

```
/* ステージ上のインスタンスにデータ型を指定する。他のインスタンスは  
   実行時に Cart クラスから追加される。*/  
var problems_cb:ComboBox;  
var products_dg:DataGrid;  
var cart_dg:DataGrid;  
var products_xmlcon:mx.data.components.XMLConnector;
```



ここで指定するインスタンス名は、コンポーネントをステージにドラッグしたとき割り当てたインスタンス名と一致している必要があります。

コンポーネントの外観のカスタマイズ

各コンポーネントにはスタイルプロパティとスタイルメソッドが備わっており、ハイライトカラー、フォント、フォントサイズを含むコンポーネントの外観をカスタマイズできます。スタイルは、個々のコンポーネントインスタンスごとに設定することも、グローバルなスタイルを設定してアプリケーション内のすべてのコンポーネントインスタンスに適用することもできます。このチュートリアルでは、グローバルなスタイルを設定します。

- 次のコードを追加してスタイルを設定します。

```
// グローバルスタイルおよび problems_cb ComboBox のイー징ング式を定義する。  
_global.style.setStyle("themeColor", "haloBlue");  
_global.style.setStyle("fontFamily", "Verdana");  
_global.style.setStyle("fontSize", 10);  
_global.style.setStyle("openEasing", mx.transitions.easing.Bounce.easeOut);
```

このコードによって、コンポーネントのテーマカラー (選択項目のハイライトカラー)、フォント、およびフォントサイズが設定され、ComboBox のイー징ングも設定されます。つまり、ComboBox タイトルバーをクリックしたときのドロップダウンリストの表示および非表示の方法が設定されます。

コンボボックスでの迷惑行為のリスト表示

このセクションでは、迷惑行為 ([Forgot to Water Your Plants] など) のリストが格納されている Web サービスに接続するコードを追加します。使用する WSDL (Web Service Description Language) ファイルは、www.flash-mx.com/mm/firstapp/problems.cfc?WSDL にあります。WSDL の構造を確認するには、ブラウザで WSDL の URL にアクセスしてください。

この `ActionScript` コードでは、Web サービスの結果を `ComboBox` インスタンスに渡して表示します。1つの関数を使用して、迷惑の度合いの順に迷惑行為を並べ替えます。Web サービスがダウンしていたり、関数が見つからないことなどが原因で Web サービスから結果が返されない場合は、[出力] パネルにエラーメッセージが表示されます。

- [アクション] パネルに次のコードを追加します。

```
/* 迷惑の配列を取得する Web サービスを定義する。  
このサービスは problems_cb ComboBox インスタンスにバインドされる。*/  
var problemService:WebService = new WebService("http://www.flash-mx.com/mm/  
    firstapp/problems.cfc?WSDL");  
var myProblems:Object = problemService.getProblems();  
  
/* Web サービスから結果が返された場合は、列ラベルとして使用されるフィールドを設定する。  
データプロバイダを、Web サービスから返された結果に設定する。*/  
myProblems.onResult = function(wsdResults:Array) {  
    problems_cb.labelField = "name";  
    problems_cb.dataProvider = wsdResults.sortOn("severity", Array.NUMERIC);  
};  
  
/* リモート Web サービスに接続できない場合は、  
[出力] パネルにエラーメッセージを表示する。*/  
myProblems.onFault = function(error:Object) {  
    trace("error:");  
    for (var prop in error) {  
        trace("  "+prop+" -> "+error[prop]);  
    }  
};
```

ペ
ネ
ル

Control+S を押して作業内容を保存してから、Control+Enter を押して (または [制御]-[ムービープレビュー] を選択して) アプリケーションをテストします。この時点では、コンボボックスの内容として迷惑行為のリストが設定されること、[Gift Ideas] として作成した空のデータグリッドが表示されること、チェックアウトボタンが表示されることを確認してください。

お勧めギフトの表示用データコンポーネントのバインディング

チュートリアル最初の段階で、DataGrid、DataSet、および XMLConnector の各コンポーネントをステージに追加しました。また、products_xmlcon という XMLConnector インスタンスの URL プロパティには、アプリケーションの [Gift Ideas] セクションに表示する商品情報が格納されている XML ファイルの場所を設定しました。

ここでは、Flash オーサリング環境のデータバインディング機能を使って、XMLConnector、DataSet、および DataGrid コンポーネントをバインドし、アプリケーションの中で XML データを使用できるようにします。

コンポーネントをバインドすると、[What Did You Do?] セクションでユーザーが選択する迷惑の度合いに応じて、DataSet コンポーネントによるフィルタが XML ファイル内の商品リストに適用されます。その結果は DataGrid コンポーネントに一覧表示されます。

XML データソースの構造を表すスキーマの指定

XMLConnector コンポーネントを使って外部 XML データソースに接続する場合は、"スキーマ" を指定する必要があります。スキーマとは、XML ドキュメントの構造を表す概念図です。スキーマによって、XMLConnector コンポーネントは XML データソースの読み取り方法を理解します。スキーマを指定する最も簡単な方法は、接続しようとしている XML ファイルのコピーを読み込み、そのコピーをスキーマとして使用する方法です。

1. Web ブラウザを開き、www.flash-mx.com/mm/firstapp/products.xml (XMLConnector の URL パラメータで設定した場所) に移動します。
2. [ファイル]-[名前を付けて保存] を選択します。
3. "products.xml" を、作業対象の FLA ファイルと同じ場所に保存します。
4. メインタイムラインでフレーム 1 を選択します。
5. ステージの横で、products_xmlcon (XMLConnector) インスタンスを選択します。
6. [コンポーネントインスペクタ] パネルで、[スキーマ] タブをクリックします。[スキーマ] タブの右側でスクロールペインの上にある [読み込み] ボタンをクリックします。[開く] ダイアログボックスで、手順 3 で読み込んだ "products.xml" ファイルを検索し、[開く] をクリックします。"products.xml" ファイルのスキーマが [スキーマ] タブのスクロールペインに表示されます。

[スキーマ] タブの上部ペインで、image エレメントを選択します。下部ペインで、data_type を選択し、value フィールドを <empty> から String に変更します。この手順を description 要素についても繰り返します。

迷惑の種類に応じたお勧めギフトのフィルタ処理

XMLConnector、DataSet、および DataGrid の各コンポーネントインスタンスを相互にバインドするには、[コンポーネントインスペクタ] パネルの [バインディング] タブを使用します。

1. ステージ上の `products_xmlcon` (XMLConnector) インスタンスを選択した状態で、[コンポーネントインスペクタ] パネルの [バインディング] タブをクリックします。
2. [バインディングの追加] ボタンをクリックします。
3. [バインディングの追加] ダイアログボックスで、`results.products.product: array` アイテムを選択し、[OK] をクリックします。
4. [バインディング] タブの下部にある、属性の名前と値のペアが示されているバインディング属性ペインの [バインド] アイテムをクリックします。
5. [バインド] アイテムの [値] 列で、虫めがねのアイコンをクリックして [バインド] ダイアログボックスを開きます。
6. [バインド] ダイアログボックスのコンポーネントのパスペインで、`<products_ds> DataSet` インスタンスを選択します。スキーマの場所ペインで、`dataProvider:array` を選択します。[OK] をクリックします。
7. [バインディング] タブで、バインディング属性ペインの [方向] アイテムをクリックします。[値] 列のポップアップメニューから [アウト] を選択します。

このオプションは、データが `products_xmlcon` インスタンスから `products_ds` インスタンスに渡される (双方向の受け渡しは発生しない、つまり、`DataSet` インスタンスから `XMLConnector` インスタンスには渡されない) ことを意味します。

8. ステージ上で、`products_ds` インスタンスを選択します。[コンポーネントインスペクタ] パネルの [バインディング] タブでは、上部のバインディングリストペインにコンポーネントのデータプロバイダが表示されます。バインディング属性ペインの [バインド] パラメータによって、`products_ds` インスタンスが `products_xmlcon` インスタンスにバインドされていて、バインド方向は [イン] であることが示されます。

次に示す手順では、`DataSet` インスタンスを `DataGrid` インスタンスにバインドして、データセットによってフィルタが適用されたデータをデータグリッドに表示します。

9. `products_ds` インスタンスを選択したままの状態、[バインディング] タブの [バインディングの追加] ボタンをクリックします。
10. [バインディングの追加] ダイアログボックスで、`dataProvider: array` アイテムを選択し、[OK] をクリックします。
11. [バインディング] タブのバインディングリストで `dataProvider: array` アイテムが選択されていることを確認します。

12. バインディング属性ペインの [バインド] アイテムをクリックします。
13. [バインド] アイテムの [値] 列で、虫めがねのアイコンをクリックして [バインド] ダイアログボックスを開きます。
14. [バインド] ダイアログボックスのコンポーネントのパスペインで、products_dg (DataGrid) インスタンスを選択します。スキーマの場所ペインで、dataProvider:array を選択します。[OK] をクリックします。

[Gift Ideas] セクションへの列の追加

商品の情報と価格を表示するために、アプリケーションの [Gift Ideas] セクション内のデータグリッドに列を追加します。

- Actions レイヤーを選択します。DataGrid インスタンスで "Name" 列と "Price" 列を作成、設定、および追加する次のコードを、[アクション] パネルに追加します。

```
// products_dg DataGrid インスタンス内の各データグリッド列と
// デフォルト幅を定義する。
var name_dgc:DataGridColumn = new DataGridColumn("name");
name_dgc.headerText = "Name";
name_dgc.width = 280;

// DataGrid に列を追加する。
products_dg.addColumn(name_dgc);
var price_dgc:DataGridColumn = new DataGridColumn("price");
price_dgc.headerText = "Price";
price_dgc.width = 100;

// 列のラベルを実行時に設定する
// 関数を定義する。
price_dgc.labelFunction = function(item:Object) {
    if (item != undefined) {
        return "$"+item.price+" "+item.priceQualifier;
    }
};
products_dg.addColumn(price_dgc);
```

XMLConnector のトリガ

次に、XMLConnector インスタンスによる "products.xml" リモートファイルのコンテンツのロード、解析、およびバインドを実現するコードを追加します。このファイルは、前に作成した XMLConnector インスタンスの URL プロパティに入力した URL にあります。ファイルには、アプリケーションの [Gift Ideas] セクションに表示される商品の情報が格納されています。

- [アクション] パネルに次のコードを追加します。

```
products_xmlcon.trigger();
```

お勧めギフトをフィルタ処理するイベントリスナーの追加

このセクションでは、ユーザーが [What Did You Do?] セクション (problems_cb ComboBox インスタンス) で失敗内容を選択したことを検出するイベントリスナーを追加します。リスナーには、ユーザーが選択する失敗内容に応じて [Gift Ideas] リストにフィルタを適用する関数が含まれています。生じた迷惑の度合いが低い場合は控えめな品 (CD や花など) のリストが表示され、生じた迷惑の度合いが高い場合は、高価な品が表示されます。

イベントリスナーの利用の詳細については、『ActionScript 2.0 の学習』の「イベントリスナーの使用」を参照してください。

- [アクション] パネルに次のコードを追加します。

```
/* problems_cb ComboBox インスタンスにリスナーを定義する。  
このリスナーは、DataSet ( および DataGrid ) 内の商品にフィルタを適用する。  
フィルタは、ComboBox 内で現在選択されているアイテムの重大度に基づく。*/  
var cbListener:Object = new Object();  
cbListener.change = function(evt:Object) {  
    products_ds.filtered = false;  
    products_ds.filtered = true;  
    products_ds.filterFunc = function(item:Object) {  
        // 現在のアイテムの重大度が、ComboBox 内で選択されている  
        // アイテムの重大度以上の場合は true を返す。  
        return (item.severity>=evt.target.selectedItem.severity);  
    };  
};  
  
// ComboBox にリスナーを追加する。  
problems_cb.addEventListener("change", cbListener);
```

change() 関数の先頭で filtered プロパティをリセットする (false に設定し、それから true に設定する) ことで、ユーザーが [What Did You Do?] の選択を何度変更しても関数は正しく動作します。

filterFunc() 関数は、多数のギフトの中の指定アイテムが、ユーザーがコンボボックスで選択した重大度の範囲に収まるかどうかを確認します。選択した重大度の範囲にギフトが該当する場合は、DataSet インスタンスにバインドされている DataGrid インスタンスにそれが表示されます。

最後のコード行では、problems_cb ComboBox インスタンスにリスナーを登録します。

ショッピングカートの追加

次に、カスタム Cart クラスのインスタンスを作成および初期化するコードを追加します。

- [アクション] パネルに次のコードを追加します。

```
var myCart:Cart = new Cart(this);  
myCart.init();
```

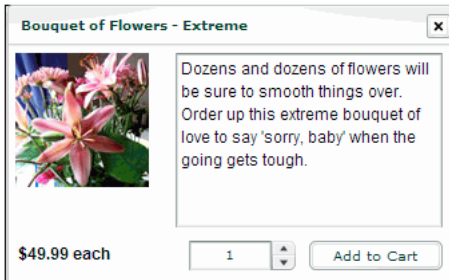

このコードでは、Cart クラスの `init()` メソッドを使って DataGrid インスタンスをステージに追加し、列を定義し、DataGrid インスタンスをステージに配置しています。Button コンポーネントインスタンスの追加と配置、およびボタンの Alert ハンドラの追加も行われます。Cart クラスの `init()` メソッドのコードを確認するには、"Cart.as" ファイルを開いてください。

マウス

Control+S を押して作業内容を保存してから、Control+Enter を押して (または [制御]-[ムービープレビュー] を選択して) アプリケーションをテストします。コンボボックスで迷惑行為を選択すると、[Gift Ideas] として作成したデータグリッドに、迷惑の度合いに見合うギフトのみに絞り込まれたリストが表示されることを確認してください。

ギフトの詳細の表示

ユーザーが [Gift Ideas] セクションにある商品をクリックすると、アプリケーション内にポップアップウィンドウが表示されます。ポップアップウィンドウには、テキストによる説明、イメージ、および価格を含む、商品に関する情報を表示するコンポーネントインスタンスが含まれています。このポップアップウィンドウを作成するには、ムービークリップシンボルを作成し、Loader、TextArea、Label、NumericStepper、および Button コンポーネントのインスタンスを追加します。たとえば、ギフト "Bouquet of Flowers Extreme" の商品詳細ウィンドウは次のように表示されます。



後の手順では、各商品のムービークリップのインスタンスを動的に作成する `ActionScript` を追加します。これらのムービークリップインスタンスは、既にライブラリに追加した Window コンポーネントに表示されます。コンポーネントインスタンスには、外部 XML ファイルの要素が設定されます。

1. Window コンポーネントのインスタンスを [コンポーネント] パネルからライブラリにドラッグします。

Window コンポーネントシンボルがライブラリに追加されます。後の手順では、`ActionScript` を使って Window コンポーネントのインスタンスを作成します。

2. [ライブラリ] パネル ([ウィンドウ]-[ライブラリ]) で、タイトルバーの右側にあるオプションメニューをクリックし、[新規シンボル] を選択します。

3. [新規シンボルの作成] ダイアログボックスで、[名前]に「ProductForm」を入力し、[タイプ]で[ムービークリップ]を選択します。
4. [詳細] ボタンをクリックします。[リンケージ]の[ActionScript に書き出し]を選択し、[最初のフレームに書き出し]を選択したままにして、[OK]をクリックします。新規シンボルのドキュメントウィンドウが、シンボル編集モードで開きます。

ライブラリにはあるがステージにはないムービークリップシンボルの場合は、ActionScript を使ってこれを操作できるように、[ActionScript に書き出し]を選択する必要があります。最初のフレームに書き出すことによって、最初のフレームがロードされると同時にムービークリップを使用できるようになります。後の手順では、ユーザーが[Gift Ideas]セクション内の商品をクリックするたびにムービークリップのインスタンスを動的に作成する ActionScript を追加します。

5. 新規シンボルのタイムラインでレイヤー1を選択し、その名前を **Components** に変更します。
6. Loader コンポーネントのインスタンスを、[コンポーネント]パネルからステージにドラッグします。x 座標と y 座標に対しそれぞれ「5」を入力します。インスタンス名として「image_ldr」を入力します。プロパティインスペクタで[パラメータ]タブをクリックします。autoLoad として false を選択し、scaleContent として false を選択します。

Loader コンポーネントインスタンスは、商品のイメージを表示するときに使用されます。autoLoad の false 設定は、イメージが自動的にロードされないことを指定します。scaleContent の false 設定は、イメージが拡大または縮小されないことを指定します。後の手順では、ユーザーが[Gift Ideas]セクションで選択する商品に基づいて、イメージを動的にロードするコードを追加します。

7. TextArea コンポーネントのインスタンスを、[コンポーネント]パネルからステージにドラッグします。これを Loader コンポーネントの横に配置します。x 座標に「125」、y 座標に「5」を入力します。インスタンス名として「description_ta」を入力します。幅を 200、高さを 130 に設定します。プロパティインスペクタで[パラメータ]タブをクリックします。editable には false を選択します。html には true を選択します。wordWrap には true を選択します。

この TextArea コンポーネントインスタンスは、選択した商品の説明テキストを表示するときに使用します。選択した設定は、ユーザーがテキストを編集できないこと、HMTL タグを使ってフォーマットできること、およびテキスト領域のサイズに合わせて行が回り込み表示されることをそれぞれ指定するものです。

8. Label コンポーネントのインスタンスを、[コンポーネント]パネルからステージにドラッグします。これを Loader コンポーネントの下に配置します。x 座標を 5、y 座標を 145 に設定します。インスタンス名として「price_lbl」を入力します。プロパティインスペクタで[パラメータ]タブをクリックします。autoSize には left を選択します。html には true を選択します。

この Label コンポーネントインスタンスには、商品の価格と単位数量("/個"または"/ダース"など、指定価格の基準となる商品の数量)を表示します。

9. **NumericStepper** コンポーネントのインスタンスを、[コンポーネント] パネルからステージにドラッグします。これを **TextArea** コンポーネントの下に配置します。x 座標を **135**、y 座標を **145** に設定します。インスタンス名として「**quantity_ns**」を入力します。プロパティインスペクタで [パラメータ] タブをクリックします。minimum には「**1**」を入力します。

minimum を 1 に設定するのは、ユーザーが商品を少なくとも 1 つ選択しないとアイテムをカートに追加できないようにするためです。

10. **Button** コンポーネントのインスタンスを、[コンポーネント] パネルからステージにドラッグします。これを **NumericStepper** コンポーネントの横に配置します。x 座標を **225**、y 座標を **145** に設定します。インスタンス名として「**addToCart_button**」を入力します。プロパティインスペクタで [パラメータ] タブをクリックします。label には「**Add To Cart**」を入力します。

商品の詳細表示をトリガするイベントリスナーの追加

次に、各商品の情報を表示するイベントリスナーを **products_dg** **DataGrid** インスタンスに追加します。ユーザーが [Gift Ideas] セクション内の商品をクリックすると、ポップアップウィンドウが表示され、そこに商品に関する情報が表示されます。

- メインタイムラインの [アクション] パネルに次のコードを追加します。

```
// DataGrid 内の行が変更されたことを検出する
// DataGrid のリスナーを作成する。
var dgListener:Object = new Object();
dgListener.change = function(evt:Object) {
    // DataGrid 内の現在の行が変更されたら、
    // 商品の詳細を表示する新規ポップアップウィンドウを起動する。
    myWindow = mx.managers.PopUpManager.createPopUp(_root,
        mx.containers.Window, true, {title:evt.target.selectedItem.name,
        contentPath:"ProductForm", closeButton:true});
    // ポップアップウィンドウのサイズを設定する。
    myWindow.setSize(340, 210);
    // ユーザーが閉じるボタンをクリックしたら
    // ポップアップウィンドウを閉じるリスナーを定義する。
    var closeListener:Object = new Object();
    closeListener.click = function(evt) {
        evt.target.deletePopUp();
    };
    myWindow.addEventListener("click", closeListener);
};
products_dg.addEventListener("change", dgListener);
```

このコードによって **dgListener** という新規イベントリスナーが作成され、以前にライブラリに追加した **Window** コンポーネントのインスタンスが作成されます。新規ウィンドウのタイトルは商品の名前に設定されます。ウィンドウのコンテンツパスは **ProductForm** ムービークリップに設定されます。ウィンドウのサイズは **340x210** ピクセルに設定されます。

情報を表示した後にウィンドウを閉じるための閉じるボタンも追加されています。

ProductForm ムービークリップへのコードの追加

次に、作成した ProductForm ムービークリップに ActionScript を追加します。ActionScript では、選択したギフトの情報をムービークリップのコンポーネントに設定し、選択した商品をカートに追加するイベントリスナーを [Add to Cart] ボタンに追加します。

イベントリスナーの利用の詳細については、『ActionScript 2.0 の学習』の「イベントリスナーの使用」を参照してください。

1. ProductForm ムービークリップのタイムラインで、新規レイヤーを作成し、**Actions** という名前を指定します。Actions レイヤーの最初のフレームを選択します。
2. [アクション] パネルに次のコードを追加します。

```
// DataGrid 内の選択商品アイテムを参照するオブジェクトを作成する。  
var thisProduct:Object = this._parent._parent.products_dg.selectedItem;  
// description_ta TextArea インスタンスと price_lbl Label インスタンスに  
// 選択した商品に関するデータを設定する。  
description_ta.text = thisProduct.description;  
price_lbl.text = "<b>${thisProduct.price+ " "+thisProduct.priceQualifier+"</b>";  
// アプリケーションディレクトリから商品のイメージをロードする。  
image_ldr.load(thisProduct.image);
```



コードには、コードの目的を説明するコメントが含まれています。記述するすべての ActionScript コードに、このようなコメントを挿入しておくといよいでしょう。自分や別の人が後でコードを使用するときに、その内容を簡単に理解できます。

コードの先頭では、以降のコード内で選択商品を参照する変数を定義しています。thisProduct 変数を定義すると、指定の商品を参照するために、

this._parent._parent.products_dg.selectedItem というパスを使用しなくて済みます。

次に、thisProduct オブジェクトの description、price、および priceQualifier プロパティを使って、TextArea インスタンスと Label インスタンスが設定されています。これらのプロパティは、このチュートリアル最初に products_xmlcon XMLConnector インスタンスにリンクした products.xml ファイル内のエレメントに対応しています。このチュートリアルの後半では、XMLConnector、DataSet、および DataGrid コンポーネントインスタンスをバインドします。これにより、この XML ファイル内のエレメントは、他の 2 つのコンポーネントインスタンスで使用されます。

最後に、thisProduct オブジェクトの image プロパティを使って、商品のイメージを Loader コンポーネントにロードしています。

3. ユーザーが [Add to Cart] ボタンをクリックしたときに商品をカートに追加するイベントリスナーを追加します。後の手順では、アプリケーションのメインタイムラインに **ActionScript** を追加して、**Cart** クラスのインスタンスを作成します。次のコードを追加します。

```
var cartListener:Object = new Object();
cartListener.click = function(evt:Object) {
    var tempObj:Object = new Object();
    tempObj.quantity = evt.target._parent.quantity_ns.value;
    tempObj.id = thisProduct.id;
    tempObj.productObj = thisProduct;
    var theCart = evt.target._parent._parent.myCart;
    theCart.addProduct(tempObj.quantity, thisProduct);
};
addToCart_button.addEventListener("click", cartListener);
```

4. [シンタックスチェック] ボタン(スクリプトペインの上にある青色のチェックマーク)をクリックして、コードにシンタックスエラーがないことを確認します。

アプリケーションにコードを追加する際は、頻繁にシンタックスをチェックしてください。検出されたコード内のエラーはすべて、[出力] パネルに表示されます。シンタックスチェックを行う場合、チェックされるのは現在のスクリプトのみであり、FLA ファイル内に他のスクリプトがあってもチェックされません。詳細については、『Flash ユーザーガイド』の「スクリプトのデバッグ」を参照してください。

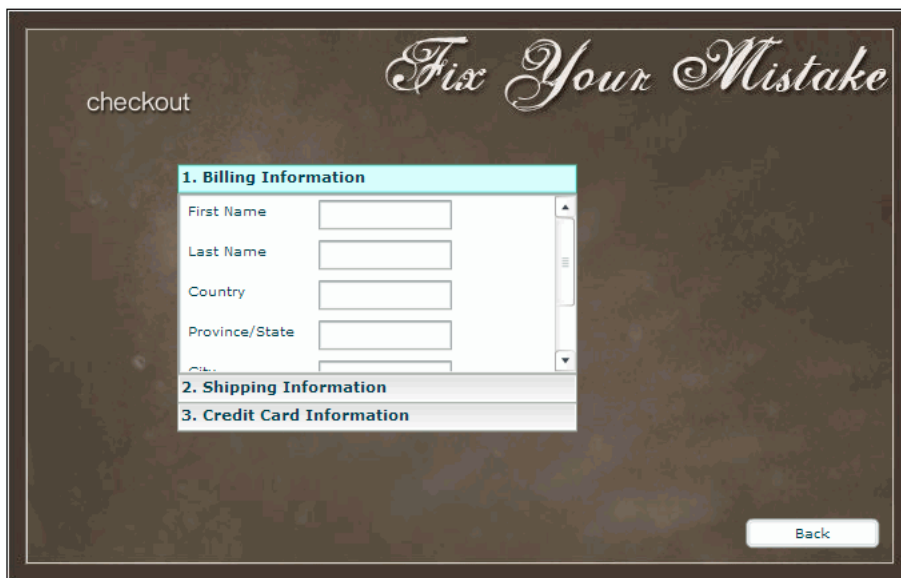
5. ドキュメントウィンドウの左上にある矢印ボタンをクリックするか [表示]-[ドキュメントの編集] を選択してシンボル編集モードを終了し、メインタイムラインに戻ります。

ペ
ア
ン
ル

Control+S を押して作業内容を保存してから、Control+Enter を押して (または [制御]-[ムービープレビュー] を選択して) アプリケーションをテストします。この時点では、ギフトをクリックして選択するとウィンドウが開き、ギフトのイメージ、説明および価格と、必要な数量を指定するための数値ステッパが表示されることを確認してください。

チェックアウトスクリーンの作成

ユーザーがメインスクリーンで [Checkout] ボタンをクリックすると、[Checkout] スクリーンが表示されます。[Checkout] スクリーンは、請求先、配送先、およびクレジットカード情報を入力できるフォームを備えています。チェックアウトスクリーンは次のように表示されます。

The screenshot shows a checkout interface with a dark brown background. At the top left, the word 'checkout' is written in a simple font. At the top right, the phrase 'Fix Your Mistake' is written in a large, elegant, white cursive script. In the center, there is a white rectangular form with a light blue header bar that reads '1. Billing Information'. Below the header, there are four input fields labeled 'First Name', 'Last Name', 'Country', and 'Province/State'. To the right of these fields is a vertical scrollbar. Below the first section, there are two more sections: '2. Shipping Information' and '3. Credit Card Information', each with a light blue header bar. In the bottom right corner of the form area, there is a small white button with the text 'Back' in blue.

チェックアウトインターフェイスは、アプリケーション内のフレーム 10 のキーフレームに配置されるコンポーネントで構成されます。このチェックアウトインターフェイスは、Accordion コンポーネントを使用して作成します。Accordion コンポーネントは、複数の子を一つずつ順に表示するナビゲーターです。また、Button コンポーネントインスタンスを追加して、ユーザーがメインスクリーンに戻るための [Back] ボタンを作成します。

後の手順では、Accordion インスタンス内の子として Billing Information ペイン、Shipping Information ペイン、および Credit Card Information ペインを表示するために使用するムービークリップを作成します。

1. アプリケーションのメインタイムラインで、再生ヘッドをフレーム 10 ([Checkout] ラベル) に移動します。Form レイヤーを選択します。
2. Form レイヤー内のフレーム 10 に空白のキーフレームを挿入します (フレームを選択し、[挿入]-[タイムライン]-[空白キーフレーム] を選択)。

3. 新規キーフレームを選択した状態で、Accordion コンポーネントのインスタンスを、[コンポーネント] パネルからステージにドラッグします。プロパティインスペクタで、インスタンス名として「checkout_acc」を入力します。幅を 300 ピクセルに、高さを 200 ピクセルに設定します。
4. Button コンポーネントのインスタンスを、[コンポーネント] パネルからステージの右下隅にドラッグします。プロパティインスペクタで、インスタンス名として「back_button」を入力します。[パラメータ] タブをクリックし、label プロパティとして「Back」を入力します。

請求先、配送先、およびクレジットカードの各ペインについて

Billing Information ペイン、Shipping Information ペイン、および Credit Card Information ペインは、Accordion コンポーネントインスタンス内に表示するムービークリップインスタンスを使って作成します。各ペインは、ネストしている 2 つのムービークリップで構成されます。

親ムービークリップには ScrollPane コンポーネントを設定します。これは、スクロール可能領域にコンテンツを表示するために使用します。子ムービークリップには Label コンポーネントと TextInput コンポーネントを設定します。これによって、ユーザーは名前や住所などの個人情報を入力できます。ScrollPane コンポーネントを使って子ムービークリップを表示することで、ユーザーは情報フィールドをスクロールできます。

Billing Information ペインの作成

まず、[Billing Information] フォームフィールドを表示する 2 つのムービークリップを作成します。それは、ScrollPane コンポーネントインスタンスが含まれる親ムービークリップと、Label コンポーネントインスタンスおよび TextArea コンポーネントインスタンスが含まれる子ムービークリップです。

1. [ライブラリ] パネル ([ウィンドウ]-[ライブラリ]) で、タイトルバーの右側にあるオプションメニューをクリックし、[新規シンボル] を選択します。
2. [新規シンボルの作成] ダイアログボックスで、[名前] に「checkout1_mc」を入力し、[タイプ] で [ムービークリップ] を選択します。
3. [詳細] ボタンをクリックします。[リンケージ] の [ActionScript に書き出し] を選択し、[最初のフレームに書き出し] を選択したままにして、[OK] をクリックします。
新規シンボルのドキュメントウィンドウが、シンボル編集モードで開きます。
4. ScrollPane コンポーネントのインスタンスをステージにドラッグします。
5. プロパティインスペクタで、インスタンス名として「checkout1_sp」を入力します。W の値を 300、H の値を 135 に設定します。x 座標に 0、y 座標に 0 を設定します。

6. [パラメータ] タブをクリックします。contentPath プロパティを **checkout1_sub_mc** に設定します。

checkout1_sub_mc ムービークリップがスクロールペイン内に表示され、ここに Label コンポーネントと TextInput コンポーネントが含まれることになります。次に、このムービークリップを作成します。

7. [ライブラリ] のオプションメニューから [新規シンボル] を選択します。
8. [新規シンボルの作成] ダイアログボックスで、[名前] に「**checkout1_sub_mc**」を入力し、[タイプ] で [ムービークリップ] を選択します。
9. [詳細] ボタンをクリックします。[リンケージ] の [ActionScript に書き出し] を選択し、[最初のフレームに書き出し] を選択したままにして、[OK] をクリックします。

新規シンボルのドキュメントウィンドウが、シンボル編集モードで開きます。

10. 6 個分の Label コンポーネントのインスタンスをステージにドラッグします。または、1 つのインスタンスをステージにドラッグしてから、そのインスタンスをステージ上で Ctrl キー (Windows) または Option キー (Macintosh) を押しながらクリックおよびドラッグしてコピーを作成することもできます。インスタンスの名前と位置は次のように設定します。

- 1 番目のインスタンスに対し、インスタンス名として「**firstname_lbl**」を入力し、x 座標と y 座標をそれぞれ **5** に設定します。[パラメータ] タブをクリックし、text パラメータとして「**First Name**」を入力します。
- 2 番目のインスタンスに対し、インスタンス名として「**lastname_lbl**」を入力し、x 座標を **5** に、y 座標を **35** に設定します。[パラメータ] タブをクリックし、text パラメータとして「**Last Name**」を入力します。
- 3 番目のインスタンスに対し、インスタンス名として「**country_lbl**」を入力し、x 座標を **5** に、y 座標を **65** に設定します。[パラメータ] タブをクリックし、text パラメータとして「**Country**」を入力します。
- 4 番目のインスタンスに対し、インスタンス名として「**province_lbl**」を入力し、x 座標を **5** に、y 座標を **95** に設定します。[パラメータ] タブをクリックし、text パラメータとして「**Province/State**」を入力します。
- 5 番目のインスタンスに対し、インスタンス名として「**city_lbl**」を入力し、x 座標を **5** に、y 座標を **125** に設定します。[パラメータ] タブをクリックし、text パラメータとして「**City**」を入力します。
- 6 番目のインスタンスに対し、インスタンス名として「**postal_lbl**」を入力し、x 座標を **5** に、y 座標を **155** に設定します。[パラメータ] タブをクリックし、text パラメータとして「**Postal/Zip Code**」を入力します。

11. 6 個分の TextInput コンポーネントのインスタンスをステージにドラッグします。各 TextInput インスタンスは、Label インスタンスのすぐ右側に配置します。たとえば、最初の TextInput インスタンスの x 座標は **105**、y 座標は **5** になります。次のとおり TextInput インスタンスに名前を付けます。

- 1 番目のインスタンス名は **billingFirstName_ti** です。
- 2 番目のインスタンス名は **billingLastName_ti** です。
- 3 番目のインスタンス名は **billingCountry_ti** です。
- 4 番目のインスタンス名は **billingProvince_ti** です。
- 5 番目のインスタンス名は **billingCity_ti** です。
- 6 番目のインスタンス名は **billingPostal_ti** です。

スクロールペイン内のコンテンツは、ペインの境界に近すぎると表示が切れることがあります。次に示す手順では、Label インスタンスと TextInput インスタンスが適切に表示されるように、白色の四角形を checkout1_sub_mc ムービークリップに追加します。

12. タイムラインで、[レイヤーの追加] ボタンをクリックします。作成した新規レイヤーを既存のレイヤーの下にドラッグします。コンポーネントの表示が影響を受けないように、この四角形のレイヤーは一番下にする必要があります。
13. 新規レイヤーのフレーム 1 を選択します。
14. ツールパネルで、矩形ツールを選択します。[線のカラー] を [なし] に設定し、[塗りのカラー] を白色に設定します。
- [ツール] パネルの [線のカラー] コントロールをクリックし、[なし] ボタン (白地に赤い斜線の付いた色見本) をクリックします。[塗りのカラー] コントロールをクリックし、白色の色見本をクリックします。
15. ドラッグして、Label インスタンスと TextInput インスタンスの下端と右端を越える四角形を作成します。

Shipping Information ペインの作成

Shipping Information ペインのムービークリップは、Billing Information ペインのムービークリップによく似ています。ただし、CheckBox コンポーネントを追加して、Billing Information ペインで入力したデータと同じデータを Shipping Information ペインのフォームフィールドに設定できるようにします。

1. 前述の説明 (39 ページの「Billing Information ペインの作成」を参照) に従って、Credit Card Information ペイン用のムービークリップを作成します。ただし、次のとおり異なる名前を指定します。
 - 1 番目のムービークリップについては、シンボル名として「checkout2_mc」を入力し、インスタンス名として「checkout2_sp」を入力します。プロパティインスペクタの [パラメータ] タブで、contentPath プロパティを checkout2_sub_mc に設定します。
 - 2 番目のムービークリップについては、シンボル名として「checkout2_sub_mc」を入力します。
 - TextInput インスタンスについては、各インスタンス名に含まれる "billing" を "shipping" に変更します。
2. checkout2_sub_mc ムービークリップをシンボル編集モードで開き、CheckBox コンポーネントのインスタンスをステージにドラッグして、これを先頭の Label インスタンスのすぐ上に配置します。

このインスタンスは、他のコンポーネントインスタンスと同様に必ずレイヤー 1 に配置します。
3. プロパティインスペクタで、インスタンス名として「sameAsBilling_ch」を入力します。
4. [パラメータ] タブをクリックします。label プロパティを Same As Billing Info に設定します。

Credit Card Information ペインの作成

Credit Card Information ペインのムービークリップも、Billing Information ペインおよび Shipping Information ペインのムービークリップに似ています。ただし、Credit Card Information ペインのネストしているムービークリップは、クレジットカード番号とその他のカードデータに対応しており、他の 2 つのペインのフィールドとは多少異なります。

1. Billing Information ペインを作成するための手順 1～9 (39 ページの「Billing Information ペインの作成」を参照) に従って、Credit Card Information ペインのムービークリップを作成します。ただし、次のとおり異なる名前を指定します。
 - 1 番目のムービークリップについては、シンボル名として「checkout3_mc」を入力し、インスタンス名として「checkout3_sp」を入力します。プロパティインスペクタの [パラメータ] タブで、contentPath プロパティを checkout3_sub_mc に設定します。
 - 2 番目のムービークリップについては、シンボル名として「checkout3_sub_mc」を入力します。

2. 4 個分の Label コンポーネントのインスタンスをステージにドラッグします。インスタンスの名前と位置は次のように設定します。
 - 1 番目のインスタンスに対し、インスタンス名として「**ccName_lbl**」を入力し、x 座標と y 座標を **5** に設定します。[パラメータ] タブをクリックし、text パラメータとして「**Name On Card**」を入力します。
 - 2 番目のインスタンスに対し、インスタンス名として「**ccType_lbl**」を入力し、x 座標を **5** に、y 座標を **35** に設定します。[パラメータ] タブをクリックし、text パラメータとして「**Card Type**」を入力します。
 - 3 番目のインスタンスに対し、インスタンス名として「**ccNumber_lbl**」を入力し、x 座標を **5** に、y 座標を **65** に設定します。[パラメータ] タブをクリックし、text パラメータとして「**Card Number**」を入力します。
 - 4 番目のインスタンスに対し、インスタンス名として「**ccExp_lbl**」を入力し、x 座標を **5** に、y 座標を **95** に設定します。[パラメータ] タブをクリックし、text パラメータとして「**Expiration**」を入力します。
3. TextInput コンポーネントのインスタンスをステージにドラッグし、ccName_lbl インスタンスの右側に配置します。新規インスタンスに **ccName_ti** という名前を指定します。x 座標を **105**、y 座標を **5** に設定します。幅を **140** に設定します。
4. ComboBox コンポーネントのインスタンスをステージにドラッグし、ccType_lbl インスタンスの右側に配置します。新規インスタンスに **ccType_cb** という名前を指定します。x 座標を **105**、y 座標を **35** に設定します。幅を **140** に設定します。
5. TextInput コンポーネントのインスタンスをもう 1 つステージにドラッグし、ccNumber_lbl インスタンスの右側に配置します。新規インスタンスに **ccNumber_ti** という名前を指定します。x 座標を **105**、y 座標を **65** に設定します。幅を **140** に設定します。
6. 2 個分の ComboBox コンポーネントのインスタンスをステージにドラッグします。1 つは ccExp_lbl インスタンスの右側に配置し、もう 1 つはその右側に配置します。1 番目の新規インスタンスに **ccMonth_cb** という名前を指定します。幅を **60** に、x 座標を **105** に、y 座標を **95** にそれぞれ設定します。2 番目の新規インスタンスに **ccYear_cb** という名前を指定します。幅を **70** に、x 座標を **175** に、y 座標を **95** にそれぞれ設定します。
7. Button コンポーネントのインスタンスをステージにドラッグし、フォーム下部の ccMonth_cb インスタンスの下側に配置します。新規インスタンスに **checkout_button** という名前を指定します。x 座標を **125**、y 座標を **135** に設定します。プロパティインスペクタの [パラメータ] タブで、label プロパティを **Checkout** に設定します。
8. Billing Information ペインを作成するための手順 14 ~ 15 ([39 ページの「Billing Information ペインの作成」](#)を参照) に従って、フォームの下部に四角形を追加します。

[Checkout] ボタンへのイベントリスナーの追加

ユーザーが [Checkout] ボタンをクリックしたら [Checkout] スクリーンを表示するコードを追加します。

- [アクション] パネルで、タイムラインのフレーム 1 に既に追加した `ActionScript` の後に、次のコードを追加します。

```
// [Checkout] ボタンをクリックされたら "checkout" フレームラベルに移動する。
var checkoutBtnListener:Object = new Object();
checkoutBtnListener.click = function(evt:Object) {
    evt.target._parent.gotoAndStop("checkout");
};
checkout_button.addEventListener("click", checkoutBtnListener);
```

このコードでは、ユーザーが [Checkout] ボタンをクリックすると、再生ヘッドがタイムライン内の [Checkout] ラベルに移動するよう指定しています。

[Checkout] スクリーンへのコードの追加

メインタイムラインのフレーム 10 で、アプリケーションの [Checkout] スクリーンにコードを追加します。このコードは、`Accordion` コンポーネントとその他のコンポーネントを使って作成した `Billing Information` ペイン、`Shipping Information` ペイン、および `Credit Card Information` ペインにユーザーが入力したデータを処理するものです。

1. タイムラインで `Actions` レイヤーのフレーム 10 を選択し、空白のキーフレームを挿入します ([挿入]-[タイムライン]-[空白キーフレーム] を選択)。
2. [アクション] パネルを開きます (F9)。
3. [アクション] パネルに次のコードを追加します。

```
stop();
import mx.containers.*;

// ステージ上の Accordion コンポーネントを定義する。
var checkout_acc:Accordion;
```

4. 最初の子を `Accordion` コンポーネントに追加します。これは、ユーザーから請求先情報の入力を受け付けるために使用します。次のコードを追加します。

```
// Accordion コンポーネントの子を定義する。
var child1 = checkout_acc.createChild("checkout1_mc", "child1_mc", {label:"1.
    Billing Information"});
var thisChild1 = child1.checkout1_sp.spContentHolder;
```

先頭行では `Accordion` コンポーネントの `createChild()` メソッドを呼び出して、以前に作成した `checkout1_mc` ムービークリップシンボルのインスタンスを作成します。インスタンス名は `child1_mc` で、ラベルは `"1. Billing Information"` です。2 番目のコード行では、埋め込まれている `ScrollPane` コンポーネントインスタンスへのショートカットを作成します。

5. Accordion インスタンスの 2 番目の子を作成し、配送先情報を受け取ります。

```
/* 2 番目の子 Accordion に追加する。
sameAsBilling_ch CheckBox のイベントリスナーを追加する。
最初の子のフォーム値が 2 番目の子にコピーされる。*/
var child2 = checkout_acc.createChild("checkout2_mc", "child2_mc", {label:"2.
    Shipping Information"});
var thisChild2 = child2.checkout2_sp.spContentHolder;
var checkboxListener:Object = new Object();
checkboxListener.click = function(evt:Object) {
    if (evt.target.selected) {
        thisChild2.shippingFirstName_ti.text =
            thisChild1.billingFirstName_ti.text;
        thisChild2.shippingLastName_ti.text =
            thisChild1.billingLastName_ti.text;
        thisChild2.shippingCountry_ti.text = thisChild1.billingCountry_ti.text;
        thisChild2.shippingProvince_ti.text =
            thisChild1.billingProvince_ti.text;
        thisChild2.shippingCity_ti.text = thisChild1.billingCity_ti.text;
        thisChild2.shippingPostal_ti.text = thisChild1.billingPostal_ti.text;
    }
};
thisChild2.sameAsBilling_ch.addEventListener("click", checkboxListener);
```

コードの先頭の 2 行は、請求先情報用の子を作成するコードに似ています。まず、checkout2_mc ムービークリップシンボルのインスタンスを作成し、インスタンス名には child2_mc、ラベルには “2. Shipping Information” を指定します。2 番目のコード行では、埋め込まれている ScrollPane コンポーネントインスタンスへのショートカットを作成します。

コードの 3 行目からは、CheckBox インスタンスにイベントリスナーを追加します。ユーザーがチェックボックスをクリックすると、ユーザーが Billing Information ペインに入力したデータが配送先情報で使用されます。

6. Accordion インスタンスの 3 番目の子を作成します。これは、クレジットカード情報用です。

```
// Accordion の 3 番目の子を定義する。
var child3 = checkout_acc.createChild("checkout3_mc", "child3_mc", {label:"3.
    Credit Card Information"});
var thisChild3 = child3.checkout3_sp.spContentHolder;
```

7. 次のコードを追加して、クレジットカードの月、年、および種類の ComboBox インスタンスを作成し、それぞれを静的に定義された配列に設定します。

```
/* 3 つの ComboBox インスタンス ccMonth_cb、ccYear_cb、および ccType_cb の値をス
    テージに設定する。
ccMonth_cb、ccYear_cb and ccType_cb */
thisChild3.ccMonth_cb.labels = ["01", "02", "03", "04", "05", "06", "07",
    "08", "09", "10", "11", "12"];
thisChild3.ccYear_cb.labels = [2004, 2005, 2006, 2007, 2008, 2009, 2010];
thisChild3.ccType_cb.labels = ["VISA", "MasterCard", "American Express",
    "Diners Club"];
```

8. 最後に、次のコードを追加し、[Checkout] ボタンと [Back] ボタンにイベントリスナーを追加します。ユーザーが [Checkout] ボタンをクリックすると、リスナーオブジェクトによって、Billing Information ペイン、Shipping Information ペイン、および Credit Card Information ペインのフォームフィールドが、サーバーに送られる LoadVars オブジェクトにコピーされます。この LoadVars クラスによって、オブジェクト内のすべての変数が指定の URL に送られます。ユーザーが [Back] ボタンをクリックすると、アプリケーションのメインスクリーンに戻ります。

```
/* checkout_button Button インスタンスのリスナーを作成する。
このリスナーは、ユーザーが [Checkout] ボタンをクリックすると、すべてのフォーム変数をサー
バーに送信する。*/
var checkoutListener:Object = new Object();
checkoutListener.click = function(evt:Object){
    evt.target.enabled = false;
    /* 変数をリモートサーバーに送信および受信する
    2 つの LoadVars オブジェクトを作成する。*/
    var response_lv:LoadVars = new LoadVars();
    var checkout_lv:LoadVars = new LoadVars();
    checkout_lv.billingFirstName = thisChild1.billingFirstName_ti.text;
    checkout_lv.billingLastName = thisChild1.billingLastName_ti.text;
    checkout_lv.billingCountry = thisChild1.billingCountry_ti.text;
    checkout_lv.billingProvince = thisChild1.billingProvince_ti.text;
    checkout_lv.billingCity = thisChild1.billingCity_ti.text;
    checkout_lv.billingPostal = thisChild1.billingPostal_ti.text;
    checkout_lv.shippingFirstName = thisChild2.shippingFirstName_ti.text;
    checkout_lv.shippingLastName = thisChild2.shippingLastName_ti.text;
    checkout_lv.shippingCountry = thisChild2.shippingCountry_ti.text;
    checkout_lv.shippingProvince = thisChild2.shippingProvince_ti.text;
    checkout_lv.shippingCity = thisChild2.shippingCity_ti.text;
    checkout_lv.shippingPostal = thisChild2.shippingPostal_ti.text;
    checkout_lv.ccName = thisChild3.ccName_ti.text;
    checkout_lv.ccType = thisChild3.ccType_cb.selectedItem;
    checkout_lv.ccNumber = thisChild3.ccNumber_ti.text;
    checkout_lv.ccMonth = thisChild3.ccMonth_cb.selectedItem;
    checkout_lv.ccYear = thisChild3.ccYear_cb.selectedItem;

    /* checkout_lv LoadVars からサーバー上のリモートスクリプトに変数を送信する。
    response_lv インスタンス内の結果を保存する。*/
    checkout_lv.sendAndLoad("http://www.flash-mx.com/mm/firstapp/cart.cfm",
    response_lv, "POST");
    response_lv.onLoad = function(success:Boolean) {
        evt.target.enabled = true;
    };
};
thisChild3.checkout_button.addEventListener("click", checkoutListener);
cart_mc._visible = false;
var backListener:Object = new Object();
backListener.click = function(evt:Object) {
    evt.target._parent.gotoAndStop("home");
}
back_button.addEventListener("click", backListener);
```

アプリケーションのテスト

おめでとうございます。ここまでで、アプリケーションの作成が完了しました。Control+S を押して作業内容を保存してから、Control+Enter を押して (または [制御]-[ムービープレビュー] を選択して) アプリケーションをテストします。

完成済みアプリケーションの参照

チュートリアルをうまく完成できなかった場合は、動作可能な完成済みアプリケーションを参照できます。サンプルファイルについては、Web サイトの Flash サンプルページ (www.adobe.com/go/learn_fl_samples_jp) を参照してください。次のサンプルファイルがあります。

- 最初の Flash (FLA) ファイル "first_app_start fla"
- 最終的なファイル "first_app fla"

アプリケーションの FLA ファイルを参照するには、"components_application" フォルダにある "first_app fla" ファイルを開きます。

自分で作成したファイルをこれらのファイルと比較すると、ミスを発見するのに役立ちます。

このアプリケーションを作成する際に使用するすべてのコンポーネントはライブラリにあります (また、このアプリケーションで使用するグラフィックファイルやその他のアセットもライブラリに含まれています)。いくつかのコンポーネントはインスタンスとしてステージに表示されますが、ActionScript コードで参照するコンポーネントは実行時まで表示されません。

コンポーネントの利用

この章では、各種の Adobe Flash (FLA) ファイルおよび ActionScript クラスファイルを使用して、コンポーネントをドキュメントに追加し、そのプロパティを設定する方法について説明します。また、コードヒントの使用、カスタムフォーカスナビゲーションの作成、コンポーネントの深さの管理、Adobe Component Architecture バージョン1 コンポーネントからバージョン2 へのアップグレードなど、いくつかの高度なトピックについても説明します。

この章で使用するサンプルファイルについては、Web サイトの Flash サンプルページ (www.adobe.com/go/learn_fl_samples_jp) を参照してください。次のサンプルファイルがあります。

- TipCalculator.flc
- TipCalculator.swf

この章では、次のトピックについて説明します。

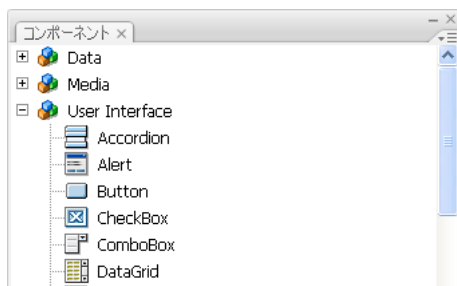
[コンポーネント] パネル	50
Flash ドキュメントへのコンポーネントの追加	50
[ライブラリ] パネル内のコンポーネント	54
コンポーネントパラメータの設定	54
コンポーネントのサイズ変更	56
Flash ドキュメントからのコンポーネントの削除	57
コードヒントの使用	58
スクリーンでの ActionScript の使用	58
カスタムフォーカスナビゲーションの作成	61
ドキュメント内のコンポーネントの深度を管理する	62
ライブプレビュー内のコンポーネント	63
コンポーネントでのプリローダーの使用	64
コンポーネントのロードについて	65
バージョン1 コンポーネントからバージョン2 アーキテクチャへのアップグレード	66

[コンポーネント] パネル

[コンポーネント] パネルには、ユーザーレベル設定ディレクトリおよび "Components" ディレクトリ内のすべてのコンポーネントが表示されます。このディレクトリの詳細については、[14 ページの「コンポーネントファイルの格納場所」](#)を参照してください。

[コンポーネント] パネルを表示するには：

- [ウィンドウ]-[コンポーネント] を選択します。



[コンポーネント] パネルメニュー

Flash の起動後にインストールされたコンポーネントを表示するには：

1. ActionScript 2.0 を使用してパブリッシュするようパブリッシュ設定を行います。
[ファイル]-[パブリッシュ設定]-[Flash] タブを選択し、[ActionScript] ドロップダウンメニューから [ActionScript 2.0] を選択します。
2. [ウィンドウ]-[コンポーネント] を選択します。
3. [コンポーネント] パネルポップアップメニューから [リロード] を選択します。

Flash ドキュメントへのコンポーネントの追加

[コンポーネント] パネルからステージにコンポーネントをドラッグすると、コンパイルされたクリップ (SWC) のシンボルが [ライブラリ] パネルに追加されます。SWC シンボルがライブラリに追加された後、複数のインスタンスをステージにドラッグできます。また、`UIObject.createClassObject()` ActionScript メソッドを使用して、そのコンポーネントを実行時にドキュメントに追加することもできます。詳細については、『ActionScript 2.0 コンポーネントリファレンスガイド』を参照してください。

×
中

Menu コンポーネントと Alert コンポーネントは例外です。これらのコンポーネントは `UIObject.createClassObject()` を使ってインスタンス化することはできません。代わりに `show()` メソッドを使用してください。

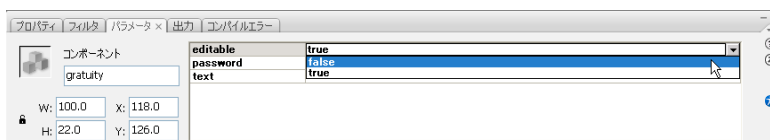
オーサリング時にコンポーネントを追加する

[コンポーネント] パネルでコンポーネントをドキュメントに追加した後、[ライブラリ] パネルからコンポーネントをステージにドラッグしてコンポーネントのインスタンスをドキュメントに追加することができます。追加したインスタンスのプロパティは、プロパティインスペクタの [パラメータ] タブ、または [コンポーネントインスペクタ] の [パラメータ] タブで設定できます。

[コンポーネント] パネルでコンポーネントを Flash ドキュメントに追加するには：

1. ActionScript 2.0 を使用してパブリッシュするようパブリッシュ設定を行います。
[ファイル]-[パブリッシュ設定]-[Flash] タブを選択し、[ActionScript] ドロップダウンメニューから [ActionScript 2.0] を選択します。
2. [ウィンドウ]-[コンポーネント] を選択します。
3. 次のいずれかの操作を行います。
 - コンポーネントを [コンポーネント] パネルからステージまでドラッグします。
 - [コンポーネント] パネル内のコンポーネントをダブルクリックします。
4. コンポーネントが FLA ファイル (インストールされたすべてのバージョン 2 コンポーネントが SWC ファイル) であり、同時に、同じコンポーネント内の他のインスタンスのスキン、または追加しようとしているコンポーネントとスキンを共有しているコンポーネントのスキンが編集されている場合は、次のいずれかの操作を行います。
 - [既存のアイテムを置き換えない] を選択して編集済みのスキンを保存し、編集済みのスキンを新規ドキュメントに適用します。
 - [既存のアイテムを置き換える] を選択して、すべてのスキンをデフォルトスキンに置き換えます。新規コンポーネントおよびすべての旧バージョンのコンポーネント、またはそのスキンを共有するすべての旧バージョンのコンポーネントで、デフォルトスキンが使用されます。
5. ステージでコンポーネントを選択します。
6. [ウィンドウ]-[プロパティ]-[プロパティ] を選択します。
7. プロパティインスペクタで、コンポーネントインスタンスのインスタンス名を入力します。
8. [パラメータ] タブをクリックして、インスタンスのパラメータを指定します。

次の図は、"TipCalculator.fl" サンプルファイルに含まれている TextInput コンポーネントのプロパティインスペクタを示しています。"TipCalculator.fl" サンプルファイルへのアクセス方法については、Web サイトの Flash サンプルページ (www.adobe.com/go/learn_fl_samples_jp) を参照してください。



詳細については、54 ページの「コンポーネントパラメータの設定」を参照してください。

9. 幅と高さの値を編集することにより、必要に応じてコンポーネントのサイズを変更します。

特定のコンポーネントタイプのサイズ変更の詳細については、『[Action Script 2.0 コンポーネントリファレンスガイド](#)』の各コンポーネントのセクションを参照してください。

10. コンポーネントの色およびテキストのフォーマットを変更するには、必要に応じて次の操作を行います。

- すべてのコンポーネントで使用できる `setStyle()` メソッドを使用して、コンポーネントインスタンスの特定のプロパティ値を設定または変更します。詳細については、『[Action Script 2.0 コンポーネントリファレンスガイド](#)』の『[UIObject.setStyle\(\)](#)』を参照してください。
- すべてのバージョン 2 コンポーネントに割り当てられているグローバルスタイル宣言で、複数のプロパティを編集します。
- 特定のコンポーネントインスタンスで使用するカスタムスタイル宣言を作成します。
詳細については、[82 ページ](#)の『[スタイルによるコンポーネントのカラーとテキストの変更](#)』を参照してください。

11. コンポーネントの外観をカスタマイズするには、次のいずれかの操作を行います。

- テーマを適用します ([108 ページ](#)の『[テーマについて](#)』を参照してください)。
- コンポーネントのスキンを編集します ([96 ページ](#)の『[コンポーネントへのスキンの適用について](#)』を参照してください)。

ActionScript を使用して実行時にコンポーネントを追加する

このセクションの説明は、ActionScript についてある程度の知識を持っている方向けです。

コンポーネントを Flash アプリケーションに動的に追加するには、`createClassObject()` メソッドを使用します。ほとんどのコンポーネントは、このメソッドを `UIObject` クラスから継承しています。たとえば、ユーザー設定の環境に基づいたページレイアウト (Web ポータルのホームページなど) を作成するコンポーネントを追加できます。

Flash と共にインストールされるバージョン 2 コンポーネントは、パッケージディレクトリにあります。詳細については、『[ActionScript 2.0 の学習](#)』の『[パッケージについて](#)』を参照してください。オーサリング時にコンポーネントをステージに追加する場合は、そのインスタンス名 (`myButton` など) だけでコンポーネントを参照できます。しかし、実行時に ActionScript を使用してコンポーネントをアプリケーションに追加する場合は、クラスの完全修飾名 (`mx.controls.Button` など) を指定するか、または `import` ステートメントを使用してパッケージを読み込む必要があります。

たとえば、`Alert` コンポーネントを参照する ActionScript コードを記述するには、クラスを参照する `import` ステートメントを使用します。

```
import mx.controls.Alert;
Alert.show("The connection has failed", "Error");
```

または、パッケージの完全パスも使用できます。

```
mx.controls.Alert.show("The connection has failed", "Error");
```

詳細については、『[ActionScript 2.0 の学習](#)』の「[クラスファイルの読み込みについて](#)」を参照してください。

動的に追加されたコンポーネントのパラメータは、ActionScript のメソッドを使って設定することができます。詳細については、『[ActionScript 2.0 コンポーネントリファレンスガイド](#)』を参照してください。

×
❏

実行時にコンポーネントをドキュメントに追加するには、SWF ファイルのコンパイル時に、コンポーネントがライブラリにある必要があります。コンポーネントをライブラリに追加するには、[コンポーネント] パネルからライブラリにコンポーネントアイコンをドラッグします。また、ActionScript を使って動的にインスタンス化されるコンポーネントを含んだムービークリップを別のムービークリップ内に読み込むには、そのコンポーネントが、SWF ファイルのコンパイル時にメインのムービークリップのライブラリ内にある必要があります。

ActionScript を使用してコンポーネントを Flash ドキュメントに追加するには：

1. コンポーネントを、[コンポーネント] パネルからカレントドキュメントのライブラリまでドラッグします。

×
❏

コンポーネントは、デフォルトで [最初のフレームに書き出し] に設定されます (Windows の場合は右クリック、Macintosh の場合は Control キーを押しながらクリックし、[リンケージ] メニューオプションを選択して、[最初のフレームに書き出し] 設定を表示します)。コンポーネントを含むアプリケーションに対してプリローダーを使用する場合、書き出しフレームを変更する必要があります。手順については、[64 ページの「コンポーネントでのプリローダーの使用」](#)を参照してください。

2. コンポーネントを追加するタイムライン内のフレームを選択します。
3. [アクション] パネルが開いていなければ、開きます。
4. コンポーネントのインスタンスを実行時に作成するには、createClassObject() を呼び出します。このメソッドは単独で呼び出すこともできますが、任意のコンポーネントのインスタンスから呼び出すこともできます。createClassObject() メソッドのパラメータでは、コンポーネントのクラス名、新規インスタンスのインスタンス名、深さ、および実行時にプロパティを設定するときに使用するオプションの初期化オブジェクトを指定できます。

次の例に示すように、クラス名パラメータにクラスパッケージを指定できます。

```
createClassObject(mx.controls.CheckBox, "cb", 5, {label:"Check Me"});
```

また、クラスパッケージは次のようにして読み込むこともできます。

```
import mx.controls.CheckBox;  
createClassObject(CheckBox, "cb", 5, {label:"Check Me"});
```

詳細については、[67 ページ、第 4 章の「コンポーネントイベントの処理」](#)、および『[ActionScript 2.0 コンポーネントリファレンスガイド](#)』の「[UIObject.createClassObject\(\)](#)」を参照してください。

[ライブラリ] パネル内のコンポーネント

コンポーネントをドキュメントに追加すると、コンパイルされたクリップ (SWC ファイル) のシンボルとして [ライブラリ] パネルに表示されます。



[ライブラリ] パネル内の ComboBox コンポーネント

ライブラリ内のコンポーネントアイコンをステージにドラッグすると、コンポーネントのインスタンスをさらに追加できます。

コンパイルされたクリップの詳細については、[19 ページの「コンパイルされたクリップと SWC ファイルについて」](#)を参照してください。

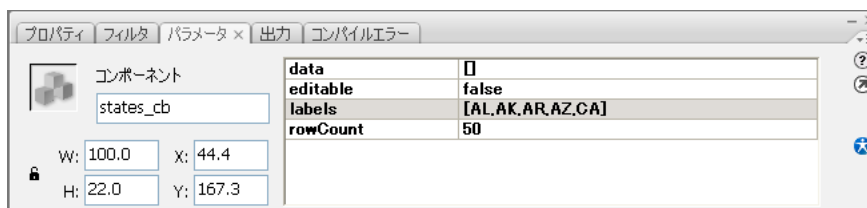
コンポーネントパラメータの設定

各コンポーネントには、その外観やビヘイビアを変更するためのパラメータがあります。パラメータは、プロパティインスペクタと [コンポーネントインスペクタ] に表示されるプロパティです。最もよく使用されるプロパティはオーサリングパラメータとして表示されます。その他のものは ActionScript を使用して設定する必要があります。オーサリング時に設定できるすべてのパラメータは、ActionScript を使用して設定することもできます。ActionScript で設定したパラメータは、オーサリング時に設定した値よりも優先されます。

バージョン 2 のユーザーインターフェイス (UI) コンポーネントはすべて、UIObject クラスおよび UIComponent クラスからプロパティとメソッドを継承します。すべてのコンポーネントが使用するプロパティおよびメソッドとしては、「UIObject.setSize()」、「UIObject.setStyle()」、「UIObject.x」、「UIObject.y」などがあります。各コンポーネントには独自のプロパティやメソッドもあり、その一部はオーサリングパラメータとして使用できます。たとえば、ProgressBar コンポーネントには percentComplete プロパティ (「ProgressBar.percentComplete」) があり、NumericStepper コンポーネントには nextValue および previousValue プロパティ (「NumericStepper.nextValue」、「NumericStepper.previousValue」) があります。コンポーネントインスタンスのパラメータは、[コンポーネントインスペクタ] またはプロパティインスペクタを使用して設定できます。どちらのパネルを使用しても結果は同じです。

プロパティインスペクタでコンポーネントのインスタンス名を入力するには：

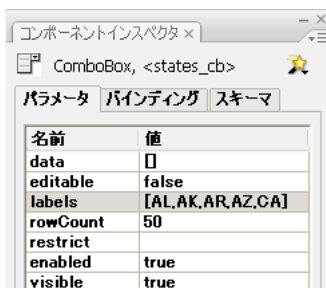
1. [ウィンドウ]-[プロパティ]-[プロパティ] を選択します。
2. ステージでコンポーネントのインスタンスを選択します。
3. [コンポーネント] という文字列の下にあるテキストフィールドにインスタンス名を入力します。
このとき、コンポーネントの種類を表す接尾辞をインスタンス名に付加するとよいでしょう。ActionScript コードが理解しやすくなります。この例の場合、コンポーネントは米国の州をリストするコンボボックスであるため、インスタンス名は **states_cb** です。



[コンポーネントインスペクタ] でコンポーネントのインスタンスのパラメータを入力するには：

1. [ウィンドウ]-[コンポーネントインスペクタ] を選択します。
2. ステージでコンポーネントのインスタンスを選択します。

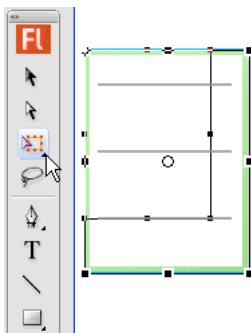
3. パラメータを入力するには、[パラメータ]タブをクリックします。



4. コンポーネントのバインディングやスキーマを入力または表示するには、該当のタブをクリックします。

コンポーネントのサイズ変更

コンポーネントインスタンスのサイズを変更するには、自由変形ツール、または `setSize()` メソッドを使用します。



自由変形ツールによるステージ上での Menu コンポーネントのサイズ変更

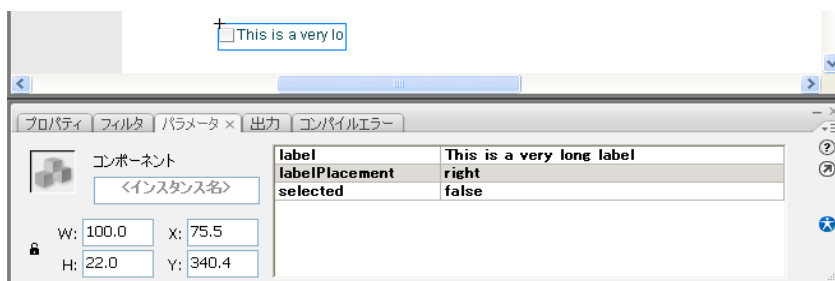
setSize() メソッドは、すべてのコンポーネントインスタンスから呼び出して、コンポーネントのサイズを変更することができます (『ActionScript 2.0 コンポーネントリファレンスガイド』の「UIObject.setSize()」を参照してください)。次のコードでは、TextArea コンポーネントのサイズが、幅が 200 ピクセル、高さが 300 ピクセルに変更されます。

```
myTextArea.setSize(200, 300);
```



ActionScript のプロパティ `_width` および `_height` を使用して、コンポーネントの幅と高さを調整する場合、コンポーネントのサイズは変更されますが、コンポーネント内のコンテンツのレイアウトは同じままです。そのため、ムービー再生の際にコンポーネントに歪みを生じることがあります。

コンポーネントのラベルは、適切に表示されるようにそのサイズが自動的に調整されることはありません。ドキュメントに追加されたコンポーネントインスタンスが小さすぎてラベルを表示できない場合は、ラベルのテキストの一部が省略されます。ラベルが適切に表示されるように、コンポーネントのサイズを調整する必要があります。



端が切り取られた CheckBox コンポーネントのラベル

コンポーネントのサイズ変更の詳細については、『ActionScript 2.0 コンポーネントリファレンスガイド』の各コンポーネントの説明を参照してください。

Flash ドキュメントからのコンポーネントの削除

Flash ドキュメントからコンポーネントのインスタンスを削除するには、コンパイルされたクリップのアイコンをライブラリから削除して、コンポーネントを削除する必要があります。ステージからコンポーネントを削除するだけでは十分ではありません。

コンポーネントをドキュメントから削除するには：

1. [ライブラリ] パネルで、コンパイルされたクリップ (SWC) のシンボルを選択します。
2. [ライブラリ] パネルの下部にある [削除] ボタンをクリックするか、[ライブラリ] のオプションメニューから [削除] を選択します。
3. [削除] ダイアログボックスで、[削除] をクリックして削除を確認します。

コードヒントの使用

ActionScript 2.0 を使用している場合、コンポーネントクラスを含むビルトインクラスに基づいた変数を厳密に型指定することができます。厳密な型指定を行うことで、ActionScript エディタでその変数のコードヒントが表示されます。たとえば、次のように入力したとします。

```
import mx.controls.CheckBox;  
var myCheckBox:CheckBox;  
myCheckBox.
```

変数が CheckBox 型として指定されているため、myCheckBox の後にピリオドを入力するとすぐに、CheckBox コンポーネントで利用できるメソッドとプロパティのリストが表示されます。詳細については、『ActionScript 2.0 の学習』の「データ型の割り当てと厳密な型指定について」、および『Flash ユーザーガイド』の「コードヒントを使用するには」を参照してください。

スクリーンでの ActionScript の使用

スクリーン、スライド、およびフォームは、Flash CS3 では使用できません。ただし、Flash 8 で作成した既存のプロジェクトファイルを開いた場合は使用できますが、ActionScript 2.0 および ActionScript 3.0 のいずれの場合も、Flash CS3 ではスクリーン、スライド、およびフォームの各アプリケーションを新規に作成できません。

ドキュメント内のスクリーンを制御するには、ActionScript を使用して、スクリーンの挿入、削除、名前の変更、順序の変更など、さまざまな操作を実行します。

ActionScript では、スクリーンを制御する際にスクリーンのインスタンス名、クラス名、および基準点を使用します。詳細については、[59 ページの「スクリーンのインスタンス名、クラス名、基準点」](#)を参照してください。また、ActionScript ではスクリーンパラメータも使用します。詳細については、『Flash ユーザーガイド』の「スクリーンのパラメータの設定」を参照してください。

×
中


スクリーンとムービークリップは、同様の方法で ActionScript とやり取りしますが、大きく異なる点がいくつかあります。詳細については、[59 ページの「スクリーンと ActionScript のやり取りの例」](#)を参照してください。

詳細については、『ActionScript 2.0 コンポーネントリファレンスガイド』の「Screen クラス」、「Form クラス」、および「Slide クラス」を参照してください。

スクリーンのインスタンス名、クラス名、基準点

スクリーン名から、そのスクリーンのインスタンス名とクラス名が自動生成されます。これらの識別ラベルは、ActionScript を使用してさまざまな方法でスクリーンを操作する際に必要です。スクリーンの動作を調整するには、スクリーンの基準点を変更します。これらの機能は、次に示す方法で操作できます。

- インスタンス名はスクリーンに割り当てられた一意の名前で、ActionScript でスクリーンを特定する際に使用します。プロパティインスペクタで、インスタンス名を変更します。インスタンス名は、スクリーンアウトラインペインのスクリーン名およびスクリーンのリンケージ識別子と同じものになります。インスタンス名を更新すると、スクリーン名とリンケージ識別子も更新されます。詳細については、『Flash ユーザーガイド』の「スクリーンのプロパティおよびパラメータの設定」を参照してください。

	ムービークリップ、ボタン、グラフィックなどのシンボルインスタンスにもインスタンス名があります。詳細については、『Flash ユーザーガイド』の「シンボル、インスタンス、およびライブラリアセットの使用」を参照してください。
---	--

- クラス名は、スクリーンの割り当て先となる ActionScript クラスを識別します。デフォルトでは、スライドスクリーンは `mx.screens.Slide` クラスに割り当てられ、フォームスクリーンは `mx.screens.Form` クラスに割り当てられます。スクリーンで使用できるメソッドやプロパティを変更するには、スクリーンを別のクラスに割り当てます。ActionScript クラスの詳細については、『ActionScript 2.0 の学習』の「クラス」を参照してください。
- プロパティインスペクタでは、x 座標と y 座標のフィールドおよび基準点グリッドに基準点が表示されます。詳細については、『Flash ユーザーガイド』の「スクリーンのプロパティおよびパラメータの設定」を参照してください。スクリーンコンテンツの操作において制御性を高めるために、基準点を移動したい場合があります。たとえば、スクリーンの中央にらせん形を作成する場合、スクリーンの基準点を中央に移動し、その基準点を中心にスクリーンを回転させることができます。

スクリーンと ActionScript のやり取りの例

スクリーンは、ActionScript とやり取りするという点で、ネストされたムービークリップに似ています。詳細については、『Flash ユーザーガイド』の「ネストされたムービークリップについて」を参照してください。ただし、いくつかの違いがあります。

スクリーンでの ActionScript については、次のガイドラインを参照してください。

- [スクリーンアウトライン]ペインでスクリーンを選択して ActionScript を追加すると、ActionScript がムービークリップに直接追加されるのと同様に、スクリプトがオブジェクトのアクションとして直接スクリーンに追加されます。スクリーン間のナビゲーションを作成するなどの単純なコードにはオブジェクトのアクションを使用し、より複雑なコードには外部 ActionScript ファイルを使用します。

- ドキュメント構造とスクリーン名は、ActionScript を追加する前に確定しておくことをお勧めします。スクリーン名を変更すると自動的にインスタンス名も変化するので、記述する ActionScript に含まれるすべてのインスタンス名を変更する必要があります。
- スクリーンのタイムラインにフレームアクションを追加するには、スクリーンを選択し、デフォルトで縮小されているタイムラインを展開して、タイムラインの最初のフレームを選択します。スクリーンで複雑なコードが使用されている場合は、フレームのアクションではなく外部 ActionScript ファイルを使用します。
- スクリーンベースのドキュメントのメインタイムラインを表示または操作することはできません。ただし、ターゲットパスの `_root` を使用してメインタイムラインをターゲットにすることは可能です。
- 各スクリーンは、クラスに基づいて自動的に ActionScript と関連付けられます。詳細については、『Flash ユーザーガイド』の「スライドスクリーンとフォームスクリーン」を参照してください。スクリーンとクラスの関連付けは変更可能です。また、プロパティインスペクタでスクリーンのいくつかのパラメータを設定することもできます。詳細については、『Flash ユーザーガイド』の「スクリーンのプロパティおよびパラメータの設定」を参照してください。
- ActionScript を使用してスクリーンを制御するには、Screen クラス、Slide クラス、Form クラスを使用します。
- インタラクティブ機能を作成するには、可能な限りコンポーネントを使用します。単一の FLA ファイルに配置できるコンポーネントインスタンスの合計は、最大で 125 個です。
- スライド間のナビゲーションを作成するには、`rootSlide` を使用します。たとえば、現在のスライドを取得するには、`rootSlide.currentSlide` を使用します。
- スライドナビゲーションを `on(reveal)` ハンドラまたは `on(hide)` ハンドラ内で実行しないでください。
- スクリーンを制御する ActionScript コードには、`on(keydown)` イベントや `on(keyup)` イベントを追加しないでください。

ActionScript を使用したスクリーン制御の詳細については、『ActionScript 2.0 コンポーネントリファレンスガイド』の「Screen クラス」、「Slide クラス」、および「Form クラス」を参照してください。Object クラスと `onClipEvent()` イベントハンドラの詳細については、『ActionScript 2.0 リファレンスガイド』の「オブジェクト」および「onClipEvent ハンドラ」を参照してください。

スクリーンでのコンポーネントの使用

複雑で構造化されたアプリケーションを Flash で作成するには、スクリーンでコンポーネントを使用します。コンポーネントはフォームで使用すると非常に有用で、データを表示する構造化アプリケーションを作成することや、非連続的なユーザー操作を受け付けるインタラクティブな操作性を実現することができます。たとえば、フォームを使用してコンテナコンポーネントを埋め込みます。

コンポーネント間のカスタムナビゲーションを作成するには、スクリーンでコンポーネントを使用するとフォーカスマネージャを使用できます。フォーカスマネージャは、ユーザーが Tab キーを押してアプリケーション内を移動したときに、コンポーネントがフォーカスを受け取る順序を指定します。たとえば、ユーザーが Tab キーを押してフィールド間を移動し、Return キー (Macintosh) または Enter キー (Windows) を押してフォームを送信できるように、フォームアプリケーションをカスタマイズできます。

フォーカスマネージャの詳細については、『[ActionScript 2.0 コンポーネントリファレンスガイド](#)』の [61 ページ](#)の「[カスタムフォーカスナビゲーションの作成](#)」および「FocusManager クラス」を参照してください。

また、タブ順序は [アクセシビリティ] パネルからも作成できます。詳細については、『Flash ユーザーガイド』の「[タブ順序と読み取り順序の表示と作成](#)」を参照してください。

カスタムフォーカスナビゲーションの作成

ユーザーが Tab キーを押して Flash アプリケーション内を移動したり、アプリケーション内をクリックしたりすると、FocusManager クラスは入力フォーカスを受け取るコンポーネントを判別します (詳細については、『[ActionScript 2.0 コンポーネントリファレンスガイド](#)』の FocusManager クラスを参照してください)。フォーカスマネージャを有効にするために、FocusManager インスタンスをアプリケーションに追加したり、コードを書いたりする必要はありません。

RadioButton オブジェクトがフォーカスを取得した場合、フォーカスマネージャは、そのオブジェクトと共通の groupName 値を持つすべてのオブジェクトを検査し、selected プロパティが true であるオブジェクトにのみフォーカスを設定します。

各モーダル Window コンポーネントはフォーカスマネージャのインスタンスをそれぞれ備えており、そのウィンドウ上のコントロールは独自のタブセットになります。そのため、ユーザーが Tab キーを押して、意図せずに他のウィンドウ上にあるコンポーネントに移動することはありません。

アプリケーション内のフォーカスナビゲーションを作成するには、フォーカスを受け取るコンポーネント (ボタンも含む) すべてについて、tabIndex プロパティを設定します。ユーザーが Tab キーを押すと、FocusManager は、現在の tabIndex 値より大きな tabIndex 値を持つ有効なオブジェクトを探します。FocusManager クラスは、最も値の大きい tabIndex プロパティに達した後、0 に戻ります。たとえば、次のコードでは、comment オブジェクト (おそらく TextArea コンポーネント) が先にフォーカスを取得し、その後で okButton インスタンスがフォーカスを取得します。

```
var comment:mx.controls.TextArea;
var okButton:mx.controls.Button;
comment.tabIndex = 1;
okButton.tabIndex = 2;
```

また、[アクセシビリティ] パネルを使用して、`tabIndex` の値を割り当てることもできます。

`tabIndex` 値を持つオブジェクトがステージ上にない場合、フォーカスマネージャは深度 (z 順序) を使用します。深度は主に、コンポーネントがステージにドラッグされた順序によって設定されます。ただし、[修正] メニューの [重ね順]-[最前面へ] または [重ね順]-[最背面へ] の各コマンドを使用して最終的な z 順序を決定することもできます。

アプリケーションでコンポーネントにフォーカスを当てるには、「`FocusManager.setFocus()`」を呼び出します。

ユーザーが Enter キー (Windows) または Return キー (Macintosh) を押した場合にフォーカスを受け取るボタンを作成するには、次のコードのように、そのボタンのインスタンスを「`FocusManager.defaultPushButton`」プロパティに設定します。

```
focusManager.defaultPushButton = okButton;
```

`FocusManager` クラス (API) は、Flash Player のデフォルトのフォーカス領域を表す矩形を無効にして、角の丸いカスタムのフォーカス用矩形を描きます。

Flash アプリケーションでのフォーカス順序の作成に関する詳細については、『`ActionScript 2.0` コンポーネントリファレンスガイド』の `FocusManager` クラスを参照してください。

ドキュメント内のコンポーネントの深度を管理する

アプリケーションの中で、あるコンポーネントを別のオブジェクトの前面や背面に配置するには、『`ActionScript 2.0` コンポーネントリファレンスガイド』で説明されている `DepthManager` クラスを使用する必要があります。`DepthManager` クラスのメソッドを使用すると、ユーザーインターフェイスコンポーネントを適切な " 相対 " 順序で配置できます。たとえば、コンボボックスのドロップダウンリストは他のコンポーネントより手前に表示する、挿入位置は他のすべての要素より手前に表示する、ダイアログボックスはコンテンツの前面に表示する、などの規則を指定できます。

深度マネージャの目的は主に 2 つあります。任意のドキュメント内で相対的な深度の割り当てを管理することと、カーソルやツールヒントなどのシステムレベルのサービス用に予約された深度をルートオブジェクトのタイムラインで管理することです。

深度マネージャを使用するには、そのメソッドを呼び出します。

次のコードは、コンポーネントインスタンス loader を、button コンポーネントよりも下に配置します (2 つが重なる場合、パブリッシュされる SWF ファイルでは、このインスタンスはボタンの " 下 " に表示されます)。

```
loader.setDepthBelow(button);
```

×
❌

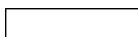
ドキュメント内の [レイヤー] オプションまたは [修正]-[重ね順] オプションを使って相対的な深さを管理することもできます。コンポーネントは、レイヤーおよび配置を使用したランタイムの深度の管理について、ムービークリップと同じ規則に従います。

ライブプレビュー内のコンポーネント

ライブプレビュー機能はデフォルトで有効になっており、ステージ上のコンポーネントがパブリッシュされた Flash コンテンツでどのように表示されるかを確認できます。コンポーネントは、おおよそのサイズで表示されます。ライブプレビューには、さまざまなコンポーネントに指定された各パラメータが反映されます。ライブプレビューに反映されるコンポーネントのパラメータの詳細については、『ActionScript 2.0 コンポーネントリファレンスガイド』の各コンポーネントのセクションを参照してください。



ライブプレビューを有効にした Button コンポーネント



ライブプレビューを無効にした Button コンポーネント

ライブプレビュー内のコンポーネントは機能しません。コンポーネントの機能をテストするには、[制御]-[ムービープレビュー] コマンドを使用します。

ライブプレビューのオンとオフを切り替えるには：

- [制御]-[ライブプレビューを有効にする] を選択します。オプションの横にチェックマークが付いていれば、そのオプションは有効になっています。

コンポーネントでのプリローダーの使用

プリロードすると、SWF ファイルのデータの一部分が、ユーザーが操作を開始する前にロードされます。デフォルトでは、各コンポーネントおよびクラスは、コンポーネントを含むドキュメントの最初のフレームに書き出すよう設定されています。コンポーネントとクラスは、ロードの対象となる最初のデータであるため、プログレスバーの実装やアニメーションのロードで問題が発生する可能性があります。具体的には、コンポーネントとクラスはプログレスバーの前にロードされるが、プログレスバーに、すべてのデータ (クラスを含む) のロードの進行状況を表示したい場合が考えられます。この場合、クラスをロードするタイミングを、SWF ファイルの他の部分をロードするよりは後、ただしコンポーネントを使用するよりは前にする必要があります。

この操作を実行するには、コンポーネントを含むアプリケーション用のカスタムプリローダーを作成するときに、コンポーネントを含むフレームにすべてのクラスを書き出すよう、ファイルのパブリッシュ設定を設定します。[最初のフレームに書き出し] に設定されたアセットを持つ Halo テーマおよび Sample テーマのすべてのコンポーネントのリストについては、[117 ページの「書き出し設定の変更」](#)を参照してください。

すべてのクラス用の書き出しフレームを変更するには：

1. [ファイル]-[パブリッシュ設定] を選択します。
2. [パブリッシュ設定] ダイアログボックスの [Flash] タブで、[ActionScript のバージョン] が [ActionScript 2.0] に設定されていることを確認します。
3. [ActionScript のバージョン] の右にある [設定] ボタンをクリックします。
4. [ActionScript 2.0 設定] の [クラス用のフレームの書き出し] テキストボックスの数値を、コンポーネントが最初に表示されるフレームに変更します。

クラスのロードを指定したフレームに再生ヘッドが到達するまでは、どのクラスも使用できません。コンポーネントが機能するためにはクラスが必要なので、コンポーネントは、クラスのロード用に指定したフレームの後にロードする必要があります。クラスをフレーム 3 に書き出した場合は、再生ヘッドがフレーム 3 に達してデータがロードされるまで、クラスのどのような機能も使用できません。

コンポーネントを使用するファイルをプリロードする場合は、SWF ファイル内のコンポーネントもプリロードしなければなりません。これを実現するには、コンポーネントが SWF ファイル内の異なるフレームに書き出されるように設定する必要があります。

× ❌	ActionScript を使用してコンポーネントをステージに追加する場合は、追加するコンポーネントのインスタンスをペーストボード (ステージの周囲の領域) にドラッグする必要があります。こうすると Flash は、アプリケーションでコンポーネントが使用されていて、それが未使用のライブラリアイテムでないことを認識します。未使用のライブラリアイテムは SWF ファイルに追加されない点に注意してください。
--------	--

コンポーネントの書き出し先のフレームを変更するには：

1. [ウィンドウ]-[ライブラリ]を選択して、[ライブラリ]パネルを開きます。
2. ライブラリ内のコンポーネントを右クリック (Windows) または Control キーを押しながらクリック (Macintosh) します。
3. ショートカットメニューから [リンケージ] を選択します。
4. [最初のフレームに書き出し] をオフにします。
5. [OK] をクリックします。
6. [ファイル]-[パブリッシュ設定] を選択します。
7. [Flash] タブを選択し、[設定] ボタンをクリックします。
8. [クラス用のフレームの書き出し] テキストボックスに数値を入力し、[OK] をクリックします。クラスはこのフレーム内にロードされます。
9. [OK] をクリックして [パブリッシュ設定] ダイアログボックスを閉じます。

コンポーネントを最初のフレームにロードしなければ、SWF ファイルの最初のフレームのカスタムプログレスバーを作成することができます。手順 7 で指定したフレームのクラスをロードするまでは、ActionScript でコンポーネントを参照することも、ステージに何かコンポーネントを含めることもできません。



コンポーネントは、そのコンポーネントで使用する ActionScript クラスよりも後に書き出す必要があります。

コンポーネントのロードについて

バージョン 2 コンポーネントを SWF ファイルまたは Loader コンポーネントにロードしても、コンポーネントが正しく動作しないことがあります。このようなコンポーネントには、Alert、ComboBox、DateField、Menu、MenuBar、Window があります。

loadMovie() を呼び出す場合、または Loader コンポーネントにロードする場合は、_lockroot プロパティを使用します。Loader コンポーネントを使用している場合は、次のコードを追加します。

```
myLoaderComponent.content._lockroot = true;
```

ムービークリップで loadMovie() への呼び出しを使用している場合は、次のコードを追加します。

```
myMovieClip._lockroot = true;
```

ローダーにロードされるムービークリップの中で _lockroot が true に設定されていない場合、ローダーがアクセスできるのは自身のライブラリのみで、ロードされたムービークリップ内のライブラリにはアクセスできません。

_lockroot プロパティは、Flash Player 7 でサポートされています。このプロパティの詳細については、『ActionScript 2.0 リファレンスガイド』の「_lockroot (MovieClip._lockroot プロパティ)」を参照してください。

バージョン 1 コンポーネントからバージョン 2 アーキテクチャへのアップグレード

バージョン 2 コンポーネントは、イベント (www.w3.org/TR/DOM-Level-3-Events/events.html を参照)、スタイル、getter/setter の考え方などに関するいくつかの Web 標準に準拠しており、Macromedia Flash MX と Macromedia Flash MX 2004 以前にリリースされた DRK に含まれているバージョン 1 コンポーネントとは大きく異なります。バージョン 2 コンポーネントは異なる API を持ち、ActionScript 2.0 で記述されています。このため、同じアプリケーション内でバージョン 1 コンポーネントとバージョン 2 コンポーネントを共存させると、予測しない動作が起きる可能性があります。バージョン 1 コンポーネントをアップグレードして、バージョン 2 のイベント処理やスタイルを使用する方法、およびメソッドではなくプロパティへの getter/setter アクセスができるようにする方法については、[125 ページ、第 6 章の「コンポーネントの作成」](#)を参照してください。

バージョン 1 コンポーネントを含む Flash アプリケーションを Flash Player 6 または Flash Player 6 (6.0.65.0) 向けにパブリッシュした場合、これらのアプリケーションは Flash Player 6 および Flash Player 7 で正しく動作します。Flash Player 7 以降向けにパブリッシュして動作させるようにアプリケーションを更新する場合は、厳密な型指定を使用できるようにコードを変換する必要があります。詳細については、『ActionScript 2.0 の学習』の「カスタムクラスファイルの記述」を参照してください。

コンポーネントイベントの処理

すべてのコンポーネントは、ユーザーがコンポーネントを操作するとブロードキャストされるイベント (click イベントや change イベントなど)、またはコンポーネントに重大な変化が生じたときにブロードキャストされるイベント (load イベントなど) を備えています。イベントを処理するには、イベントが発生したときに実行する **ActionScript** コードを作成します。

各コンポーネントでは、それぞれに特有のイベントセットがブロードキャストされます。このセットには、当該コンポーネントが継承しているクラスすべてに関するイベントが含まれます。**UIObject** クラスと **UIComponent** クラスはバージョン 2 アーキテクチャの基本クラスなので、メディアコンポーネントを除くすべてのコンポーネントは、**UIObject** クラスと **UIComponent** クラスのイベントを継承しています。コンポーネントがブロードキャストするイベントの一覧については、『**ActionScript 2.0** コンポーネントリファレンスガイド』の特定のコンポーネントおよびその祖先クラスの項目を参照してください。

この章では、**TipCalculator** という単純な Flash アプリケーションを例にとって数種類のバージョンを示し、コンポーネントイベントの処理方法について説明します。**TipCalculator** のサンプルについては、Web サイトの Flash サンプルページ (www.adobe.com/go/learn_fl_samples_jp) を参照してください。

この章では、次のセクションについて説明します。

リスナーによるイベントの処理.....	68
イベントの委譲.....	76
イベントオブジェクトについて.....	79
on() コンポーネントイベントハンドラの使用.....	80

リスナーによるイベントの処理

バージョン 2 コンポーネントアーキテクチャでは、ブロードキャスター / リスナーイベントモデルを使用しています。"ブロードキャスター"は"ディスパッチャー"とも呼ばれます。このモデルに関しては、次のキーポイントを理解しておくことが重要です。

- すべてのイベントは、コンポーネントクラスのインスタンスによってブロードキャストされます。ブロードキャストを実行したコンポーネントインスタンスが"ブロードキャスター"です。
- "リスナー"は関数またはオブジェクトです。オブジェクトをリスナーとする場合、そのオブジェクトにはコールバック関数を定義しておく必要があります。リスナーがイベントを"処理"するとは、イベントが発生したときにコールバック関数を実行することを意味します。

- ブロードキャスターに対してリスナーを登録するには、ブロードキャスターの `addEventListener()` メソッドを呼び出します。これには次のシンタックスを使用します。
`componentInstance.addEventListener("eventName", listenerObjectORFunction);`

- 1つのコンポーネントインスタンスには複数のリスナーを登録できます。

```
myButton.addEventListener("click", listener1);  
myButton.addEventListener("click", listener2);
```

- また、1つのリスナーを複数のコンポーネントインスタンスに登録することもできます。

```
myButton.addEventListener("click", listener1);  
myButton2.addEventListener("click", listener1);
```

- ハンドラ関数には、イベントオブジェクトが渡されます。

関数本体の中では、イベントオブジェクトを使用して、イベントタイプの情報と、イベントをブロードキャストしたインスタンスの情報を取得します。[79 ページの「イベントオブジェクトについて」](#)を参照してください。

- リスナーオブジェクトは、「`EventDispatcher.removeEventListener()`」によって明示的に削除されるまではアクティブな状態を維持します。次に例を示します。

```
myComponent.removeEventListener("change", ListenerObj);
```

リスナーオブジェクトの使用

リスナーオブジェクトを使用するには、`this` キーワードで現在のオブジェクトをリスナーとして指定する方法、アプリケーション内の既存のオブジェクトを使用する方法、または新しいオブジェクトを作成する方法があります。

- `this` を使用する方法が最も一般的です。

現在のオブジェクト (`this`) をリスナーとして登録すると、ブロードキャストされたイベントに反応するコンポーネントが現在のスコープに含まれているので、通常はこの方法が最も簡単です。

- 既存のオブジェクトを登録するのが都合な場合は、その方法を使用します。

たとえば、Flash フォームアプリケーションで、イベントに反応するコンポーネントがフォームに含まれている場合は、そのフォームをリスナーオブジェクトとして使用します。フォームのタイムラインのフレームにコードを配置します。

- 多くのコンポーネントからブロードキャストされるイベント (`click` イベントなど) について、少数のリスナーオブジェクトだけで応答する必要がある場合は、新しいリスナーオブジェクトを使用します。

`this` オブジェクトを使用する場合は、処理するイベントと同じ名前の関数を定義します。シンタックスは次のとおりです。

```
function eventName(evtObj:Object){  
    // ここにコードを記述  
};
```

新しいリスナーオブジェクトを使用する場合は、オブジェクトを作成し、イベントと同じ名前のプロパティを定義して、イベントがブロードキャストされたとき実行するコールバック関数にそのプロパティを割り当てる必要があります。シンタックスは次のとおりです。

```
var listenerObject:Object = new Object();  
listenerObject.eventName = function(evtObj:Object){  
    // ここにコードを記述  
};
```

既存のオブジェクトを使用する場合は、新しいリスナーオブジェクトの場合と同じシンタックスを使用します。ただし、新しいオブジェクトを作成する必要はありません。シンタックスは次のとおりです。

```
existingObject.eventName = function(evtObj:Object){  
    // ここにコードを記述  
};
```

メモ

`evtObj` パラメータは、イベントがトリガされたときに自動生成されるオブジェクトで、コールバック関数に渡されます。イベントオブジェクトには、イベントに関する情報を含むプロパティがあります。詳細については、[79 ページの「イベントオブジェクトについて」](#)を参照してください。

最後に、イベントをブロードキャストするコンポーネントインスタンスの `addEventListener()` メソッドを呼び出します。`addEventListener()` メソッドには 2 つのパラメータとして、イベントの名前を示す文字列と、リスナーオブジェクトへの参照を渡します。

```
componentInstance.addEventListener("eventName", listenerObject);
```

次に示すコードセグメントは、全体をコピー、ペーストして流用できます。ただし、コードのイタリック体の箇所は実際の値に置き換えてください。`listenerObject` と `evtObj` はそのままでも識別子として有効ですが、`eventName` はイベントの名前に変更する必要があります。

```
var listenerObject:Object = new Object();
listenerObject.eventName = function(evtObj:Object){
    // ここに記述するコードは
    // イベントがトリガされると実行される
};
componentInstance.addEventListener("eventName", listenerObject);
```

次のコードセグメントでは、`this` キーワードをリスナーオブジェクトとして使用しています。

```
function eventName(evtObj:Object){
    // ここに記述するコードは
    // イベントがトリガされると実行される
}
componentInstance.addEventListener("eventName", this);
```

`addEventListener()` は、どのコンポーネントインスタンスに対しても呼び出せます。これは `EventDispatcher` クラスからの Mix-in として、すべてのコンポーネントに備わっています。"Mix-in" とはクラスの一つで、別のクラスの動作を強化するために特定の機能を提供するものです。詳細については、『*Action Script 2.0 コンポーネントリファレンスガイド*』の「`EventDispatcher.addEventListener()`」を参照してください。

コンポーネントがブロードキャストするイベントの詳細については、『*ActionScript 2.0 コンポーネントリファレンスガイド*』で該当するコンポーネントのセクションを参照してください。たとえば、`Button` コンポーネントイベントは `Button` コンポーネントの項にリストされています (または、[ヘルプ] の [*ActionScript 2.0 コンポーネントリファレンスガイド*]-[`Button` コンポーネント]-[`Button` クラス]-[`Button` クラスのイベント一覧] を参照)。

Flash (FLA) ファイルにリスナーオブジェクトを登録するには：

1. Flash で、[ファイル]-[新規] を選択し、新しい Flash ドキュメントを作成します。
2. `Button` コンポーネントを、[コンポーネント] パネルからステージヘドラッグします。
3. プロパティインスペクタで、インスタンス名として「`myButton`」を入力します。
4. `TextInput` コンポーネントを、[コンポーネント] パネルからステージヘドラッグします。
5. プロパティインスペクタで、インスタンス名として「`myText`」を入力します。
6. タイムラインでフレーム 1 を選択します。
7. [ウィンドウ]-[アクション] を選択します。

8. [アクション] パネルに次のコードを入力します。

```
var myButton:mx.controls.Button;  
var myText:mx.controls.TextInput;
```

```
function click(evt){  
    myText.text = evt.target;  
}
```

```
myButton.addEventListener("click", this);
```

evt イベントオブジェクトの target プロパティは、イベントをブロードキャストしたインスタンスへの参照です。このコードでは、TextInput コンポーネントの target プロパティの値を表示します。

クラス (AS) ファイルにリスナーオブジェクトを登録するには：

1. [49 ページの「コンポーネントの利用」](#)で指定されている場所から "TipCalculator.fla" ファイルを開きます。
2. [49 ページの「コンポーネントの利用」](#)で指定されている場所から "TipCalculator.as" ファイルを開きます。
3. FLA ファイル内でフォーム 1 を選択し、プロパティインスペクタに TipCalculator というクラス名を表示します。

これはフォームとクラスファイルの間のリンクです。このアプリケーションのコードはすべて "TipCalculator.as" ファイル内にあります。フォームは、このクラスで定義されるプロパティとビヘイビアがこのフォームに割り当てられているものとして動作します。

4. AS ファイルをスクロールして、25 行目の public function onLoad():Void を表示します。
onLoad() 関数は、フォームが Flash Player にロードされたとき実行されます。関数の本体では、subtotal TextInput インスタンスと 3 つの RadioButton インスタンス (percentRadio15、percentRadio18、および percentRadio20) が addEventListener() メソッドを呼び出して、リスナーをイベントに登録します。
5. 27 行目の subtotal.addEventListener("change", this) に移ります。
addEventListener() を呼び出す場合は、2 つのパラメータを渡す必要があります。最初のパラメータはブロードキャストされるイベントの名前を示すストリングで、ここでは "change" です。2 番目のパラメータは、オブジェクトまたはイベントを処理する関数への参照です。ここでは、クラスファイルのインスタンス (オブジェクト) を参照する this キーワードを渡します。Flash はこのオブジェクト内でイベントと同じ名前の関数を探して呼び出します。
6. 63 行目の public function change(event:Object):Void に移ります。
これは、subtotal TextInput インスタンスに変更があると実行される関数です。
7. "TipCalculator.fla" を選択し、[制御]-[ムービープレビュー] を選択してファイルをテストします。

handleEvent コールバック関数の使用

handleEvent 関数をサポートするリスナーオブジェクトを使用することもできます。ブロードキャストされるイベントの名前に関係なく、リスナーオブジェクトの handleEvent() メソッドが呼び出されます。複数のイベントを処理するには if..else または switch ステートメントを使う必要があります。たとえば、次のコードでは if..else ステートメントを使って、click と change の 2 つのイベントを処理しています。

```
// handleEvent 関数の定義
// evt をイベントオブジェクトパラメータとして渡す

function handleEvent(evt){
  // イベントが click かどうかを確認
  if (evt.type == "click"){
    // イベントが click だった場合の処理
  } else if (evt.type == "change"){
    // イベントが change だった場合の処理
  }
};

// リスナーオブジェクトを
// 2 つの異なるコンポーネントインスタンスに登録する。
// 関数が "this" オブジェクトで定義されているので、
// リスナーとして this を登録する。

instance.addEventListener("click", this);
instance2.addEventListener("change", this);
```

リスナー関数の使用

一部のリスナー関数では、handleEvent シンタックスと異なり、異なるイベントを処理できます。そのため、myHandler に if および else if チェックを記述する代わりに、次のように change イベント用に myChangeHandler を定義し、scroll イベント用に myScrollHandler を定義して、登録することができます。

```
myList.addEventListener("change", myChangeHandler);
myList.addEventListener("scroll", myScrollHandler);

リスナー関数を使用するには、まず関数を定義する必要があります。

function myFunction:Function(evtObj:Object){
  // ここにコードを記述
}
```

📌

evtObj パラメータは、イベントがトリガされたときに自動生成され、関数に渡されるオブジェクトです。イベントオブジェクトには、イベントに関する情報を含むプロパティがあります。詳細については、[79 ページの「イベントオブジェクトについて」](#)を参照してください。

次に、イベントをブロードキャストするコンポーネントインスタンスの `addEventListener()` メソッドを呼び出します。`addEventListener()` メソッドには 2 つのパラメータとして、イベントの名前を示す文字列と、関数への参照を渡します。

```
componentInstance.addEventListener("eventName", myFunction);
```

`addEventListener()` は、どのコンポーネントインスタンスに対しても呼び出せます。これは `EventDispatcher` クラスからの Mix-in として、すべての UI コンポーネントに備わっています。詳細については、『*ActionScript 2.0 コンポーネントリファレンスガイド*』の「`EventDispatcher.addEventListener()`」を参照してください。

コンポーネントがブロードキャストするイベントの詳細については、『*ActionScript 2.0 コンポーネントリファレンスガイド*』の各コンポーネントのセクションを参照してください。

Flash (FLA) ファイルにリスナーオブジェクトを登録するには：

1. Flash で、[ファイル]-[新規] を選択し、新しい Flash ドキュメントを作成します。
2. List コンポーネントを、[コンポーネント] パネルからステージヘドラッグします。
3. プロパティインスペクタで、インスタンス名として「`myList`」と入力します。
4. タイムラインでフレーム 1 を選択します。
5. [ウィンドウ]-[アクション] を選択します。
6. [アクション] パネルに次のコードを入力します。

```
// 変数の宣言
var myList:mx.controls.List;
var myHandler:Function;

// リストに項目を追加
myList.addItem("Bird");
myList.addItem("Dog");
myList.addItem("Fish");
myList.addItem("Cat");
myList.addItem("Ape");
myList.addItem("Monkey");

// myHandler 関数を定義
function myHandler(eventObj:Object){

    // eventObj パラメータを使用して
    // イベントタイプを取得する
    if (eventObj.type == "change"){
        trace("The list changed");
    } else if (eventObj.type == "scroll"){
        trace("The list was scrolled");
    }
}

// myHandler 関数を myList に登録する。
```

```
// 項目が選択される (change イベントがトリガされる) か、  
// リストがスクロールされると、myHandler が実行される。  
myList.addEventListener("change", myHandler);  
myList.addEventListener("scroll", myHandler);
```

evt イベントオブジェクトの type プロパティは、イベント名への参照です。

7. [制御]-[ムービープレビュー]を選択します。リスト内の項目を選択し、リストをスクロールして、[出力]パネルに表示される結果を確認します。

リスナー関数内の `this` キーワードは、関数が定義されているタイムラインやクラスではなく、`addEventListener()` を呼び出したコンポーネントインスタンスを参照します。ただし、`Delegate` クラスを使用してリスナー関数を別のスコープに委譲することもできます。[76 ページの「イベントの委譲」](#)を参照してください。関数のスコープ設定の例については、次の項を参照してください。

リスナー内のスコープについて

" スコープ " とは、関数が実行されるときに属するオブジェクトです。その関数の中で行われるすべての変数参照は、そのオブジェクトのプロパティとして認識されます。Delegate クラスを使用すると、リスナーのスコープを指定できます。詳細については、[76 ページの「イベントの委譲」](#)を参照してください。

前述のように、リスナーをコンポーネントインスタンスに登録するには、`addEventListener()` を呼び出します。このメソッドには 2 つのパラメータとして、イベントの名前を示す文字列と、オブジェクトまたは関数への参照を渡します。次の表は、各パラメータタイプのスコープの一覧です。

リスナーのタイプ	スコープ
Object	リスナーオブジェクト
Function	イベントをブロードキャストするコンポーネントインスタンス

addEventListener() にオブジェクトを渡すと、そのオブジェクトに割り当てられているコールバック関数（またはそのオブジェクトで定義されている関数）が、オブジェクトのスコープで起動されます。つまり、コールバック関数の中で使用されている場合、this キーワードはリスナーオブジェクトを参照します。

```
var lo:Object = new Object();
lo.click = function(evt){
    // this は lo オブジェクトを参照
    trace(this);
}
myButton.addEventListener("click", lo);
```

一方、`addEventListener()` に関数を渡すと、その関数が、`addEventListener()` を呼び出したコンポーネントインスタンスのスコープで起動されます。つまり、関数の中で使用されている場合、`this` キーワードはブロードキャストしたコンポーネントインスタンスを参照します。このため、関数をクラスファイルの中で定義していると問題が発生します。`this` がクラスのインスタンスを指していないので、本来のパスを使用してクラスファイルのプロパティとメソッドにアクセスできないためです。この問題を解決するには、**Delegate** クラスを使用して、関数を適切なスコープに委譲します。[76 ページの「イベントの委譲」](#)を参照してください。

次のコードは、クラスファイル内で `addEventListener()` に関数を渡した場合の、関数のスコープを示しています。このコードを使用するには、"**Cart.as**" という名前の **ActionScript (AS)** ファイルにコードをコピーします。次に、`myButton` という **Button** コンポーネントと `myGrid` という **DataGrid** コンポーネントから成る **Flash (FLA)** ファイルを作成します。ステージで両方のコンポーネントを選択し、**F8** キーを押して、これらを新しい **Cart** というシンボルに変換します。**Cart** シンボルの [**リネージプロパティ**] で、これを **Cart** クラスに割り当てます。

```
class Cart extends MovieClip {

    var myButton:mx.controls.Button;
    var myGrid:mx.controls.DataGrid;

    function myHandler(eventObj:Object){

        // eventObj パラメータを使用して
        // イベントタイプを取得する。
        if (eventObj.type == "click"){

            /* this の値を [ 出力 ] パネルに送る。
            myHandler は、リスナーオブジェクトで
            定義される関数ではない。したがって、this は、
            myHandler を登録するコンポーネントインスタンスへの参照
            (myButton)。また、this は Cart クラスのインスタンスを
            参照していないため、myGrid は未定義となる。
            */
            trace("this: " + this);
            trace("myGrid: " + myGrid);
        }
    }

    // myHandler 関数を myButton に登録する。
    // ボタンがクリックされると、myHandler が実行される。

    function onLoad():Void{
        myButton.addEventListener("click", myHandler);
    }
}
```

イベントの委譲

Delegate クラスをスクリプトやクラスに読み込んで、イベントを特定のスコープや関数に委譲できます (『ActionScript 2.0 コンポーネントリファレンスガイド』の「Delegate クラス」を参照してください)。Delegate クラスを読み込むには、次のシンタックスを使用します。

```
import mx.utils.Delegate;  
compInstance.addEventListener("eventName", Delegate.create(scopeObject,  
    function));
```

scopeObject パラメータは、指定の *function* パラメータが呼び出されるスコープを指定します。

Delegate.create() の呼び出しは、一般に次の 2 つの目的で使用されます。

- 2 つの異なる関数に対して同じイベントを送り出す場合。
詳細については、次のセクションを参照してください。
- その関数が含まれているクラスのスコープで関数を呼び出す場合。
関数をパラメータとして addEventListener() に渡すと、関数が宣言されているオブジェクトではなく、ブロードキャスターコンポーネントインスタンスのスコープで関数が起動されます。
[78 ページの「関数のスコープの委譲」](#)を参照してください。

関数へのイベントの委譲

同じ名前のイベントをブロードキャストするコンポーネントが 2 つある場合は、Delegate.create() の呼び出しを使用すると便利です。たとえば、チェックボックスとボタンがある場合などで、click イベントがいずれのコンポーネントからブロードキャストされたかを判断するために、switch ステートメントで eventObject.target プロパティから取得した情報を区別することがあります。

次のコードを使用するには、myCheckBox_chb という名前のチェックボックスと myButton_btn という名前のボタンをステージに配置します。両方のインスタンスを選択し、F8 キーを押して新しいシンボルを作成します。ダイアログボックスが基本モードになっている場合は [詳細] をクリックし、[ActionScript に書き出し] を選択します。[AS 2.0 クラス] テキストボックスに「Chart」と入力します。プロパティインスペクタで、必要に応じた任意のインスタンス名を新しいシンボルに設定します。これによってシンボルが Cart クラスに関連付けられ、このシンボルのインスタンスは Cart クラスのインスタンスとなります。

```
import mx.controls.Button;  
import mx.controls.CheckBox;  
  
class Cart {  
    var myCheckBox_chb:CheckBox;  
    var myButton_btn:Button;  
  
    function onLoad() {  
        myCheckBox_chb.addEventListener("click", this);  
        myButton_btn.addEventListener("click", this);  
    }  
}
```

```

    }

    function click(eventObj:Object) {
        switch(eventObj.target) {
            case myButton_btn:
                // ブロードキャスターのインスタンス名と
                // イベントタイプを [ 出力 ] パネルに送る
                trace(eventObj.target + ": " + eventObj.type);
                break;
            case myCheckBox_chb:
                trace(eventObj.target + ": " + eventObj.type);
                break;
        }
    }
}

```

次のコードは、**Delegate** を使用するように同じクラスファイル (**Cart.as**) を変更したものです。

```

import mx.utils.Delegate;
import mx.controls.Button;
import mx.controls.CheckBox;

class Cart {
    var myCheckBox_chb:CheckBox;
    var myButton_btn:Button;

    function onLoad() {
        myCheckBox_chb.addEventListener("click", Delegate.create(this,
        chb_onClick));
        myButton_btn.addEventListener("click", Delegate.create(this, btn_onClick));
    }

    // イベントを処理する 2 つの関数

    function chb_onClick(eventObj:Object) {
        // ブロードキャスターのインスタンス名と
        // イベントタイプを [ 出力 ] パネルに送る
        trace(eventObj.target + ": " + eventObj.type);
        // FLA ファイルで Cart クラスに関連付けた
        // シンボルの絶対パスを
        // [ 出力 ] パネルに送る
        trace(this)
    }

    function btn_onClick(eventObj:Object) {
        trace(eventObj.target + ": " + eventObj.type);
    }
}

```

関数のスコープの委譲

`addEventListener()` メソッドには 2 つのパラメータがあります。イベントの名前とリスナーへの参照です。リスナーは、オブジェクトまたは関数です。オブジェクトを渡すと、オブジェクトに割り当てられているコールバック関数が、オブジェクトのスコープで起動されます。また、関数を渡した場合は、`addEventListener()` を呼び出したコンポーネントインスタンスのスコープで関数が起動されます。詳細については、[74 ページの「リスナー内のスコープについて」](#)を参照してください。

関数はブロードキャスターインスタンスのスコープで起動されるため、関数の本体内にある `this` キーワードは、その関数を含んでいるクラスではなくブロードキャスターインスタンスを指します。このため、関数を含んでいるクラスのプロパティやメソッドにはアクセスできません。関数を含んでいるクラスのプロパティやメソッドにアクセスするには、`Delegate` クラスを使って、関数のスコープを、関数を含んでいるクラスに委譲します。

次に示す例では、前のセクションの例と同じ方法を使用していますが、"`Cart.as`" クラスファイルの内容が異なります。

```
import mx.controls.Button;
import mx.controls.CheckBox;

class Cart {

    var myCheckBox_chb:CheckBox;
    var myButton_btn:Button;

    // chb_onClick 関数からアクセスする変数を
    // 定義する。
    var i:Number = 10

    function onLoad() {
        myCheckBox_chb.addEventListener("click", chb_onClick);
    }

    function chb_onClick(eventObj:Object) {
        // 一見、変数 i にアクセスして
        // 10 を出力できるように見える。
        // しかし、i が定義されている Cart インスタンスが
        // 実際には関数のスコープではないため、
        // [出力] パネルに undefined が
        // 送られる。
        trace(i);
    }
}
```

`Cart` クラスのプロパティとメソッドにアクセスするには、`addEventListener()` の 2 番目のパラメータとして `Delegate.create()` への呼び出しを渡します。

```
import mx.utils.Delegate;
import mx.controls.Button;
import mx.controls.CheckBox;
```

```

class Cart {
    var myCheckBox_chb:CheckBox;
    var myButton_btn:Button;
    // chb_onClick 関数からアクセスする変数を
    // 定義する。
    var i:Number = 10

    function onLoad() {
        myCheckBox_chb.addEventListener("click", Delegate.create(this,
        chb_onClick));
    }

    function chb_onClick(eventObj:Object) {
        // 関数のスコープが
        // Cart インスタンスに設定されているため、
        // [ 出力 ] パネルに 10 が送られる。
        trace(i);
    }
}

```

イベントオブジェクトについて

イベントオブジェクトは、ActionScript Object クラスのインスタンスです。イベントに関する情報を含んだ次のプロパティがあります。

プロパティ	説明
type	イベントの名前を示す文字列です。
target	イベントをブロードキャストしたコンポーネントインスタンスへの参照です。

この他のプロパティについては、「コンポーネント辞書」の各イベントのセクションに一覧されています。イベントオブジェクトは、イベントがトリガされると自動的に生成され、リスナーオブジェクトのコールバック関数またはリスナー関数に渡されます。

関数内でイベントオブジェクトを使用すると、ブロードキャストされたイベントの名前や、イベントをブロードキャストしたコンポーネントのインスタンス名にアクセスできます。インスタンス名から、他のコンポーネントプロパティにアクセスすることもできます。たとえば、次のコードでは evtObj イベントオブジェクトの target プロパティを使用して、myButton インスタンスの label プロパティにアクセスし、値を [出力] パネルに送ります。

```

var myButton:mx.controls.Button;
var listener:Object;

listener = new Object();

listener.click = function(evtObj){
    trace("The " + evtObj.target.label + " button was clicked");
}
myButton.addEventListener("click", listener);

```

on() コンポーネントイベントハンドラの使用

ボタンやムービークリップにハンドラを割り当てると同じように、コンポーネントインスタンスには on() イベントハンドラを割り当てることができます。on() イベントハンドラは、簡単なテストを行う場合に便利ですが、すべてのアプリケーションに対しては、代わりにイベントリスナーを使用します。詳細については、[68 ページの「リスナーによるイベントの処理」](#)を参照してください。

コンポーネント ([アクション] パネルのコンポーネントインスタンスに割り当てられている) に直接割り当てられた on() ハンドラ内で this キーワードを使用する場合、this はこのコンポーネントインスタンスを参照します。たとえば、次のコードを Button コンポーネントインスタンス myButton に直接割り当てると、"_level0.myButton" を [出力] パネルに表示します。

```
on(click){
    trace(this);
}
```

on() ハンドラを使用するには：

1. User Interface コンポーネントをステージにドラッグします。
たとえば、Button コンポーネントをステージにドラッグします。
2. ステージで、コンポーネントを選択し、[アクション] パネルを開きます。
3. on() ハンドラを、次の形式で [アクション] パネルに追加します。

```
on(event){
    //your statements go here
}
```

次に例を示します。

```
on(click){
    trace(this);
}
```

Flash は、on() ハンドラのイベントが発生したとき (このケースではボタンがクリックされたとき) に、on() ハンドラ内のコードを実行します。

4. [制御]-[ムービープレビュー] を選択して、ボタンをクリックし、出力を表示します。

コンポーネントのカスタマイズ

コンポーネントの外観を、作成するアプリケーションに応じて変更する必要がある場合は、次の3つの方法をいずれか単独で、または組み合わせて使用することにより、コンポーネントの外観をカスタマイズできます。

スタイル ユーザーインターフェイス (UI) コンポーネントには各種のスタイルプロパティがあり、コンポーネントの外観をさまざまな観点から設定できます。各コンポーネントには、それぞれ固有の変更可能なスタイルプロパティのセットがあります。スタイルを設定する方法では、コンポーネントの外観のすべての部分を変更できるわけではありません。詳細については、[82 ページの「スタイルによるコンポーネントのカラーとテキストの変更」](#)を参照してください。

スキン スकिनは、あるコンポーネントのグラフィック表示を構築するシンボルのコレクションです。スキニングは、ソースのグラフィックを修正または置換することでコンポーネントの外観を変更するプロセスです。スキンには、境界線の端や角のような小さな要素もあれば、押されていない状態のボタン全体を表す画像のような複合要素もあります。また、画像を持つシンボルではなく、コンポーネントの一部を描くコードを含むシンボルをスキンとすることもできます。スキンを変更する方法では、スタイルプロパティで設定できないような観点からコンポーネントの外観をカスタマイズできる場合があります。詳細については、[96 ページの「コンポーネントへのスキンの適用について」](#)を参照してください。

テーマ テーマはスタイルおよびスキンの両方から成るコレクションであり、FLA ファイルとして保存することによって他のドキュメントに適用できます。詳細については、[108 ページの「テーマについて」](#)を参照してください。

この章では、次のセクションについて説明します。

スタイルによるコンポーネントのカラーとテキストの変更	82
コンポーネントへのスキンの適用について	96
テーマについて	108
スキンとスタイルの組み合わせによるコンポーネントのカスタマイズ	119

スタイルによるコンポーネントのカラーとテキストの変更

Flash では、どの UI コンポーネントにも編集可能なスタイルプロパティがあります。マニュアルで個々の特定コンポーネントの項目を参照すると、そのコンポーネントについて変更可能なスタイルの一覧表があります (たとえば、『[ActionScript 2.0 コンポーネントリファレンスガイド](#)』の「[Accordion コンポーネントでのスタイルの使用](#)」には Accordion コンポーネントのスタイルの一覧表があります)。また、UI コンポーネントには UIObject クラスから継承した `setStyle()` メソッドおよび `getStyle()` メソッドがあります (『[ActionScript 2.0 コンポーネントリファレンスガイド](#)』の「[UIObject.setStyle\(\)](#)」および「[UIObject.getStyle\(\)](#)」を参照してください)。コンポーネントインスタンスで `setStyle()` メソッドと `getStyle()` メソッドを使用することにより、スタイルプロパティの値を設定および取得できます。これについては [85 ページ](#)の「[コンポーネントインスタンスのスタイルの設定](#)」で後述します。



メディアコンポーネントに対してスタイルを設定することはできません。

スタイル宣言およびテーマの使用

より大局的な観点からは、スタイルをまとめて "スタイル宣言" として扱うことにより、複数のコンポーネントインスタンスのスタイルプロパティ値を制御できます。スタイル宣言は `CSSStyleDeclaration` クラスによって作成されるオブジェクトであり、スタイル宣言のプロパティはコンポーネントに割り当てることができるスタイル設定です。ActionScript におけるスタイル宣言の使用方法は、HTML ページに `CSS` (カスケーディングスタイルシート) を適用する方法がモデルとなっています。HTML ページの場合は、スタイルシートファイルを作成することにより、複数の HTML ページのコンテンツについてスタイルプロパティを定義できます。コンポーネントの場合は、スタイル宣言オブジェクトを作成し、スタイルプロパティをスタイル宣言オブジェクトに追加することにより、Flash ドキュメント内にある複数のコンポーネントについて外観を制御できます。

さらに、スタイル宣言をまとめたものは " テーマ " として扱われます。Flash には、コンポーネント用のビジュアルテーマとして、Halo (HaloTheme.fla) と Sample (SampleTheme.fla) の 2 つのテーマが用意されています。" テーマ " とは、ドキュメント内のコンポーネントの外観を決定する一群のスタイルとグラフィックをセットにしたものです。どのテーマも、複数のコンポーネントに対してスタイルを提供します。各コンポーネントでどのようなスタイルが使用されるかは、部分的には、ドキュメントでどのようなテーマが使用されるかに依存しています。defaultIcon など一部のスタイルは、それらに関連付けられたコンポーネントで、ドキュメントに適用されたテーマに関係なく使用されます。themeColor や symbolBackgroundColor などの他のスタイルは、対応するテーマの使用中のみに使用されます。たとえば、themeColor は、Halo テーマの使用中のみに使用されます。また、symbolBackgroundColor は、Sample テーマの使用中のみに使用されます。特定のコンポーネントに対して設定できるスタイルプロパティの種類を知るには、そのコンポーネントに割り当てるテーマを特定する必要があります。『ActionScript 2.0 コンポーネントリファレンスガイド』にある各コンポーネントのスタイルテーブルでは、各スタイルプロパティがこれらのテーマのうち一方に当てはまるのか、両方に当てはまるのかを確認できます。詳細については、[108 ページの「テーマについて」](#)を参照してください。

スタイル設定について

スタイルやスタイル宣言を使用していると、さまざまな方法でスタイルを設定できることがわかります (グローバル、テーマ、クラス、スタイル宣言、またはプロパティレベル)。また、スタイルプロパティによっては親コンポーネントから継承することもできます (たとえば、Accordion の子パネルではフォントの取り扱い方法を Accordion コンポーネントから継承できます)。スタイルの挙動に関しては次のような点が重要です。

テーマへの依存性 特定のコンポーネントに設定できるスタイルプロパティは、現在のテーマによって決まります。デフォルトで Flash コンポーネントには Halo テーマが使用されますが、それ以外に Sample テーマも用意されています。スタイルプロパティの一覧表 (たとえば、Button コンポーネントの場合は、『ActionScript 2.0 コンポーネントリファレンスガイド』の「Button コンポーネントでのスタイルの使用」) を読む際には、必要なスタイルがどのテーマでサポートされているかに注意する必要があります。表には、この情報が Halo、Sample、または両方と記載されています (両方と記載されているスタイルプロパティは、どちらのテーマでもサポートされます)。現在のテーマを変更する方法については、[109 ページの「テーマの切り替え」](#)を参照してください。

継承 ActionScript で継承を設定することはできません。コンポーネントの子は、あるスタイルを親コンポーネントから継承するように設定されているか、設定されていないかのいずれかです。

グローバルスタイルシート HTML ドキュメントに対する CSS では " カスケーディング " がサポートされていますが、Flash ドキュメントに対するスタイル宣言ではサポートされていません。スタイルシート宣言オブジェクトは、すべてアプリケーションレベル (グローバルレベル) で定義されます。

優先順位 同じコンポーネントスタイルを複数の方法で設定した場合 (たとえば、textColor をグローバルレベルとコンポーネントインスタンスレベルの両方で設定した場合)、[92 ページの「同じドキュメント内でのグローバル、カスタム、クラススタイルの使用」](#)に示す順序に従って最初に発見されたスタイルが使用されます。

スタイルの設定

スタイルプロパティ、スタイル宣言による複数スタイルプロパティの編成、および、テーマによる複数スタイルとグラフィックのさらに大きな編成という構造があることにより、コンポーネントのカスタマイズは次のような方法で実行できます。

- コンポーネントインスタンスにスタイルを設定する。

個々のコンポーネントインスタンスで、カラーやテキストのプロパティを変更することができません。これは場合によっては効果的ですが、ドキュメント内のすべてのコンポーネントについて個別にプロパティを設定する必要がある場合などは、かえって時間がかかる可能性もあります。

詳細については、[85 ページの「コンポーネントインスタンスのスタイルの設定」](#)を参照してください。

- グローバルスタイル宣言を調整して、ドキュメント内のすべてのコンポーネントにスタイルを設定する。

ドキュメント全体に一貫性のある外観を適用するには、グローバルスタイル宣言でスタイルを作成します。

詳細については、[87 ページの「グローバルスタイルの設定」](#)を参照してください。

- カスタムスタイル宣言を作成し、複数のコンポーネントインスタンスに適用する。

ドキュメント内にあるコンポーネントのうち特定のグループに共通のスタイルを適用するには、カスタムスタイル宣言を作成して、指定したコンポーネントに適用します。

詳細については、[88 ページの「コンポーネントのグループに対するカスタムスタイルの設定」](#)を参照してください。

- デフォルトのクラススタイル宣言を作成する。

特定クラスのすべてのインスタンスに適用されるデフォルトの外観を指定するには、デフォルトのクラススタイル宣言を定義します。

詳細については、[89 ページの「コンポーネントクラスのスタイルの設定」](#)を参照してください。

- 継承スタイルを使って、ドキュメントの特定部分に含まれるコンポーネントにスタイルを設定する。

コンテナにスタイルのプロパティを設定すると、そのコンテナに含まれるコンポーネントにプロパティの値が継承されます。

詳細については、[90 ページの「コンテナの継承スタイルの設定」](#)を参照してください。

ライブプレビュー機能を使ってステージにコンポーネントを表示する場合、スタイルプロパティに加えた変更は表示されません。詳細については、[63 ページの「ライブプレビュー内のコンポーネント」](#)を参照してください。

コンポーネントインスタンスのスタイルの設定

コンポーネントインスタンスのスタイルプロパティを設定または取得するには、ActionScript コードを記述します。「UIObject.setStyle()」と「UIObject.getStyle()」の両メソッドは、あらゆる UI コンポーネントから直接呼び出すことができます。次のシンタックスで、コンポーネントインスタンスのプロパティと値を指定します。

```
instanceName.setStyle("propertyName", value);
```

次のコードは、Halo テーマを使用する myButton という Button インスタンスのアクセントカラーを設定する例です。

```
myButton.setStyle("themeColor", "haloBlue");
```



文字列で指定する場合には、引用符で囲む必要があります。

プロパティとしてスタイルに直接アクセスすることもできます (たとえば、myButton.color = 0xFF00FF)。

setStyle() を使用してコンポーネントのインスタンスに設定したスタイルプロパティは、最も優先度が高く、スタイル宣言やテーマによる他のスタイル設定よりも優先されます。ただし、単一のコンポーネントインスタンスに対して setStyle() を使用して設定したプロパティが多ければ多いほど、該当するコンポーネントの実行時の表示速度は遅くなります。カスタマイズしたコンポーネントの表示速度を向上させるには、「UIObject.createClassObject()」を使用して、ActionScript でコンポーネントインスタンス作成時にスタイルプロパティを定義し、スタイル設定を *initObject* パラメータで指定する方法があります。たとえば次のコード行では、現在のライブラリに ComboBox コンポーネントがある場合に my_cb という名前のコンボボックスインスタンスを作成し、そのコンボボックスのテキストをイタリックおよび右揃えに設定します。

```
createClassObject(mx.controls.ComboBox, "my_cb", 1, {fontStyle:"italic",  
    textAlign:"right"});  
my_cb.addItem({data:1, label:"One"});
```



複数のプロパティを変更する場合や、複数のコンポーネントインスタンスのプロパティを変更する場合は、カスタムスタイルを作成できます。複数のプロパティを設定するカスタムスタイルを使用したコンポーネントインスタンスは、何度も setStyle() を呼び出したコンポーネントインスタンスよりも高速で表示されます。詳細については、[88 ページの「コンポーネントのグループに対するカスタムスタイルの設定」](#)を参照してください。

Halo テーマを使用する単一コンポーネントインスタンスのプロパティを設定または変更するには：

1. ステージでコンポーネントインスタンスを選択します。
2. プロパティインスペクタで、インスタンスに myComponent という名前を付けます。
3. [アクション] パネルを開いて [シーン 1] を選択し、[レイヤー 1: フレーム 1] を選択します。

4. インスタンスをオレンジ色にするには、次のコードを入力します。

```
myComponent.setStyle("themeColor", "haloOrange");
```

5. [制御]-[ムービープレビュー] を選択して変更を確認します。

特定のコンポーネントでサポートされるスタイルの一覧は、『ActionScript 2.0 コンポーネントリファレンスガイド』のそのコンポーネントの項目を参照してください。

コンポーネントインスタンスを作成し、ActionScript で複数のプロパティを同時に設定するには：

1. コンポーネントをライブラリにドラッグします。
2. [アクション] パネルを開いて [シーン 1] を選択し、[レイヤー 1: フレーム 1] を選択します。
3. コンポーネントのインスタンスを作成してプロパティを設定するために、次のシンタックスを入力します。

```
createClassObject(className, "instance_name", depth, {style:"setting",  
style:"setting"});
```

たとえば次の ActionScript では、ライブラリに Button コンポーネントがある場合に my_button というボタンインスタンスを深度 1 に作成し、テキストスタイルを紫色のイタリックに設定します。

```
createClassObject(mx.controls.Button, "my_button", 1, {label:"Hello",  
color:"0x9900CC", fontStyle:"italic"});
```

詳細については、「UIObject.createClassObject()」を参照してください。

4. [制御]-[ムービープレビュー] を選択して変更を確認します。

特定のコンポーネントでサポートされるスタイルの一覧は、『ActionScript 2.0 コンポーネントリファレンスガイド』のそのコンポーネントの項目を参照してください。

グローバルスタイルの設定

別のスタイル宣言が割り当てられているコンポーネントを除き、すべてのコンポーネントにはデフォルトでグローバルスタイル宣言が適用されます ([88 ページの「コンポーネントのグループに対するカスタムスタイルの設定」](#)を参照してください)。グローバルスタイル宣言は、Adobe Component Architecture のバージョン 2 で作成したすべての Flash コンポーネントに割り当てられます。_global オブジェクトは、CSSStyleDeclaration のインスタンスである style プロパティ (_global.style) を持ち、グローバルスタイル宣言として機能します。グローバルスタイル宣言でスタイルプロパティの値を変更すると、Flash ドキュメント内のすべてのコンポーネントにその変更が適用されます。

辞
典

一部のスタイルは、コンポーネントクラスの CSSStyleDeclaration インスタンスで設定されています (TextArea コンポーネントと TextInput コンポーネントの backgroundColor スタイルなどがこれにあたります)。スタイル値の決定に際して、クラススタイル宣言はグローバルスタイル宣言よりも優先されるので、グローバルスタイル宣言で backgroundColor を設定しても、TextArea コンポーネントや TextInput コンポーネントには反映されません。スタイルの優先順位の詳細については、[92 ページの「同じドキュメント内でのグローバル、カスタム、クラススタイルの使用」](#)を参照してください。コンポーネントクラスの CSSStyleDeclaration を編集する方法の詳細については、[89 ページの「コンポーネントクラスのスタイルの設定」](#)を参照してください。

グローバルスタイル宣言の1つまたは複数のプロパティを変更するには：

1. ドキュメントに少なくとも1つのコンポーネントインスタンスが含まれていることを確認します。
詳細については、[50 ページの「Flash ドキュメントへのコンポーネントの追加」](#)を参照してください。
2. タイムライン内で、コンポーネントが表示されているフレーム、または表示される前のフレームを選択します。
3. [アクション] パネルで、次のようなコードを使用して、グローバルスタイル宣言の中のプロパティを変更します。値を変更するプロパティのみを次のように列挙します。変更しないプロパティを指定する必要はありません。

```
_global.style.setStyle("color", 0xCC6699);  
_global.style.setStyle("themeColor", "haloBlue")  
_global.style.setStyle("fontSize",16);  
_global.style.setStyle("fontFamily" , "_serif");
```
4. [制御]-[ムービープレビュー] を選択して変更を確認します。

コンポーネントのグループに対するカスタムスタイルの設定

カスタムスタイル宣言を作成して、Flash ドキュメント内のコンポーネントのグループに固有のプロパティ群を指定できます。`_global` オブジェクトには、Flash ドキュメント全体に対するデフォルトのスタイル宣言を指定する `style` プロパティ (87 ページの「グローバルスタイルの設定」を参照) の他、使用可能なカスタム宣言のリストを示す `styles` プロパティがあります。スタイル宣言を `CSSStyleDeclaration` オブジェクトの新しいインスタンスとして作成する際、そのインスタンスにカスタムスタイル名を付けて `_global.styles` リストに追加できます。それからスタイルに対してプロパティや値を指定し、このカスタムスタイル名を、外観を統一したい各コンポーネントインスタンスに割り当てます。

スタイル名をコンポーネントインスタンスに割り当てても、そのコンポーネントでサポートされているスタイルプロパティしか適用されないことに注意する必要があります。各コンポーネントでサポートされるスタイルプロパティの一覧は、『ActionScript 2.0 コンポーネントリファレンスガイド』の各コンポーネントの項目を参照してください。

カスタムスタイルフォーマットは、次のシンタックスを使って変更します。

```
_global.styles.CustomStyleName.setStyle(propertyName, propertyValue);
```

カスタムスタイル設定は、クラススタイル設定、継承スタイル設定、およびグローバルスタイル設定よりも優先的に使われます。スタイルの優先順位の一覧は、92 ページの「同じドキュメント内でのグローバル、カスタム、クラススタイルの使用」を参照してください。

コンポーネントグループのカスタムスタイル宣言を作成するには：

1. コンポーネントを少なくとも1つ、ステージに追加します。

詳細については、50 ページの「Flash ドキュメントへのコンポーネントの追加」を参照してください。

以降の例では、a、b、c というインスタンス名を持つ3つのボタンコンポーネントを使用します。別のコンポーネントを使用する場合は、プロパティインスペクタでそれらにインスタンス名を付け、そのインスタンス名を手順8で使用します。

2. タイムライン内で、コンポーネントが表示されているフレーム、または表示される前のフレームを選択します。
3. [アクション] パネルを開きます。
4. 次の `import` ステートメントを追加して、`CSSStyleDeclaration` クラス内から新しいスタイル宣言を作成するためのコンストラクタにアクセスできるようにします。

```
import mx.styles.CSSStyleDeclaration;
```

5. 次のシンタックスを使用して、`CSSStyleDeclaration` オブジェクトのインスタンスを作成し、新規カスタムスタイル形式を定義します。

```
var new_style:Object = new CSSStyleDeclaration();
```


6. スタイル宣言の名前(たとえば "myStyle")を、カスタムスタイル宣言のリスト `_global.styles` に設定し、その新しいスタイル宣言のプロパティをすべて含んだオブジェクトを指定します。

```
_global.styles.myStyle = new_style;
```

7. `UIObject` クラスから継承した `setStyle()` メソッドを使用し、`new_style` オブジェクトにプロパティを追加します。ここで追加するプロパティが、カスタムスタイル宣言 `myStyle` に関連付けられます。

```
new_style.setStyle("fontFamily", "_serif");
new_style.setStyle("fontSize", 14);
new_style.setStyle("fontWeight", "bold");
new_style.setStyle("textDecoration", "underline");
new_style.setStyle("color", 0x666699);
```

8. 同じスクリプトペインで、次のシンタックスを使って、特定の3つのコンポーネントの `styleName` プロパティをカスタムスタイル宣言の名前に設定します。

```
a.setStyle("styleName", "myStyle");
b.setStyle("styleName", "myStyle");
c.setStyle("styleName", "myStyle");
```

`setStyle()` メソッドと `getStyle()` メソッドを使用して、宣言の `styleName` グローバルプロパティを通じてカスタムスタイル宣言のスタイルにアクセスすることもできます。たとえば、次のコードは、`myStyle` スタイル宣言で `backgroundColor` スタイルを設定します。

```
_global.styles.myStyle.setStyle("themeColor", "haloOrange");
```

ただし、手順5と6は、`new_style` インスタンスとスタイル宣言を関連付けています。これは次のコードに示すようにシンタックスを簡潔にするためです `new_style.setStyle("themeColor", "haloOrange")`。

`setStyle()` メソッドと `getStyle()` メソッドの詳細については、『**ActionScript 2.0** コンポーネントリファレンスガイド』の「`UIObject.setStyle()`」および「`UIObject.getStyle()`」を参照してください。

コンポーネントクラスのスタイルの設定

クラススタイル宣言は、コンポーネントのあらゆるクラス (`Button` や `CheckBox` など) に対して定義することができ、そのクラスの各インスタンスにデフォルトのスタイルを設定します。インスタンスを作成する前にスタイル宣言を作成する必要があります。`TextArea` や `TextInput` など一部のコンポーネントについては、`borderStyle` および `backgroundColor` プロパティを必ずカスタマイズするので、定義済みのクラススタイル宣言がデフォルトで用意されています。



クラスのスタイルシートを置き換える場合は、元のスタイルシートから必要なスタイルを追加してください。そうしないと、それらのスタイルは上書きされます。

次に示すコードでは、まず現在のテーマに `CheckBox` のスタイル宣言が既に含まれているかどうか確認し、含まれない場合は新しいスタイル宣言を作成します。次に、`setStyle()` メソッドを使用して `CheckBox` スタイル宣言のスタイルプロパティを定義します (ここでは "color" によって、すべてのチェックボックスのラベルテキストのカラーを青色に設定しています)。

```
if (_global.styles.CheckBox == undefined) {  
    _global.styles.CheckBox = new mx.styles.CSSStyleDeclaration();  
}  
_global.styles.CheckBox.setStyle("color", 0x0000FF);
```

`CheckBox` コンポーネントに設定できるスタイルプロパティの一覧表は、『[ActionScript 2.0 コンポーネントリファレンスガイド](#)』の「`CheckBox` コンポーネントでのスタイルの使用」を参照してください。

カスタムスタイル設定は、継承スタイル設定とグローバルスタイル設定よりも優先的に使用されます。スタイルの優先順位の一覧は、[92 ページの「同じドキュメント内でのグローバル、カスタム、クラススタイルの使用」](#)を参照してください。

コンテナの継承スタイルの設定

継承スタイルは、ドキュメントの `MovieClip` 階層にある親コンポーネントから値を継承するスタイルです。テキストまたはカラーのスタイルがインスタンス、カスタム、またはクラスレベルで設定されていない場合に、Flash は `MovieClip` 階層内でスタイル値を検索します。そのため、コンテナコンポーネントにスタイルを設定すると、これらのスタイル設定がコンテナ内のコンポーネントに継承されます。

継承スタイルには、次のようなスタイルがあります。

- `fontFamily`
- `fontSize`
- `fontStyle`
- `fontWeight`
- `textAlign`
- `textIndent`
- すべての単一値カラースタイル (たとえば、`themeColor` は継承スタイルですが、`alternatingRowColors` は継承スタイルではありません)

Style Manager は、スタイルが値を継承しているかどうかを Flash に通知します。実行時に他のスタイルを継承スタイルとして追加することもできます。詳細については、『Action Script 2.0 コンポーネントリファレンスガイド』の「StyleManager クラス」を参照してください。

✕
中

Flash コンポーネントにおけるスタイルの実装が HTML ページにおけるカスケーディングスタイルシート (CSS) と大きく異なる点として、CSS の inherit 値が Flash コンポーネントではサポートされていないことが挙げられます。スタイルが継承されるかどうかは、コンポーネントの設計によります。

継承されたスタイルは、グローバルスタイルよりも優先的に使われます。スタイルの優先順位の一覧は、[92 ページの「同じドキュメント内でのグローバル、カスタム、クラススタイルの使用」](#)を参照してください。

次の例は、Flash で提供されている Accordion コンポーネントで継承スタイルを使用する方法を示しています。

Accordion コンポーネントを作成し、各 Accordion ペイン内のコンポーネントにスタイルを継承するには：

1. 新しい FLA ファイルを開きます。
2. Accordion コンポーネントを [コンポーネント] パネルからステージにドラッグします。
3. プロパティインスペクタを使って、その Accordion コンポーネントの名前とサイズを設定します。ここでは、コンポーネントに **accordion** というインスタンス名を指定します。
4. TextInput コンポーネントと Button コンポーネントを [コンポーネント] パネルからライブラリにドラッグします。

コンポーネントをライブラリにドラッグすると、そのコンポーネントを実行時にスクリプトで使用できるようになります。

5. タイムラインの 1 番目のフレームに次の ActionScript を追加します。

```
var section1 = accordion.createChild(mx.core.View, "section1", {label: "First  
Section"});  
var section2 = accordion.createChild(mx.core.View, "section2", {label:  
    "Second Section"});  
  
var input1 = section1.createChild(mx.controls.TextInput, "input1");  
var button1 = section1.createChild(mx.controls.Button, "button1");  
  
input1.text = "Text Input";  
button1.label = "Button";  
button1.move(0, input1.height + 10);  
  
var input2 = section2.createChild(mx.controls.TextInput, "input2");  
var button2 = section2.createChild(mx.controls.Button, "button2");  
  
input2.text = "Text Input";
```

```
button2.label = "Button";  
button2.move(0, input2.height + 10);
```

上のコードでは、Accordion コンポーネントに 2 つの子を追加し、それぞれの子に TextInput コントロールおよび Button コントロールをロードします。この例では、これらのコントロールを使用してスタイルの継承を解説します。

6. [制御]-[ムービープレビュー]を選択して、スタイル継承を追加する前のドキュメントを確認します。
7. 次の `ActionScript` を、1 番目のフレームでスクリプトの末尾に追加します。

```
accordion.setStyle("fontStyle", "italic");
```

8. [制御]-[ムービープレビュー] を選択して変更を確認します。

Accordion コンポーネントの `fontStyle` 設定は、Accordion のテキストのみでなく、Accordion コンポーネント内の TextInput コンポーネントや Button コンポーネントに関連付けられたテキストにも作用します。

同じドキュメント内でのグローバル、カスタム、クラススタイルの使用

ドキュメント内の 1 か所でしかスタイルが定義されていなければ、Flash はその定義を使用してプロパティの値を特定します。ただし、1 つの Flash ドキュメント内で、コンポーネントインスタンスに直接設定されたスタイルプロパティ、カスタムスタイル宣言、デフォルトのクラススタイル宣言、継承スタイル、グローバルスタイル宣言などのさまざまなスタイル設定を使用できます。このため、Flash は特定の順序でこれらの定義を確認し、プロパティの値を決定します。

Flash は値が見つかるまで次の順序でスタイルを調べていきます。

1. コンポーネントインスタンスのスタイルプロパティを探します。
2. インスタンスの `styleName` プロパティをチェックして、そこにカスタムスタイル宣言が割り当てられているか調べます。
3. デフォルトクラススタイル宣言でそのプロパティを探します。
4. 継承スタイルに含まれるスタイルの場合は、親階層から継承される値を探します。
5. グローバルスタイル宣言でそのスタイルを探します。
6. 検索の結果そのプロパティが定義されていない場合は、値として `undefined` が使用されます。

カースタイルプロパティについて

カースタイルプロパティの動作は、その他のプロパティとは異なります。すべてのカラープロパティには、`backgroundColor`、`disabledColor`、`color` のように、"Color" で終わる名前が付いています。カースタイルプロパティを変更すると、そのインスタンスと、該当するすべての子インスタンスにおいてすぐにカラーが変化します。他のスタイルプロパティでは、変更を加えると、再描画が必要であるというマークがオブジェクトに付けられ、次のフレームでその変更が反映されます。

カースタイルプロパティの値は、数値、文字列、オブジェクトのいずれかです。数値で指定する場合には、16 進の RGB 値 (0xRRGGBB) を使用します。文字列で指定する場合には、カラー名を使用します。

カラー名は、一般的に使われるカラーを示す文字列です。Style Manager を使うと、新しいカラー名を追加できます (『ActionScript 2.0 コンポーネントリファレンスガイド』の StyleManager クラスを参照してください)。次の表は、デフォルトのカラー名の一覧です。

カラー名	値
black	0x000000
white	0xFFFFFFFF
red	0xFF0000
green	0x00FF00
blue	0x0000FF
magenta	0xFF00FF
yellow	0xFFFF00
cyan	0x00FFFF
haloGreen	0x80FF4D
haloBlue	0x2BF5F5
haloOrange	0xFFC200

×
!!

定義されていないカラー名を使用すると、コンポーネントが正しく描画されない可能性があります。

独自のカラー名を作成する際には、ActionScript 識別子として有効であればどのような名前でも使用できます ("WindowText"、"ButtonText" など)。Style Manager を使って新しいカラーを定義するには、次のように指定します。

```
mx.styles.StyleManager.registerColorName("special_blue", 0x0066ff);
```

大部分のコンポーネントでは、オブジェクトをカラースタイルプロパティの値として扱うことはできません。ただし、特定のコンポーネントでは、グラデーションやカラーの組み合わせを表すカラーオブジェクトを扱えます。詳細については、『**ActionScript 2.0** コンポーネントリファレンスガイド』の各コンポーネントの「スタイルの使用」セクションを参照してください。

クラススタイル宣言とカラー名を使用すると、スクリーン上のテキストやシンボルのカラーを簡単に制御できます。たとえば、**Microsoft Windows** にあるような画面デザイン設定画面を作成するには、**ButtonText** や **WindowText** のようなカラー名と、**Button**、**CheckBox**、**Window** のようなクラススタイル宣言を定義する方法が考えられます。

✕
#

コンポーネントによっては、カラーの配列を設定するスタイルプロパティ (alternatingRowColors など) を持つものがあります。これらのスタイルをカラー名として設定することはできません。RGB を表す数値の配列で設定する必要があります。

コンポーネントアニメーションのカスタマイズ

Accordion、**ComboBox**、**Tree** などの一部のコンポーネントでは、**Accordion** コンポーネントの子の切り替え、**ComboBox** ドロップダウンリストの展開、**Tree** フォルダの展開または折り畳みなどを行う場合に、アニメーションを使用してコンポーネント状態のトランジションを表示できます。また、コンポーネントでは、行やリストなどのアイテムの選択や選択解除に関連するアニメーションも表示できます。

これらのアニメーションの機能を制御するには、次のスタイルを使用します。

アニメーションのスタイル	説明
openDuration	Accordion 、 ComboBox 、および Tree コンポーネントでのオープンイメージのトランジション時間です (ミリ秒単位)。デフォルト値は 250 です。
openEasing	Accordion 、 ComboBox 、および Tree コンポーネント内のアニメーション状態を制御するトゥイーン関数に対する参照です。デフォルトの式としては、サインインとサインアウトの式が使われます。
popupDuration	Menu コンポーネントでメニューが開く際のトランジションの継続時間です (ミリ秒単位)。デフォルト値は 150 です。ただし、アニメーションでは、必ずデフォルトのサインインとサインアウトの式が使用されます。
selectionDuration	ComboBox 、 DataGrid 、 List 、および Tree コンポーネントで通常状態と選択状態の間のトランジションが継続する長さです (ミリ秒単位)。デフォルト値は 200 です。
selectionEasing	ComboBox 、 DataGrid 、 List 、および Tree コンポーネント内の選択アニメーションを制御するトゥイーン関数に対する参照です。このスタイルは、通常状態から選択状態へのトランジションにのみ適用されます。デフォルトの式としては、サインインとサインアウトの式が使われます。

mx.transitions.easing パッケージには、イージングを制御する次の 6 つのクラスが用意されています。

イージングクラス	説明
Back	トランジション範囲の片側または両側を一度超えて拡張し、軽いオーバーフロー効果が発生させます。
Bounce	トランジション範囲を超えずに、この範囲の片側または両側でバウンス効果が発生させます。バウンス回数は長さに関係し、長いほどバウンス回数が多くなります。
Elastic	トランジション範囲の片側または両側で外部に伸縮効果が発生させます。伸縮度は、長さの影響を受けません。
None	効果、減速、または加速を設定せずに、最初から最後まで等速で実行されます。通常、このようなトランジションを線状トランジションと呼びます。
Regular	加速効果や減速効果を実現するために、一端または両端で実行速度を減速します。
Strong	一端または両端で実行速度を大幅に減速します。この効果は、Regular と似ていますが、より大きい効果があります。

mx.transitions.easing パッケージの各クラスでは、次の 3 種類のイージングメソッドを使用できます。

イージングメソッド	説明
easeIn	トランジションの最初にイージング効果が発生させます。
easeOut	トランジションの最後にイージング効果が発生させます。
easeInOut	トランジションの最初と最後にイージング効果が発生させます。

イージングメソッドはイージングクラスの静的メソッドなので、イージングクラスをインスタンス化する必要はありません。このメソッドは、次の例のように `setStyle()` 呼び出しで使用されます。

```
import mx.transitions.easing.*;
trace("_global.styles.Accordion = " + _global.styles.Accordion);
_global.styles.Accordion.setStyle("openDuration", 1500);
_global.styles.Accordion.setStyle("openEasing", Bounce.easeOut);
```

✕

前

前に説明したイージングクラスでは、すべてのトランジションで使用できるデフォルトの式はありません。イージングメソッドを指定した後にコンポーネントでデフォルトのイージングメソッドを使用するには、`setStyle("openEasing", null)` を呼び出します。

詳細については、『ActionScript 2.0 コンポーネントリファレンスガイド』の「コンポーネントへのイージングメソッドの適用」を参照してください。

スタイルプロパティの値の取得

スタイルプロパティの値を取得するには、『ActionScript 2.0 コンポーネントリファレンスガイド』の「UIObject.getStyle()」を使用します。UIObject のサブクラスにあたるコンポーネント (Media コンポーネントを除くすべてのバージョン 2 コンポーネント) は、いずれも getStyle() メソッドを継承しています。したがって、setStyle() をすべてのコンポーネントインスタンスから呼び出せるのと同様に、getStyle() もすべてのコンポーネントインスタンスから呼び出せます。

次のコードでは、fontSize スタイルの値を取得し、それを変数 oldFontSize に割り当てます。

```
var myCheckBox:mx.controls.CheckBox;  
var oldFontSize:Number  
  
oldFontSize = myCheckBox.getStyle("fontSize");  
trace(oldFontSize);
```

コンポーネントへのスキンの適用について

スキンは、コンポーネントの外観を表示するためのムービークリップのシンボルです。多くのスキンには、コンポーネントの外観を構成するシェイプが含まれます。一部のスキンには、ドキュメント内でコンポーネントを描画する ActionScript のコードだけが含まれます。

バージョン 2 コンポーネントはコンパイルされたクリップなので、ライブラリ内でアセットを見ることはできません。ただし、Flash をインストールすると、全コンポーネントのスキンを含む FLA ファイルもインストールされます。これらの FLA ファイルは "テーマ" と呼ばれます。テーマごとに外観と動作は異なりますが、どのテーマに含まれるスキンでも、シンボル名とリンケージ識別子は共通しています。これにより、ドキュメントのステージにテーマをドラッグしてドキュメントの外観を変更できるようになります。テーマの FLA ファイルは、コンポーネントのスキンの編集にも使用します。スキンは、各テーマ FLA ファイルの [ライブラリ] パネルの "Themes" フォルダにあります。テーマの詳細については、[108 ページの「テーマについて」](#)を参照してください。

1つのコンポーネントは多数のスキンで構成されています。たとえば、ScrollBar サブコンポーネントの下矢印は、ScrollDownArrowDisabled、ScrollDownArrowDown、ScrollDownArrowOver、および ScrollDownArrowUp の 4 つのスキンで構成されています。ScrollBar 全体では、13 種類のスキンシンボルを使用します。

一部のコンポーネントは同じスキンを共有しています。たとえば、スクロールバーを使うコンポーネント (ComboBox、List、ScrollPane など) は "ScrollBar Skins" フォルダ内のスキンを共有します。既存のスキンを編集して新しいスキンを作成すると、コンポーネントの外観を変更できます。

各コンポーネントクラスを定義する AS ファイルには、そのコンポーネント用のスキンをロードするコードが含まれています。各コンポーネントのスキンは、スキンシンボルのリンケージ識別子に割り当てられたスキンプロパティに対応します。たとえば、**ScrollBar** コンポーネントの押された状態の下矢印には、`downArrowDownName` というスキンプロパティ名が付いています。`downArrowDownName` プロパティのデフォルト値は `"ScrollDownArrowDown"` で、これはテーマ FLA ファイル内のスキンシンボルのリンケージ識別子です。スキンシンボルを編集して既存のリンケージ識別子をそのまま使用すると、既存のスキンを編集し、そのスキンを使用するすべてのコンポーネントに適用できます。コンポーネントインスタンスにスキンプロパティを設定すると、新しいスキンを作成してそのコンポーネントインスタンスに適用できます。スキンプロパティを変更するには、コンポーネントの AS ファイルを編集する必要はなく、ドキュメントでコンポーネントを作成する際に、コンポーネントのコンストラクタ関数にスキンプロパティの値を渡します。

各コンポーネントのスキンプロパティの詳細については、「コンポーネント辞書」の各コンポーネントの項を参照してください。たとえば、**Button** コンポーネントのスキンプロパティは、[\[ActionScript 2.0 コンポーネントリファレンスガイド\]-\[Button コンポーネント\]-\[Button コンポーネントのカスタマイズ\]-\[Button コンポーネントでのスキンの使用\]](#)に記載されています。

コンポーネントにスキンを適用するには、目的に応じて次のいずれかの方法を使用します。最初の方法が最も簡単であり、下にいくほど難度が高くなります。

- 1 つのドキュメントに含まれる特定のコンポーネントのすべてのインスタンスに関連付けられたスキンを変更するには、各スキンエレメントをコピーして変更します。詳細については、[48 ページの「ドキュメントのコンポーネントスキンの編集」](#)を参照してください。
この方法はスクリプトを記述する必要がないため、初心者にお勧めします。
- ドキュメント内のすべてのスキンを、共通の外観を持つ各種のコンポーネントを含んだ新しいセットで置き換えるには、テーマを適用します。詳細については、[108 ページの「テーマについて」](#)を参照してください。
すべてのコンポーネントと複数のドキュメントで外観と操作性を整合させる場合は、このスキンの適用方法をお勧めします。
- スキンエレメントのカラーをスタイルプロパティにリンクするには、スキンに `ActionScript` コードを追加し、カラースキンエレメントとして登録します。詳細については、[100 ページの「スキンカラーとスタイルのリンク」](#)を参照してください。
- 同じコンポーネントの複数のインスタンスにそれぞれ異なるスキンを使用するには、新しいスキンを作成して、スキンのプロパティを設定します。詳細については、[99 ページの「コンポーネントスキンの新規作成」](#)および [102 ページの「コンポーネントへの新しいスキンの適用」](#)を参照してください。
- サブコンポーネントのスキン (`List` コンポーネント内のスクロールバーなど) を変更するには、そのコンポーネントをサブクラス化します。詳細については、[103 ページの「サブコンポーネントへの新しいスキンの適用」](#)を参照してください。

- メインのコンポーネントから直接アクセスできないサブコンポーネント (ComboBox コンポーネント内の List コンポーネントなど) のスキンを変更するには、プロトタイプ内のスキンのプロパティを変更します。詳細については、[107 ページの「サブコンポーネントでのスキンプロパティの変更」](#)を参照してください。

ドキュメントのコンポーネントスキンの編集

1つのドキュメントに含まれる特定のコンポーネントのすべてのインスタンスに関連付けられたスキンを編集するには、テーマからドキュメントにスキンシンボルをコピーして、グラフィックを自由に編集します。

次の手順は、新しいテーマを作成して適用する場合と非常に似ています ([108 ページの「テーマについて」](#)を参照)。主な相違点は、この手順では既に使用されているテーマから1つのドキュメントにシンボルを直接コピーし、使用可能なスキンのごく一部だけを編集する点です。この方法は、1つのドキュメントのみを編集する場合、および、ごく少数のコンポーネントのスキンを編集する場合に適しています。編集したスキンを複数のドキュメントで共有する場合や、複数のコンポーネントに変更を適用する場合は、新しいテーマを作成した方が簡単に編集できることがあります。

スキンの高度な話題については、オンラインの Adobe デベロッパーセンターにある記事を参照してください (www.adobe.com/jp/devnet/flash/articles/skinning_fl8.html)。

ドキュメントのコンポーネントスキンを編集するには：

1. ドキュメントに Sample テーマを既に適用している場合は、手順 5 に進みます。
2. [ファイル]-[読み込み]-[外部ライブラリを開く]を選択して、"SampleTheme fla" ファイルを選択します。

このファイルは、アプリケーションレベル設定フォルダにあります。オペレーティングシステム別の正確な場所については、[108 ページの「テーマについて」](#)を参照してください。

3. テーマの [ライブラリ] パネルで "Flash UI Components 2/Themes/MMDefault" を選択し、ドキュメントに含まれるいずれかのコンポーネントの "Assets" フォルダをドキュメントのライブラリにドラッグします。

たとえば、"RadioButton Assets" フォルダを "ThemeApply fla" ライブラリにドラッグします。


4. 個別のコンポーネントの "Assets" フォルダをライブラリにドラッグした場合は、各コンポーネントの Assets シンボルが [最初のフレームに書き出し] に設定されていることを確認します。

たとえば、RadioButton コンポーネントの "Assets" フォルダは "RadioButton Assets" で、RadioButtonAssets というシンボルが格納されています。このシンボルには、すべての個別のアセットシンボルが含まれています。RadioButtonAssets シンボルに [最初のフレームに書き出し] を設定すると、すべての個別のアセットも最初のフレームに書き出されます。

5. 変更したいスキンのシンボルをダブルクリックし、シンボル編集モードで開きます。

たとえば、States/RadioButtonFalseDisabled シンボルを開きます。

6. シンボルを変更するか、グラフィックを削除して新しいグラフィックを作成します。
必要に応じて [表示]-[ズームイン] を選択し、表示比率を上げます。スキンが正常に表示されるように、スキンを編集する際には必ず基準点を維持してください。編集したすべてのシンボルについて、左上隅が (0,0) になるようにする必要があります。
たとえば、内側の円を明るい灰色に変更します。
7. スキンシンボルの編集が完了したら、ステージの一番上の編集バーの左側にある [戻る] ボタンをクリックし、ドキュメント編集モードに戻ります。
8. 変更を加えるすべてのスキンに対して、手順 5～7 を実行します。

	ステージでのコンポーネントのライブプレビューには、編集したスキンは反映されません。
---	---

9. [制御]-[ムービープレビュー] を選択します。
この例では、ステージに RadioButton インスタンスがあることを確認し、[アクション] パネルで enabled プロパティを false に設定して、新しく無効にした RadioButton コンポーネントの外観を表示します。

コンポーネントスキンの新規作成

コンポーネントの1つのインスタンスに特定のスキンを使用し、他のインスタンスには別のスキンを使用する場合には、テーマの FLA ファイルを開き、新しいスキンシンボルを作成する必要があります。コンポーネントは、異なるインスタンスに異なるスキンを容易に使用できるよう設計されています。

新しいスキンを作成するには：

1. [ファイル]-[開く] を選択し、テンプレートとして使用したいテーマの FLA ファイルを開きます。
2. [ファイル]-[名前を付けて保存] を選択し、"MyTheme fla" などの一意の名前を指定します。
3. 編集するスキン (ここでは RadioTrueUp) を選択します。
スキンは、"Themes/MMDefault/Component Assets" フォルダ (この例では "Themes/MMDefault/RadioButton Assets/States") にあります。
4. [ライブラリ] パネルのオプションメニューから (または、シンボルを右クリックして) [複製] を選択し、シンボルに "MyRadioTrueUp" などの一意の名前を付けます。
5. [シンボルプロパティ] ダイアログボックスで [詳細] をクリックし、[ActionScript に書き出し] を選択します。
シンボル名にマッチするリンケージ識別子が自動的に入力されます。
6. ライブラリで新しいスキンをダブルクリックし、シンボルの編集モードで開きます。

7. ムービークリップを修正または削除し、新規ムービークリップを作成します。
必要に応じて [表示]-[ズームイン] を選択し、表示比率を上げます。スキンが正常に表示されるように、スキンを編集する際には必ず基準点を維持してください。編集したすべてのシンボルについて、左上隅が (0,0) になるようにする必要があります。
8. スキンシンボルの編集が完了したら、ステージの一番上の編集バーの左側にある [戻る] ボタンをクリックし、ドキュメント編集モードに戻ります。
9. [ファイル]-[保存] を選択し、"MyTheme.fla" ファイルは開いたままにしておきます。新しいドキュメントを作成して、そのコンポーネントに編集したスキンを適用します。
詳細については、[102 ページの「コンポーネントへの新しいスキンの適用」](#)、[103 ページの「サブコンポーネントへの新しいスキンの適用」](#)、または [107 ページの「サブコンポーネントでのスキンプロパティの変更」](#) を参照してください。



ライブプレビューを使用してステージにコンポーネントを表示する場合、コンポーネントスキンに加えた変更は表示されません。

スキンカラーとスタイルのリンク

バージョン 2 のコンポーネントフレームワークでは、スキンを使用するコンポーネントに設定されたスタイルに、スキンエレメント内の視覚的なアセットを簡単にリンクできます。ムービークリップインスタンスまたはスキンエレメント全体を 1 つのスタイルに登録するには、スキンのタイムラインに `ActionScript` コードを追加して、

`mx.skins.ColoredSkinElement.setColorStyle(targetMovieClip, styleName)` を呼び出します。

スタイルプロパティにスキンをリンクするには：

1. ドキュメントに Sample テーマを既に適用している場合は、手順 5 に進みます。
2. [ファイル]-[読み込み]-[外部ライブラリを開く] を選択して、"SampleTheme.fla" ファイルを選択します。

このファイルは、アプリケーションレベル設定フォルダにあります。オペレーティングシステム別の正確な場所については、[108 ページの「テーマについて」](#) を参照してください。

3. テーマの [ライブラリ] パネルで "Flash UI Components 2/Themes/MMDefault" を選択し、ドキュメントに含まれるいずれかのコンポーネントの "Assets" フォルダをドキュメントのライブラリにドラッグします。

たとえば、"RadioButton Assets" フォルダをターゲットライブラリにドラッグします。

4. 個別のコンポーネントの "Assets" フォルダをライブラリにドラッグした場合は、各コンポーネントの Assets シンボルが [最初のフレームに書き出し] に設定されていることを確認します。

たとえば、RadioButton コンポーネントの "Assets" フォルダは "RadioButton Assets" で、RadioButtonAssets というシンボルが格納されています。このシンボルには、すべての個別のアセットシンボルが含まれています。RadioButtonAssets シンボルに [最初のフレームに書き出し] を設定すると、すべての個別のアセットも最初のフレームに書き出されます。

5. 変更したいスキンのシンボルをダブルクリックし、シンボル編集モードで開きます。

たとえば、States/RadioFalseDisabled シンボルを開きます。

6. カラーにするエレメントがムービークリップインスタンスではなく、グラフィックシンボルである場合は、[修正]-[シンボルに変換] を選択してムービークリップインスタンスに変換します。

この例では、RadioShape1 グラフィックシンボルのインスタンスである中央のグラフィックをムービークリップに変更し、Inner Circle という名前を付けます。ActionScript については、[書き出し] を選択する必要はありません。

編集中のコンポーネントアセットの "Elements" フォルダに、新規作成したムービークリップシンボルを移動することをお勧めします。ただし、この操作は必須ではありません。

7. 前の手順でグラフィックシンボルをムービークリップインスタンスに変換した場合は、そのインスタンスに名前を付けて ActionScript で使用できるようにします。

この例では、インスタンス名として innerCircle を指定します。

8. ActionScript コードを追加して、そのスキンエレメントまたは含まれるムービークリップインスタンスをカラースキンエレメントとして登録します。

たとえば、次のコードをスキンエレメントのタイムラインに追加します。

```
mx.skins.ColoredSkinElement.setColorStyle(innerCircle,
    "symbolBackgroundDisabledColor");
```

この例では、Sample スタイルの既存のスタイル名に既に対応しているカラーを使用します。できる限り、カスケーディングスタイルシートの公式規格または Halo テーマと Sample テーマで提供されるスタイルに対応したスタイル名を使用してください。

9. 変更を加えるすべてのスキンに対して、手順 5～8 を実行します。

この例では、RadioTrueDisabled スキンについてこれらの手順を繰り返します。ただし、既存のグラフィックをムービークリップに変換する代わりに、既存の Inner Circle シンボルを RadioTrueDisabled スキンエレメントにドラッグします。

10. スキンシンボルの編集が完了したら、ステージの一番上の編集バーの左側にある [戻る] ボタンをクリックし、ドキュメント編集モードに戻ります。

11. コンポーネントのインスタンスをステージにドラッグします。

この例では、2 つの RadioButton コンポーネントをステージにドラッグし、1 つのコンポーネントを選択状態に設定します。ActionScript を使用して両方のコンポーネントを無効にし、変更を確認します。

12. ActionScript コードをドキュメントに追加し、コンポーネントインスタンスまたはグローバルレベルで新しいスタイルプロパティを設定します。

この例では、次のようにグローバルレベルでプロパティを設定します。

```
_global.style.setStyle("symbolBackgroundDisabledColor", 0xD9D9D9);
```

13. [制御]-[ムービープレビュー] を選択します。

コンポーネントへの新しいスキンの適用

新しいスキンを作成したら、ドキュメント内のコンポーネントにそのスキンを適用します。createClassObject() メソッドを使ってコンポーネントインスタンスを動的に作成するか、コンポーネントインスタンスを手動でステージに配置します。コンポーネントインスタンスにスキンを適用する方法は2種類あり、コンポーネントをドキュメントに追加する方法に応じて異なります。

動的に作成したコンポーネントに、新しいスキンを適用するには：

1. [ファイル]-[新規] を選択し、新規 Flash ドキュメントを作成します。
2. [ファイル]-[保存] を選択して、ファイルに "DynamicSkinning fla" のような一意の名前を付け保存します。
3. コンポーネント (スキンを編集したコンポーネントを含む。この例では RadioButton) を [コンポーネント] パネルからライブラリにドラッグします。

これで、ドキュメントのライブラリにシンボルが追加されますが、シンボルはステージ上には表示されません。

4. MyRadioTrueUp などのカスタマイズしたシンボルを "MyTheme fla" から "DynamicSkinning fla" のライブラリにドラッグします。

これで、ドキュメントのライブラリにシンボルが追加されますが、シンボルはステージ上には表示されません。

5. [アクション] パネルを開いて、次のコードをフレーム 1 に入力します。

```
import mx.controls.RadioButton;  
createClassObject(RadioButton, "myRadio", 0, {trueUpIcon:"MyRadioTrueUp",  
label: "My Radio Button"});
```

6. [制御]-[ムービープレビュー] を選択します。

ステージにコンポーネントを手作業で追加して新しいスキンを適用するには：

1. [ファイル]-[新規] を選択し、新規 Flash ドキュメントを作成します。
2. [ファイル]-[保存] を選択して、ファイルに "ManualSkinning fla" のような一意の名前を付け保存します。
3. コンポーネント (スキンを編集したコンポーネントを含む。この例では RadioButton) を [コンポーネント] パネルからステージにドラッグします。

4. MyRadioTrueUp などのカスタマイズしたシンボルを "MyTheme fla" から "ManualSkinning fla" のライブラリにドラッグします。

これで、ドキュメントのライブラリにシンボルが追加されますが、シンボルはステージ上には表示されません。

5. ステージで RadioButton コンポーネントを選択し、[アクション] パネルを開きます。
6. RadioButton インスタンスに次のコードを追加します。

```
onClipEvent(initialize){  
    trueUpIcon = "MyRadioTrueUp";  
}
```

7. [制御]-[ムービープレビュー] を選択します。

サブコンポーネントへの新しいスキンの適用

コンポーネント内のサブコンポーネントのスキンを変更する場合、そのスキンのプロパティに直接アクセスすることはできません (たとえば、List コンポーネント内のスクロールバーのスキンを直接変更する方法はありません)。この場合、スクロールバーのスキンにアクセスするには次に示すコードを使用します。このコードを実行した後に作成するすべてのスクロールバーにも、新しいスキンが適用されます。

サブコンポーネントで構成されるコンポーネントに関して、どのようなサブコンポーネントが含まれているかについては、『ActionScript 2.0 コンポーネントリファレンスガイド』の各コンポーネントのセクションを参照してください。

サブコンポーネントにスキンを適用するには：

1. 99 ページの「コンポーネントスキンの新規作成」と同じ手順に従いますが、今回はスクロールバーのスキンを編集します。この例では ScrollDownArrowDown スキン編集し、これに "MyScrollDownArrowDown" という新しい名前を付けます。
2. [ファイル]-[新規] を選択し、新規 Flash ドキュメントを作成します。
3. [ファイル]-[保存] を選択して、ファイルに "SubcomponentProject fla" のような一意の名前を付け保存します。
4. List コンポーネントを [コンポーネント] パネルからライブラリにドラッグします。
これで、コンポーネントが [ライブラリ] パネルに追加されますが、コンポーネントはドキュメント内には表示されません。
5. MyScrollDownArrowDown などの編集済みシンボルを "MyTheme fla" から "SubcomponentProject fla" のライブラリにドラッグします。
これで、シンボルが [ライブラリ] パネルに追加されますが、ドキュメント内には表示されません。

6. 次のいずれかの操作を行います。

- コンポーネント内のすべてのスクロールバーを変更する場合は、タイムラインのフレーム1にある [アクション] パネルに次のコードを入力します。

```
import mx.controls.List;
import mx.controls.scrollClasses.ScrollBar;
ScrollBar.prototype.downArrowDownName = "MyScrollDownArrowDown";
```

これで、フレーム1に次のコードを入力して動的にリストを作成できるようになります。

```
createClassObject(List, "myListBox", 0, {dataProvider: ["AL","AR","AZ",
    "CA","HI","ID", "KA","LA","MA"]});
```

または、List コンポーネントをライブラリからステージヘドラッグします。

- コンポーネント内の特定のスクロールバーを変更する場合は、タイムラインのフレーム1にある [アクション] パネルに次のコードを入力します。

```
import mx.controls.List
import mx.controls.scrollClasses.ScrollBar
var oldName = ScrollBar.prototype.downArrowDownName;
ScrollBar.prototype.downArrowDownName = "MyScrollDownArrowDown";
createClassObject(List, "myList1", 0, {dataProvider: ["AL","AR","AZ",
    "CA","HI","ID", "KA","LA","MA"]});
myList1.redraw(true);
ScrollBar.prototype.downArrowDownName = oldName;
```



スクロールバーが表示されるようにするには、十分な数のデータを追加するか、vScrollPolicy プロパティを true に設定します。

7. [制御]-[ムービープレビュー] を選択します。

スキンシンボルの #initclip セクションにある、サブコンポーネントの prototype オブジェクトでスキンプロパティを設定すれば、サブコンポーネントのスキンをドキュメント内のすべてのコンポーネントに適用することもできます。

#initclip を使用して、ドキュメント内のすべてのコンポーネントに編集したスキンを適用するには：

1. 99 ページの「コンポーネントスキンの新規作成」と同じ手順に従いますが、今回はスクロールバーのスキンを編集します。この例では ScrollDownArrowDown スキンを編集し、これに "MyScrollDownArrowDown" という新しい名前を付けます。
2. [ファイル]-[新規] を選択し、新規 Flash ドキュメントを作成します。これを、"SkinsInitExample.fla" などの一意の名前を付けて保存します。
3. 編集済みテーマのライブラリ例のライブラリから MyScrollDownArrowDown シンボルを選択し、"SkinsInitExample.fla" のライブラリにドラッグします。
これにより、ステージに表示することなくシンボルをライブラリに追加できます。

4. "SkinsInitExample.fla" ライブラリで MyScrollDownArrowDown を選択し、[ライブラリ] オプションメニューから [リンケージ] を選択します。
5. [ActionScript に書き出し] チェックボックスをオンにします。[OK] をクリックします。
[最初のフレームに書き出し] は、通常は自動的にオンになりますが、オフになっている場合はオンに切り替えてください。
6. ライブラリで MyScrollDownArrowDown をダブルクリックし、シンボル編集モードで開きます。
7. MyScrollDownArrowDown シンボルのフレーム 1 に次のコードを入力します。

```
#initclip 10
    import mx.controls.scrollClasses.ScrollBar;
    ScrollBar.prototype.downArrowDownName = "MyScrollDownArrowDown";
#endinitclip
```

8. List コンポーネントをドキュメントに追加するには、次のいずれかの操作を行います。
 - List コンポーネントを [コンポーネント] パネルからステージまでドラッグします。縦のスクロールバーが表示されるようにするには、十分な数のラベルパラメータを入力します。
 - List コンポーネントを [コンポーネント] パネルからライブラリにドラッグします。SkinsInitExample.fla のメインタイムラインのフレーム 1 に次のコードを入力します。

```
createClassObject(mx.controls.List, "myListBox1", 0, {dataProvider:
    ["AL", "AR", "AZ", "CA", "HI", "ID", "KA", "LA", "MA"]});
```



縦のスクロールバーが表示されるようにするには、十分な数のデータを追加するか、vScrollPolicy を true に設定します。

次の例では、既にステージ上にあるコンポーネントにスキンを適用する方法を紹介します。この例では、List のスクロールバーにのみスキンを適用します。TextArea や ScrollPane のスクロールバーにはスキンは適用されません。

#initclip を使用して、ドキュメント内の特定のコンポーネントに編集したスキンを適用するには：

1. [98 ページの「ドキュメントのコンポーネントスキンの編集」](#)と同じ手順に従いますが、今回はスクロールバーのスキンを編集します。この例では ScrollDownArrowDown スキンを編集し、これに "MyScrollDownArrowDown" という新しい名前を付けます。
2. [ファイル]-[新規] を選択し、Flash ドキュメントを作成します。
3. [ファイル]-[保存] を選択して、ファイルに "MyVScrollTest.fla" のような一意の名前を付け保存します。
4. テーマライブラリから MyVScrollTest.fla ライブラリへ MyScrollDownArrowDown をドラッグします。
5. [挿入]-[新規シンボル] を選択し、"MyVScrollBar" のような一意の名前を付けます。

6. [ActionScript に書き出し] チェックボックスをオンにします。[OK] をクリックします。
[最初のフレームに書き出し] は、通常は自動的にオンになりますが、オフになっている場合はオンに切り替えてください。
7. MyVScrollBar シンボルのフレーム1に次のコードを入力します。

```
#initclip 10
    import MyVScrollBar
    Object.registerClass("VScrollBar", MyVScrollBar);
#endinitclip
```
8. List コンポーネントを [コンポーネント] パネルからステージまでドラッグします。
9. プロパティインスペクタで、縦のスクロールバーを表示するために必要なすべての Label パラメータを入力します。
10. [ファイル]-[保存] を選択します。
11. [ファイル]-[新規] を選択し、新規 ActionScript ファイルを作成します。
12. 次のコードを入力します。

```
import mx.controls.VScrollBar
import mx.controls.List
class MyVScrollBar extends VScrollBar{
    function init():Void{
        if (_parent instanceof List){
            downArrowDownName = "MyScrollDownArrowDown";
        }
        super.init();
    }
}
```
13. [ファイル]-[保存] を選択し、"MyVScrollBar.as" という名前でファイルを保存します。
14. ステージの空白部分をクリックし、プロパティインスペクタで [パブリッシュ] の横の [設定] ボタンをクリックします。
15. [ActionScript のバージョン] の横にある [設定] ボタンをクリックします。
16. [新規パスの追加](+) ボタンをクリックしてクラスパスを追加し、[ターゲット] ボタンを選択して、ハードディスク上の "MyVScrollBar.as" ファイルのある場所に移動します。
17. [制御]-[ムービープレビュー] を選択します。

サブコンポーネントでのスキンプロパティの変更

コンポーネントがスキン変数を直接サポートしていない場合は、そのコンポーネントのサブクラスを作成してスキンを変更することができます。たとえば、ComboBox コンポーネントでは、直接ドロップダウンリストにスキンを適用できません。これは、ComboBox コンポーネントでは List コンポーネントをドロップダウンリストとして使用しているためです。

サブコンポーネントで構成されるコンポーネントに関して、どのようなサブコンポーネントが含まれているかについては、『ActionScript 2.0 コンポーネントリファレンスガイド』の各コンポーネントのセクションを参照してください。

サブコンポーネントにスキンを適用するには：

1. 98 ページの「ドキュメントのコンポーネントスキンの編集」と同じ手順に従いますが、今回はスクロールバーのスキンを編集します。この例では ScrollDownArrowDown スキンを編集し、これに "MyScrollDownArrowDown" という新しい名前を付けます。
2. [ファイル]-[新規] を選択し、Flash ドキュメントを作成します。
3. [ファイル]-[保存] を選択して、ファイルに "MyComboTest fla" のような一意の名前を付け保存します。
4. MyScrollDownArrowDown をテーマライブラリから "MyComboTest fla" のライブラリにドラッグします。
これにより、ステージに表示することなくシンボルをライブラリに追加できます。
5. [挿入]-[新規シンボル] を選択し、"MyComboBox" のような一意の名前を付けます。
6. [ActionScript に書き出し] チェックボックスをオンにして [OK] をクリックします。
[最初のフレームに書き出し] は、通常は自動的にオンになりますが、オフになっている場合はオンに切り替えてください。
7. MyComboBox シンボルのフレーム 1 の [アクション] パネルに次のコードを入力します。

```
#initclip 10
    import MyComboBox
    Object.registerClass("ComboBox", MyComboBox);
#endinitclip
```
8. シンボルの編集が完了したら、ステージの一番上の編集バーの左側にある [戻る] ボタンをクリックし、ドキュメント編集モードに戻ります。
9. ComboBox コンポーネントをステージにドラッグします。
10. プロパティインスペクタで、縦のスクロールバーを表示するために必要なすべての Label パラメータを入力します。
11. [ファイル]-[保存] を選択します。
12. [ファイル]-[新規] を選択し、新規 ActionScript ファイルを作成します。

13. 次のコードを入力します。

```
import mx.controls.ComboBox
import mx.controls.scrollClasses.ScrollBar
class MyComboBox extends ComboBox{
    function getDropdown():Object{
        var oldName = ScrollBar.prototype.downArrowDownName;
        ScrollBar.prototype.downArrowDownName = "MyScrollDownArrowDown";
        var r = super.getDropdown();
        ScrollBar.prototype.downArrowDownName = oldName;
        return r;
    }
}
```

14. [ファイル]-[保存] を選択し、"MyComboBox.as" という名前でファイルを保存します。
15. ファイル "MyComboTest fla" に戻ります。
16. ステージの空白部分をクリックし、プロパティインスペクタで [パブリッシュ] の横の [設定] ボタンをクリックします。
17. [ActionScript のバージョン] の横にある [設定] ボタンをクリックします。
18. [新規パスの追加](+) ボタンをクリックしてクラスパスを追加し、[ターゲット] ボタンを選択して、ハードディスク上の "MyComboBox.as" ファイルのある場所に移動します。
19. [制御]-[ムービープレビュー] を選択します。

テーマについて

テーマは、スタイルとスキンのコレクションです。Flash のデフォルトでは Halo (HaloTheme.fla) というテーマを使用しています。Halo テーマを使用すると、応答性がよく表現力のあるインターフェイスをユーザーに提供できます。Flash にはもう 1 つ、Sample ("SampleTheme.fla") というテーマも用意されています。Sample テーマでは、より多くのスタイルを使用してカスタマイズする例を紹介します。Halo テーマでは、Sample テーマに含まれるすべてのスタイルを使用するわけではありません。テーマファイルは、デフォルトインストールディレクトリの次のフォルダに格納されています。

- Windows : C:\Program Files\Adobe\Adobe Flash CS3\言語\Configuration\ComponentFLA\
- Macintosh : HD/ アプリケーション /Adobe Flash CS3/Configuration/ComponentFLA/

新しいテーマを作成してアプリケーションに適用し、すべてのコンポーネントの外観を変更することもできます。たとえば、ネイティブのオペレーティングシステムの外観に似たテーマを作成することもできます。

コンポーネントでは、スキン (グラフィックまたはムービークリップのシンボル) を使用して外観を表示します。各コンポーネントを定義する AS ファイルには、そのコンポーネント用のスキンをロードするコードが含まれています。Halo または Sample テーマのコピーを作成して、スキンのグラフィックを変更すれば、新しいテーマを簡単に作成できます。

テーマに新しいスタイルのデフォルト値のセットを含めることもできます。グローバルスタイル宣言やその他の追加のスタイル宣言を作成するには、ActionScript コードを記述する必要があります。詳細については、[113 ページの「テーマに含まれるスタイルプロパティのデフォルト値の変更」](#)を参照してください。

テーマの切り替え

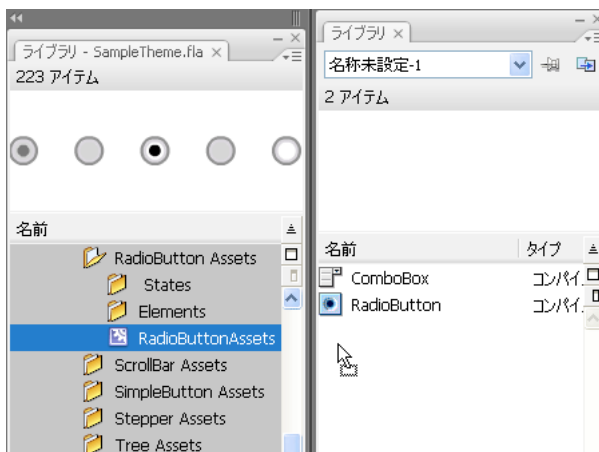
Flash には、Halo および Sample という 2 つのテーマがインストールされます。個々のコンポーネントについてのリファレンス情報には、各テーマ (または両方) で設定できるスタイルプロパティの一覧表が含まれています。スタイルプロパティの一覧表 (たとえば、Button コンポーネントの場合は、『ActionScript 2.0 コンポーネントリファレンスガイド』の「Button コンポーネントでのスタイルの使用」) を読む際には、必要なスタイルがどのテーマでサポートされているかに注意する必要があります。表には、この情報が Halo、Sample、または両方と記載されています (両方と記載されているスタイルプロパティは、どちらのテーマでもサポートされます)。

Halo テーマは各コンポーネントのデフォルトテーマです。Sample テーマを使用するには、現在のテーマの設定を Halo から Sample に切り替える必要があります。

Sample テーマに切り替えるには：

1. Flash で [ファイル]-[開く] を選択し、バージョン 2 コンポーネントを使用するドキュメントを開くか、[ファイル]-[新規] を選択して、バージョン 2 コンポーネントを使用する新規ドキュメントを作成します。
2. [ファイル]-[読み込み]-[外部ライブラリを開く] を選択して、"SampleTheme.fla" ファイルを選択し、ドキュメントに適用します。
このファイルは、アプリケーションレベル設定フォルダにあります。オペレーティングシステム別の正確な場所については、[108 ページの「テーマについて」](#)を参照してください。
3. "SampleTheme.fla" テーマの [ライブラリ] パネルで "Flash UI Components 2/Themes/MMDefault" を選択し、ドキュメントに含まれるいずれかのコンポーネントの "Assets" フォルダを、Flash ドキュメントの [ライブラリ] パネルにドラッグします。

たとえば、"RadioButton Assets" フォルダをライブラリにドラッグします。



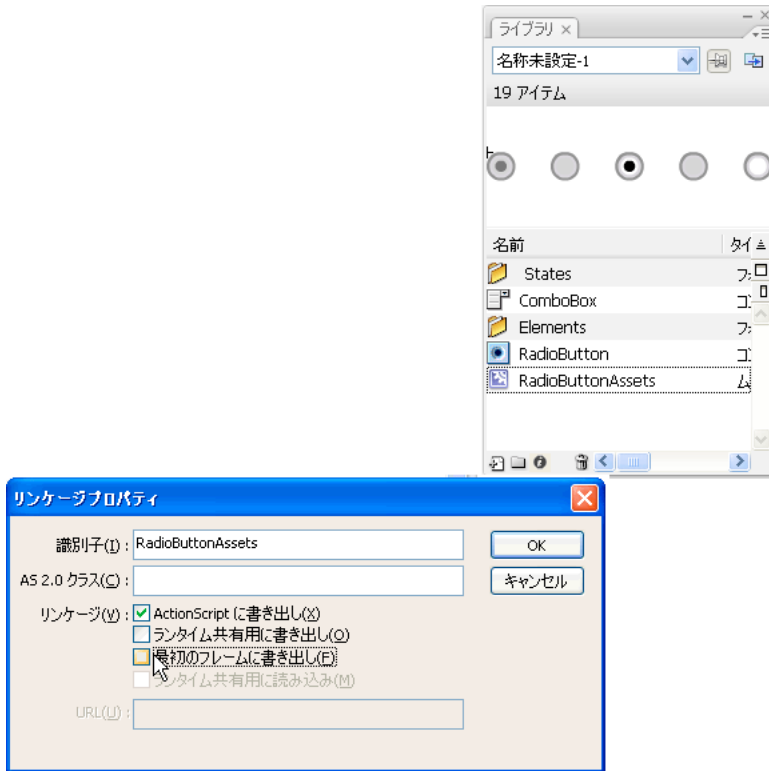
ドキュメントにどのコンポーネントが含まれているかがわからない場合は、Sample テーマ全体のムービークリップをステージにドラッグしてください。ドキュメント内のコンポーネントにスキンが自動的に割り当てられます。

× ❗	ステージで行うコンポーネントのライブプレビューには、新しいテーマは反映されません。
--------	---

4. 個別のコンポーネントの "Assets" フォルダをドキュメントの [ライブラリ] パネルにドラッグした場合は、各コンポーネントの Assets シンボルが [最初のフレームに書き出し] に設定されていることを確認してください。

たとえば、RadioButton コンポーネントの "Assets" フォルダの名前は "RadioButton Assets" です。この "RadioButton Assets" フォルダを開くと、RadioButtonAssets というムービークリップシンボルがあります。RadioButtonAssets シンボル内には、これに属する個別のアセットシンボルがすべて含まれています。

ドキュメントのライブラリで、RadioButtonAssets シンボルを右クリック (Windows) または Control キーを押しながらクリック (Macintosh) し、[リンケージ] メニューオプションを選択します。[最初のフレームに書き出し] をオンにして、個別のアセットシンボルがすべて1番目のフレームに書き出されるようにします。それから [OK] をクリックして設定を保存します。



5. 新しいテーマが反映されたドキュメントを確認するには、[制御]-[ムービープレビュー] を選択します。

新しいテーマの作成

Halo テーマや Sample テーマを使いたくない場合は、これらのテーマに変更を加えて新しいテーマを作成できます。

テーマに含まれるスキンの中には、サイズが固定されているものがあります。このようなスキンのサイズを大きくしたり小さくしたりすると、それに合わせてコンポーネントのサイズが自動的に変わります。その他のスキンは、静的な部分と伸縮可能な部分の複数の部分から構成されます。

RectBorder や ButtonSkin など一部のスキンでは、グラフィックが ActionScript の描画 API で描画されています。これは、サイズとパフォーマンスの面で効率がよいからです。このようなスキンに含まれている ActionScript のコードをテンプレートとして使用し、独自の調整を加えれば、目的に応じたスキンを描画できます。

各コンポーネントでサポートされるスキンとそのプロパティの一覧は、『ActionScript 2.0 コンポーネントリファレンスガイド』を参照してください。

新しいテーマを作成するには：

1. テンプレートとして使いたいテーマの FLA ファイルを選択し、コピーを作成します。
コピーしたファイルに、"MyTheme fla" などの一意の名前を付けます。
2. Flash で、[ファイル]-[開く]を選択し、"MyTheme fla" ファイルを開きます。
3. ライブラリが開かれていなければ、[ウィンドウ]-[ライブラリ]を選択し、ライブラリを開きます。
4. 変更したいスキンのシンボルをダブルクリックし、シンボル編集モードで開きます。
スキンは、"Flash UI Components 2/Themes/MMDefault/Component Assets" フォルダ (この例では "Flash UI Components 2/Themes/MMDefault/RadioButton Assets" フォルダ) にあります。
5. シンボルを変更するか、グラフィックを削除して新しいグラフィックを作成します。
必要に応じて [表示]-[ズームイン]を選択し、表示比率を上げます。スキンが正常に表示されるように、スキンを編集する際には必ず基準点を維持してください。編集したすべてのシンボルについて、左上隅が (0,0) になるようにする必要があります。
たとえば、States/RadioFalseDisabled アセットを開き、内側の円を明るい灰色に変更します。
6. スキンシンボルの編集が完了したら、ステージの一番上の編集バーの左側にある [戻る] ボタンをクリックし、ドキュメント編集モードに戻ります。
7. 変更を加えるすべてのスキンに対して、手順 4 ～ 6 を実行します。
8. この章で示す手順に従って、"MyTheme fla" ファイルをドキュメントに適用します。詳細については、[114 ページの「ドキュメントへの新しいテーマの適用」](#)を参照してください。

テーマに含まれるスタイルプロパティのデフォルト値の変更

スタイルプロパティのデフォルト値は、Default クラス内の各テーマから提供されます。カスタムテーマ用にデフォルト値を変更するには、テーマに適したパッケージ内に Default という新しい ActionScript クラスを作成し、デフォルト設定を変更します。

テーマに含まれるスタイルのデフォルト値を変更するには：

1. "First Run/Classes/mx/skins" フォルダ内にカスタムテーマ用の新しいフォルダを作成します。
たとえば、"myTheme" というフォルダを作成します。
2. 既存の Defaults クラスを新しいテーマフォルダにコピーします。
たとえば、"mx/skins/halo/Defaults.as" を "mx/skins/myTheme/Defaults.as" としてコピーします。
3. 新しい Defaults クラスを ActionScript エディタで開きます。
Flash を使用している場合は、Flash 内でファイルを開くことができます。それ以外の場合は、メモ帳 (Windows) や SimpleText (Macintosh) で開きます。
4. 新しいパッケージを反映するようにクラス宣言を変更します。
たとえば、新しいクラス宣言を `class mx.skins.myTheme.Defaults` のようにします。
5. スタイル設定を変更します。
たとえば、デフォルトで無効に設定されているカラーを濃い赤色に変更します。
`o.disabledColor = 0x663333;`
6. 変更した "Defaults" クラスファイルを保存します。
7. 既存の FocusRect クラスを元のテーマからカスタムテーマにコピーします。
たとえば、"mx/skins/halo/FocusRect.as" を "mx/skins/myTheme/FocusRect.as" としてコピーします。
8. 新しい FocusRect クラスを ActionScript エディタで開きます。
9. 元のテーマへのパッケージに対するすべての参照を新しいテーマのパッケージに対する参照に変更します。
たとえば、出現するすべての "halo" を "myTheme" に変更します。
10. 変更した "FocusRect" クラスファイルを保存します。
11. カスタムテーマの FLA ファイルを開きます。
この例では、"MyTheme fla" を使用します。
12. ライブラリ ([ウィンドウ]-[ライブラリ]) を開き、Defaults シンボルを検索します。
この例では、"Flash UI Components 2/Themes/MMDefault/Defaults" 内にあります。
13. Default シンボルのシンボルプロパティを編集します。

14. 新しいパッケージを反映するように AS 2.0 クラスの設定を変更します。
この例では、mx.skins.myTheme.Defaults に変更します。
15. [OK] をクリックします。
16. FocusRect シンボルを検索します。
この例では、"Flash UI Components 2/Themes/MMDefault/FocusRect" 内にあります。
17. FocusRect シンボルのシンボルプロパティを編集します。
18. 新しいパッケージを反映するように AS 2.0 クラスの設定を変更します。
この例では、mx.skins.myTheme.FocusRect に変更します。
19. [OK] をクリックします。
20. 次のセクションで示す手順に従って、このカスタムテーマをドキュメントに適用します。
カスタムテーマからドキュメントにアセットをドラッグするときに、Defaults シンボルと FocusRect シンボルも必ず含めてください。

この例では、新しいテーマを使用して、無効なコンポーネントのテキストカラーをカスタマイズしました。この特定のカスタマイズ例に関しては、1つのスタイルプロパティのデフォルト値を変更しただけなので、[82 ページの「スタイルによるコンポーネントのカラーとテキストの変更」](#)で説明されているようにスタイルを使用する方が簡単です。新しいテーマを作成してデフォルト値をカスタマイズする方法は、多数のスタイルプロパティをカスタマイズする場合、または、コンポーネントのグラフィックをカスタマイズするために新しいテーマを作成済みの場合に適しています。

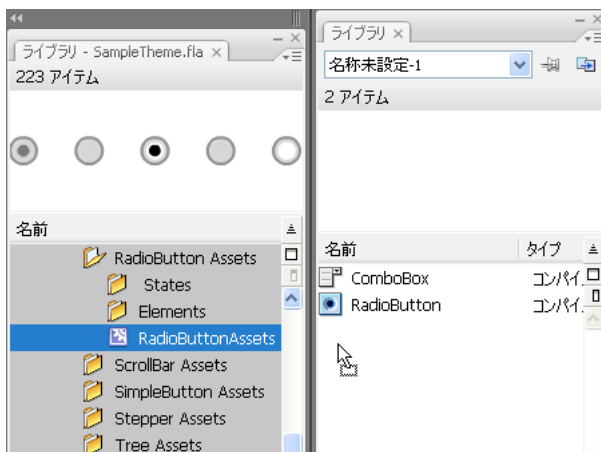
ドキュメントへの新しいテーマの適用

ドキュメントに新しいテーマを適用するには、テーマの FLA ファイルを外部ライブラリとして開き、その外部ライブラリからテーマのフォルダをドキュメントライブラリにドラッグします。その詳しい手順を次に説明します (新しいテーマは既に用意してあるものと仮定しています。詳細については、[112 ページの「新しいテーマの作成」](#)を参照してください)。


ドキュメントにテーマを適用するには：

1. Flash で [ファイル]-[開く] を選択し、バージョン 2 コンポーネントを使用するドキュメントを開くか、[ファイル]-[新規] を選択して、バージョン 2 コンポーネントを使用する新規ドキュメントを作成します。
2. [ファイル]-[読み込み]-[外部ライブラリを開く] を選択し、ドキュメントに適用するテーマの FLA ファイルを選択します。
3. テーマの [ライブラリ] パネルで "Flash UI Components 2/Themes/MMDefault" を選択し、必要なコンポーネントの "Assets" フォルダをドキュメントのライブラリにドラッグします。

たとえば、"RadioButton Assets" フォルダをライブラリにドラッグします。

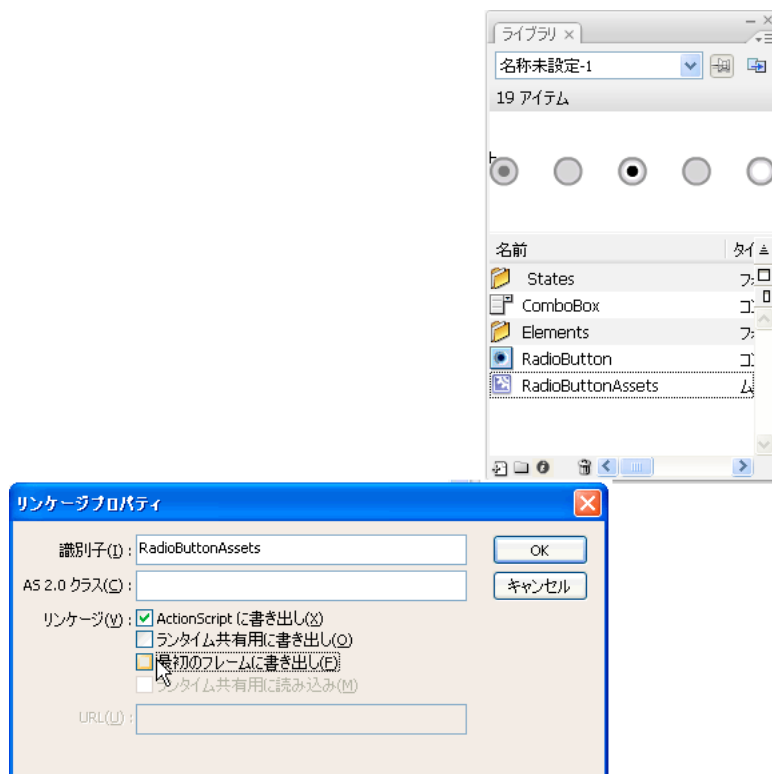


ドキュメントにどのコンポーネントが含まれているのかわからない場合は、テーマ全体のムービークリップ (たとえば、"SampleTheme.fla" の場合、メインとなるテーマムービークリップは Flash UI Components 2/SampleTheme) をステージにドラッグしてください。ドキュメント内のコンポーネントにスキンが自動的に割り当てられます。

 ステージで行うコンポーネントのライブプレビューには、新しいテーマは反映されません。

4. 個別のコンポーネントの "Assets" フォルダを "ThemeApply.fla" ライブラリにドラッグした場合は、各コンポーネントの Assets シンボルが [最初のフレームに書き出し] に設定されていることを確認します。

たとえば、RadioButton コンポーネントの "Assets" フォルダは "RadioButton Assets" で、RadioButtonAssets というシンボルが格納されています。このシンボルには、すべての個別のアセットシンボルが含まれています。RadioButtonAssets シンボルに [最初のフレームに書き出し] を設定すると、すべての個別のアセットも最初のフレームに書き出されます。



5. [制御]-[ムービープレビュー] を選択して、新しいテーマが適用されたことを確認します。

書き出し設定の変更

Sample または Halo のテーマをドキュメントに適用すると、多数のスキナセットが、最初のフレームに書き出すよう設定され、再生中にコンポーネントでスキナセットを即座に利用できるようになります。ただし、FLA ファイルのパブリッシュ書き出し設定 ([ファイル]-[パブリッシュ設定]-[Flash] タブ-[ActionScript のバージョン]-[設定] ボタン-[クラス用のフレームの書き出し]) を最初のフレームの後ろにくるフレームに変更した場合、Sample および Halo テーマのアセットの書き出し設定も変更する必要があります。この変更を実行するには、ドキュメントのライブラリ内の次に示すコンポーネントアセットを開いて、[最初のフレームに書き出し](右クリックして、[リンケージ]-[最初のフレームに書き出し]) チェックボックスをオフにする必要があります。

Sample テーマ

- Flash UI Components 2/Base Classes/UIObject
- Flash UI Components 2/Themes/MMDefault/Defaults
- Flash UI Components 2/Base Classes/UIObjectExtensions
- Flash UI Components 2/Border Classes/BoundingBox
- Flash UI Components 2/SampleTheme
- Flash UI Components 2/Themes/MMDefault/Button Assets/Elements/ButtonIcon
- Flash UI Components 2/Themes/MMDefault/DateChooser Assets/ Elements/Arrows/ cal_disabledArrow
- Flash UI Components 2/Themes/MMDefault/FocusRect
- Flash UI Components 2/Themes/MMDefault/Window Assets/ States/CloseButtonOver
- Flash UI Components 2/Themes/MMDefault/Accordion Assets/AccordionHeaderSkin
- Flash UI Components 2/Themes/MMDefault/Alert Assets/AlertAssets
- Flash UI Components 2/Themes/MMDefault/Border Classes/Border
- Flash UI Components 2/Themes/MMDefault/Border Classes/CustomBorder
- Flash UI Components 2/Themes/MMDefault/Border Classes/RectBorder
- Flash UI Components 2/Themes/MMDefault/Button Assets/ActivatorSkin
- Flash UI Components 2/Themes/MMDefault/Button Assets/ButtonSkin

Halo テーマ

- Flash UI Components 2/Base Classes/UIObject
- Flash UI Components 2/Themes/MMDefault/Defaults
- Flash UI Components 2/Base Classes/UIObjectExtensions
- Flash UI Components 2/Component Assets/BoundingBox
- Flash UI Components 2/HaloTheme
- Flash UI Components 2/Themes/MMDefault/Accordion Assets/AccordionHeaderSkin

- Flash UI Components 2/Themes/MMDefault/Alert Assets/AlertAssets
- Flash UI Components 2/Themes/MMDefault/Border Classes/Border
- Flash UI Components 2/Themes/MMDefault/Border Classes/CustomBorder
- Flash UI Components 2/Themes/MMDefault/Border Classes/RectBorder
- Flash UI Components 2/Themes/MMDefault/Button Assets/ActivatorSkin
- Flash UI Components 2/Themes/MMDefault/Button Assets/ButtonSkin
- Flash UI Components 2/Themes/MMDefault/Button Assets/Elements/ButtonIcon
- Flash UI Components 2/Themes/MMDefault/CheckBox Assets/Elements/CheckThemeColor1
- Flash UI Components 2/Themes/MMDefault/CheckBox Assets/CheckBoxAssets
- Flash UI Components 2/Themes/MMDefault/ComboBox Assets/ComboBoxAssets
- Flash UI Components 2/Themes/MMDefault/DataGrid Assets/DataGridAssets
- Flash UI Components 2/Themes/MMDefault/DateChooser Assets/DateChooserAssets
- Flash UI Components 2/Themes/MMDefault/FocusRect
- Flash UI Components 2/Themes/MMDefault/Menu Assets/MenuAssets
- Flash UI Components 2/Themes/MMDefault/MenuBar Assets/MenuBarAssets
- Flash UI Components 2/Themes/MMDefault/ProgressBar Assets/ProgressBarAssets
- Flash UI Components 2/Themes/MMDefault/RadioButton Assets/Elements/RadioThemeColor1
- Flash UI Components 2/Themes/MMDefault/RadioButton Assets/Elements/
RadioThemeColor2
- Flash UI Components 2/Themes/MMDefault/RadioButton Assets/RadioButtonAssets
- Flash UI Components 2/Themes/MMDefault/ScrollBar Assets/HScrollBarAssets
- Flash UI Components 2/Themes/MMDefault/ScrollBar Assets/ScrollBarAssets
- Flash UI Components 2/Themes/MMDefault/ScrollBar Assets/VScrollBarAssets
- Flash UI Components 2/Themes/MMDefault/Stepper Assets/Elements/StepThemeColor1
- Flash UI Components 2/Themes/MMDefault/Stepper Assets/NumericStepperAssets
- Flash UI Components 2/Themes/MMDefault/Tree Assets/TreeAssets
- Flash UI Components 2/Themes/MMDefault/Window Assets/Window Assets

スキンとスタイルの組み合わせによるコンポーネントのカスタマイズ

このセクションでは、スタイル、テーマ、およびスキン設定を使ってコンボボックスコンポーネントインスタンスをカスタマイズします。この手順では、スキンとスタイル設定を組み合わせ、コンポーネントの一意の外観を作成する方法を示します。

ステージでのコンポーネントインスタンスの作成

この実習の前半では、カスタマイズの対象となる ComboBox インスタンスを作成する必要があります。

ComboBox インスタンスを作成するには：

1. ComboBox コンポーネントをステージにドラッグします。
2. [プロパティ] パネルで、インスタンスに **my_cb** という名前を付けます。
3. メインタイムラインの最初のフレームに次の **ActionScript** を追加します (**ActionScript** を、コンポーネント自体ではなくフレームに追加していることを確認します。[アクション] パネルのタイムラインには、[アクション - フレーム] が表示されるはずです)。

```
my_cb.addItem({data:1, label:"One"});  
my_cb.addItem({data:2, label:"Two"});
```

4. [制御]-[ムービープレビュー] を選択して、Halo テーマのデフォルトのスタイルおよびスキン設定が適用されたコンボボックスを表示します。

新しいスキン宣言の作成

次に、新しいスタイル宣言を作成して、このスタイル宣言にスタイルを割り当てる必要があります。スタイル宣言に必要なすべてのスタイルを割り当てた後、新しいスタイル名をコンボボックスインスタンスに割り当てることができます。

新しいスタイル宣言を作成して、名前を指定するには：

1. メインタイムラインの最初のフレームで、**ActionScript** の先頭に次の行を追加します (コーディング規則として、すべての読み込みステートメントは、**ActionScript** の冒頭に記述する必要があります)。

```
import mx.styles.CSSStyleDeclaration;
```

2. その次の行に、新しいスタイル宣言の名前を指定し、その名前をグローバルスタイル定義に追加します。

```
var new_style:Object = new CSSStyleDeclaration();
_global.styles.myStyle = new_style;
```

新しいスタイル宣言を `_global` スタイルシートに割り当てた後、個別のスタイル設定を `new_style` スタイル宣言に関連付けることができます。単一のインスタンスにスタイル定義を設定するのではなく、コンポーネントグループ用にスタイルシートを作成する方法の詳細については、[88 ページの「コンポーネントのグループに対するカスタムスタイルの設定」](#)を参照してください。

3. スタイル設定を `new_style` スタイル宣言に関連付けます。次のスタイル設定には、ComboBox コンポーネントで利用可能なスタイル定義 (詳細な一覧については、『[ActionScript 2.0 コンポーネントリファレンスガイド](#)』の「[ComboBox コンポーネントでのスタイルの使用](#)」を参照) が含まれています。さらに、ComboBox コンポーネントは `RectBorder` クラスを使用するので、このスタイル設定には `RectBorder` クラスのスタイルも含まれています。

```
new_style.setStyle("textAlign", "right");
new_style.setStyle("selectionColor", "white");
new_style.setStyle("useRollOver", false);
// RectBorder クラスの borderStyle
new_style.setStyle("borderStyle", "none");
```

コンボボックスへのスタイル定義の割り当て

この時点で、さまざまなスタイルを含むスタイル宣言が用意されました。しかし、明示的にスタイル名をコンポーネントインスタンスに割り当てる必要があります。この新しいスタイル宣言を、ドキュメント内の任意のコンポーネントインスタンスに次の方法で割り当てることができます。`my_cb` の `addItem()` ステートメントの後に次の行を追加します (コーディング規則として、すべてのコンボボックス作成ステートメントは1か所にまとめる必要があります)。

```
my_cb.setStyle("styleName", "myStyle");
```

メインタイムラインの最初のフレームに関連付けられた `ActionScript` コードは次のようになります。

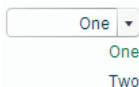
```
import mx.styles.CSSStyleDeclaration;

var new_style:Object = new CSSStyleDeclaration();
_global.styles.myStyle = new_style;

new_style.setStyle("textAlign", "right");
new_style.setStyle("selectionColor", "white");
new_style.setStyle("useRollOver", false);
// RectBorder クラスの borderStyle
new_style.setStyle("borderStyle", "none");

my_cb.addItem({data:1, label:"One"});
my_cb.addItem({data:2, label:"Two"});
my_cb.setStyle("styleName", "myStyle");
```


[制御]-[ムービープレビュー] を選択して、スタイルが設定されたコンボボックスを表示します。



コンボボックスのテーマの変更

すべてのユーザーインターフェイスコンポーネントには、そのコンポーネントで設定可能なスタイルプロパティの一覧が存在します (たとえば、ComboBox コンポーネントに設定可能なすべてのスタイルプロパティは『ActionScript 2.0 コンポーネントリファレンスガイド』の「ComboBox コンポーネントのカスタマイズ」に一覧表示されています)。スタイルプロパティの表の "テーマ" という名前が付いた列に、インストール済みのテーマが各スタイルプロパティをサポートしているかどうかが表示されます。すべてのスタイルプロパティが、インストール済みの全テーマでサポートされるわけではありません。すべてのユーザーインターフェイスコンポーネントのデフォルトのテーマは Halo です。Halo テーマを Sample テーマに変更する場合、異なるスタイルプロパティセットを使用できます (プロパティが Halo にのみ示されている場合、そのプロパティは Sample テーマでは使用できません)。

スタイルが設定されたコンポーネントのテーマを変更するには：

1. [ファイル]-[読み込み]-[外部ライブラリを開く] を選択し、"SampleTheme.flb" を選択して、サンプルテーマライブラリを Flash で開きます。

このファイルは、アプリケーションレベル設定フォルダにあります。

- Windows : C:\Program Files\Adobe\Adobe Flash CS3\言語 > Configuration\ComponentFLA
- Macintosh : HD/ アプリケーション /Adobe Flash CS3/Configuration/ComponentFLA/

2. メイン SampleTheme (Flash UI Components 2/SampleTheme) ムービークリップを SampleTheme ライブラリからドキュメントのライブラリにドラッグします。

ComboBox コンポーネントは、いくつかのコンポーネントとクラスが組み合わさって構成されており、Border アセットや ScrollBar アセットなどのコンポーネントおよびアセットを必要とします。必要とするテーマのすべてのアセットを読み込んでいることを確認する最も簡単な方法は、テーマのすべてのアセットをライブラリにドラッグすることです。

3. [制御]-[ムービープレビュー] を選択して、スタイルが設定されたコンボボックスを表示します。



コンボボックスのスキンアセットの編集

コンポーネントの外観を編集するには、コンポーネントを構成するスキンを編集します。スキンを編集するには、現在のテーマ内からコンポーネントのグラフィックアセットを開き、そのコンポーネントのシンボルを編集します。コンポーネントの外観を編集する場合は、この方法を実行するようにしてください。これは、他のコンポーネントで必要とされる可能性があるシンボルを、この方法では削除または追加する必要がないためです。この方法により、既存のコンポーネントスキンシンボルの外観を編集します。

×
注

コンポーネントが異なる名前を持つシンボルをスキンとして使用できるよう、コンポーネントのソースクラスファイルを編集することは可能ですが、推奨されません。スキンシンボル内の `ActionScript` をプログラムで変更することができます (カスタマイズされた `ActionScript` とスキンシンボルの例については、『`ActionScript 2.0` コンポーネントリファレンスガイド』の「`Accordion` コンポーネントのカスタマイズ」を参照)。ただし、`ComboBox` コンポーネントなどの一部のコンポーネントは、他のコンポーネントとアセットを共有しているので、ソースファイルを編集したり、スキンシンボルの名前を変更したりすると、予期しない結果が生じる可能性があります。

コンポーネントのスキンシンボルを編集した場合、次の説明が適用されます。

- 編集対象のコンポーネントのすべてのインスタンスは、新しいスキンを使用します (ただし、インスタンスにスタイルを明示的に関連付けない限り、カスタムスタイルは使用されません)。さらに、このコンポーネントに依存しているコンポーネントも新しいスキンを使用します。
- コンポーネントスキンの編集後に新しいテーマを割り当てた場合、既存の " 編集済み " スキンを上書きしないようにします (スキンを上書きするかどうかを確認するダイアログボックスが表示されます。このダイアログボックスを使って、Flash がスキンを上書きしないよう設定できます)。

このセクションでは、前述のセクションのコンボボックスを引き続き使用します ([121 ページの「コンボボックスのテーマの変更」](#) を参照)。次に示す手順を実行すると、コンボボックスを開く下矢印の外観が、矢印から円に変更されます。

コンボボックスの下矢印シンボルを編集するには：

1. ドキュメントのライブラリで `ComboBox` アセットを開き、実行時にコンボボックスインスタンスの開閉に使用するボタンのスキンであるムービークリップを表示します。具体的には、`"Themes/MMDefault/ComboBox Assets/States"` フォルダをドキュメントのライブラリで開きます。

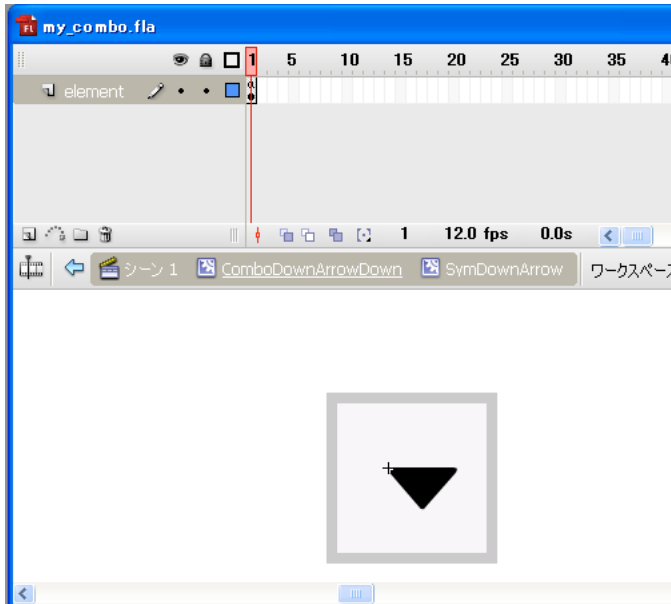
`"States"` フォルダには、`ComboDownArrowDisabled`、`ComboDownArrowDown`、`ComboDownArrowOver`、および `ComboDownArrowUp` の各ムービークリップが存在します。この 4 つのシンボルはすべて他のシンボルで構成されます。さらにこの 4 つのシンボルは、`SymDownArrow` と呼ばれる下矢印 (三角形) について同じシンボルを使用しています。
2. `ComboDownArrowDown` シンボルをダブルクリックして編集します。

ボタンの詳細を表示するには、拡大が必要になる場合があります (最大 800% の拡大が可能)。

3. 下矢印 (黒色の三角形) をダブルクリックして編集します。

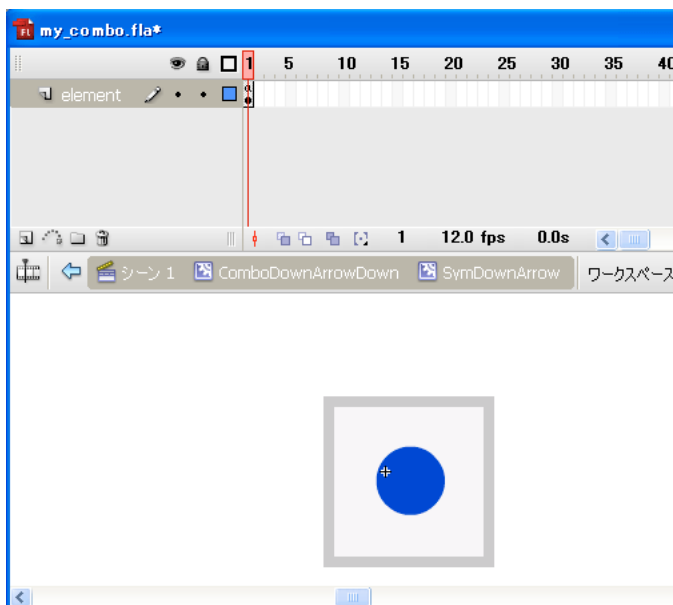
✕
rh

シンボル `SymDownArrow` が選択されていることを確認します。つまり、ムービークリップ自体ではなく、ムービークリップ内のシェイプのみを削除するようにします。

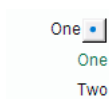


4. ステージ上の選択した下矢印 (ムービークリップ全体ではなく黒色の三角形シェイプ) を削除します。
5. `SymDownArrow` の編集で、下矢印が配置されていた場所に円を描画します。

変更をわかりやすくするため、青などの明るい色で円を描画し、およそ 4 x 4 ピクセルの大きさの円を、x 座標が 0、y 座標が -1 で中心に配置するようにしてください。



6. [制御]-[ムービープレビュー] を選択して、スキンが設定されたコンボボックスを表示します。



ドキュメントのライブラリで、ComboDownArrowOver と ComboDownArrowUp を選択した場合も、黒色の三角形の代わりに青色の円が表示されます。これは、これらのシンボルも SymDownArrow と同じ下矢印シンボルを使用しているためです。

コンポーネントの作成

この章では、ユーザー独自のコンポーネントを作成して配布用にパッケージ化する方法について説明します。

この章では、次のセクションについて説明します。

コンポーネントのソースファイル.....	125
コンポーネント構造の概要	126
初めてのコンポーネントの作成.....	127
親クラスの選択	136
コンポーネントムービークリップの作成.....	138
ActionScript クラスファイルの作成	143
複数の既存コンポーネントを組み込んだコンポーネントの作成.....	172
コンポーネントの書き出しと配布.....	180
コンポーネント開発の最後の手順.....	184

コンポーネントのソースファイル

[コンポーネント] パネルに表示されるコンポーネントは、プリコンパイルされた SWC クリップです。グラフィックが含まれているソース Flash ドキュメント (FLA) と、コンポーネントのコードが含まれているソース ActionScript クラスファイル (AS) も提供されているので、カスタムコンポーネントの作成時に利用できます。バージョン 2 コンポーネントのソースファイルは Adobe Flash と共にインストールされます。これらのファイルをいくつか開いて目を通し、構造を理解しておく、独自のコンポーネントを構築する際の参考になります。当初は RadioButton コンポーネントなどの比較的単純なコンポーネントから調査を開始することをお勧めします。バージョン 2 コンポーネントはすべて、"StandardComponents.fla" のライブラリに含まれるシンボルです。各シンボルはそれぞれ 1 つの ActionScript クラスにリンクされています。各種ファイルの場所を次に示します。

■ FLA ファイルのソースコード：

- Windows : C:\Program Files\Adobe\Adobe Flash CS3\言語\Configuration\ComponentFLA\StandardComponents.fla

- Macintosh : HD/ アプリケーション /Adobe Flash CS3/Configuration/ComponentFLA/StandardComponents fla
- ActionScript クラスファイル
 - Windows : C:\Program Files\Adobe\Adobe Flash CS3\言語 \First Run\Classes\mx
 - Macintosh : HD/ アプリケーション /Adobe Flash CS3/First Run/Classes/mx

コンポーネント構造の概要

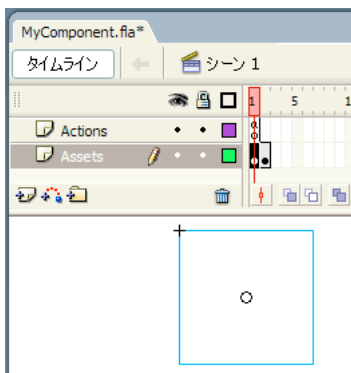
コンポーネントは、Flash (FLA) ファイルと ActionScript (AS) ファイルで構成されます。必要に応じ、これ以外のファイル (アイコンや .swd デバッグファイルなど) を作成してコンポーネントと共にパッケージ化することもできますが、FLA ファイルと ActionScript ファイルはすべてのコンポーネントに必要です。コンポーネントの開発が終了したら、それを SWC ファイルとして書き出します。



Flash (FLA) ファイル、ActionScript (AS) ファイル、SWC ファイル

FLA ファイルにはムービークリップシンボルが1つ含まれており、これを [リンケージプロパティ] ダイアログボックスと [コンポーネント定義] ダイアログボックスの両方で AS ファイルにリンクする必要があります。

このムービークリップシンボルには、2つのフレームと2つのレイヤーがあります。最初のレイヤーは Actions レイヤーで、フレーム1に stop() グローバル関数があります。もう1つのレイヤーは Assets レイヤーで、2つのキーフレームがあります。フレーム1には境界ボックスが含まれます。フレーム2には、コンポーネントが使用するその他のアセット (グラフィックや基本クラスなど) が含まれています。



コンポーネントのプロパティおよびメソッドを指定する ActionScript コードは、個別の ActionScript クラスファイルに格納します。また、コンポーネントが何らかのクラスを継承する場合は、このクラスファイルで継承を宣言します。AS クラスファイルの名前とコンポーネントの名前は同じであり、拡張子として .as が付きます。たとえば、MyComponent というコンポーネントのソースコードを含んだファイルの場合、ファイル名は必ず "MyComponent.as" となります。

コンポーネントの FLA ファイルと AS ファイルは、同じフォルダに保存し、同じ名前を付けることをお勧めします。AS ファイルが同じフォルダに保存されていない場合は、FLA ファイルが AS ファイルを参照できるよう、そのフォルダを必ずクラスパスに含める必要があります。クラスパスの詳細については、『ActionScript 2.0 の学習』の「クラス」を参照してください。

初めてのコンポーネントの作成

このセクションでは、Dial コンポーネントを作成します。Dial コンポーネントのサンプルについては、Web サイトの Flash サンプルページ (www.adobe.com/go/learn_fl_tutorials_jp) を参照してください。次のサンプルファイルがあります。

- Dial fla
- Dial.as
- DialAssets fla

Dial コンポーネントは、電圧の電位差の測定に使用される電位差計です。指針をクリックしてドラッグすると、指針の位置を変えることができます。Dial コンポーネントの API には value という唯一のプロパティがあり、これを使用して指針の位置を取得、設定できます。

このセクションでは、コンポーネントの作成手順について説明します。こうした手順の詳細については、後述のセクション (136 ページの「親クラスの選択」、138 ページの「コンポーネントムービークリップの作成」、143 ページの「ActionScript クラスファイルの作成」、および 180 ページの「コンポーネントの書き出しと配布」など) を参照してください。このセクションでは、次のトピックについて説明します。

- 128 ページの「Dial の Flash (FLA) ファイルの作成」
- 131 ページの「Dial クラスファイルの作成」
- 134 ページの「Dial コンポーネントのテストと書き出し」

Dial の Flash (FLA) ファイルの作成

コンポーネントを作成する最初の手順には、FLA ドキュメントファイル内でコンポーネントムービーファイルを作成する手順があります。

Dial の FLA ファイルを作成するには：

1. Flash で [ファイル]-[新規] を選択し、新しいドキュメントを作成します。
2. [ファイル]-[名前を付けて保存] を選択し、ファイル名を "Dial.fla" として保存します。
ファイルには任意の名前を付けることができますが、コンポーネントと同じ名前にするのが実用的です。
3. [挿入]-[新規シンボル] を選択します。コンポーネント自体は新しい MovieClip シンボルとして作成され、ライブラリを通して使用可能になります。
コンポーネントに Dial という名前を付け、[ムービークリップ] タイプを選択します。
4. [新規シンボルの作成] ダイアログボックスの [リンケージ] セクションが開いていない場合は、[詳細] ボタンをクリックして開きます。
5. [リンケージ] 領域で、[ActionScript に書き出し] をオンに、[最初のフレームに書き出し] をオフにします。
6. [識別子] テキストボックスに、「Dial_ID」などのリンケージ識別子を入力します。
7. [AS 2.0 クラス] テキストボックスに「Dial」と入力します。この値はコンポーネントのクラス名です。クラスがパッケージに含まれる場合は、パッケージ名全体を入力します
(例 : mx.controls.Button)。
8. [OK] をクリックします。
シンボル編集モードに自動的に切り替わります。
9. 新しいレイヤーを挿入します。上側のレイヤーに **Actions**、下側のレイヤーに **Assets** という名前を付けます。
10. Assets レイヤーのフレーム 2 を選択し、F6 キーを押してキーフレームを挿入します。
これでコンポーネントムービークリップの構造 (Actions レイヤーと Assets レイヤー) が作成されました。Actions レイヤーにはキーフレームが 1 つ、Assets レイヤーにはキーフレームが 2 つになります。
11. Actions レイヤーのフレーム 1 を選択し、F9 キーを押して [アクション] パネルを開きます。
stop(); グローバル関数を入力します。
これにより、ムービークリップがフレーム 2 に進むのを防ぎます。

12. [ファイル]-[読み込み]-[外部ライブラリを開く] を選択し、"Configuration/ComponentFLA" フォルダから "StandardComponents.flc" ファイルを選択します。プラットフォームごとの実際の場所は次のようになります。

- Windows : C:\Program Files\Adobe\Adobe Flash CS3\言語 > Configuration\StandardComponents.flc
- Macintosh : HD/ アプリケーション /Adobe Flash CS3/Configuration/ComponentFLA/StandardComponents.flc



フォルダの場所の詳細については、『Flash ユーザーガイド』の「Flash と共にインストールされる設定フォルダ」を参照してください。

13. Dial は UIComponent 基本クラスを継承するので、UIComponent のインスタンスを Dial ドキュメントにドラッグする必要があります。"StandardComponents.flc" のライブラリで、"Flash UI Components 2/Base Classes/FUIObject Subclasses" フォルダにある UIComponent ムービークリップを選択し、"Dial.flc" のライブラリにドラッグします。

依存するアセットは、UIComponent と共に Dial のライブラリにコピーされます。



UIComponent を Dial のライブラリにドラッグすると、Dial ライブラリ内のフォルダ階層構造が変化します。ライブラリを再利用する場合や、他のコンポーネントのグループ (バージョン 2 コンポーネントなど) とライブラリを共用する場合は、整理してシンボルの重複を防ぐために、"StandardComponents.flc" のライブラリに合わせてフォルダ階層構造を再構成してください。

14. Assets レイヤーのフレーム 2 を選択し、UIComponent のインスタンスをステージにドラッグします。
15. "StandardComponents.flc" のライブラリを閉じます。
16. [ファイル]-[読み込み]-[外部ライブラリを開く] を選択して、"DialAssets.flc" ファイルを開きます。
- "DialAssets.flc" ファイルのサンプルについては、Web サイトの Flash サンプルページ (www.adobe.com/go/learn_fl_tutorials_jp) を参照してください。

17. Assets レイヤーのフレーム 2 を選択し、DialAssets のライブラリから、DialFinal ムービークリップのインスタンスをステージにドラッグします。

コンポーネントアセットは、すべて Assets レイヤーのフレーム 2 に追加されます。Actions レイヤーのフレーム 1 には stop() グローバル関数が設定されているため、フレーム 2 のアセットがステージに配置されていても、こうしたアセットは表示されません。

アセットをフレーム 2 に追加するのは、次の 2 つの理由によります。

- アセットやサブアセットがすべて自動的にライブラリにコピーされ、動的にインスタンス化 (DialFinal の場合) またはメソッド、プロパティおよびイベントにアクセス (UIComponent の場合) できるようにするため。
- フレームにアセットを配置することにより、ムービーがストリーミングされるときにアセットがよりスムーズにロードされ、ライブラリのアセットを 1 番目のフレームに書き出すよう設定する必要がなくなるため。この方法により、初期のダウンロード中にデータ転送が途切れるのを回避できます。

18. "DialAssets.fla" のライブラリを閉じます。

19. Assets レイヤーでフレーム 1 を選択します。ライブラリの "Flash UI Components 2/ Component Assets" フォルダから BoundingBox ムービークリップをステージにドラッグします。BoundingBox インスタンスに boundingBox_mc という名前を付けます。[情報] パネルを使用して、DialFinal ムービークリップの高さと幅を 250 ピクセルに、x 座標と y 座標を 0 に設定します。

この BoundingBox インスタンスは、コンポーネントのライブプレビューを作成し、オーサリング中にサイズを変更するために使用します。境界ボックスのサイズは、コンポーネント内のグラフィカルエレメントをすべて囲むように設定する必要があります。

× ❗	コンポーネント (バージョン 2 コンポーネントを含む) を継承する場合は、継承元コンポーネントのコードが既に使用しているインスタンス名を参照するので、継承元コンポーネントで使用しているインスタンス名を一切変更せずそのまま使用する必要があります。たとえば、インクルードするバージョン 2 コンポーネントで boundingBox_mc というインスタンス名が使用されている場合、この名前を変更してはなりません。独自に作成するインスタンスについては、同じスコープ内で既に使用されている名前とコンフリクトしない一意の名前であれば任意の名前を使用できます。
--------	---

20. ライブラリで Dial ムービークリップを選択し、[ライブラリ] コンテキストメニューから [コンポーネント定義] を選択します (Windows の場合は右クリック、Mac の場合は Control キーを押しながらクリック)。

21. [AS 2.0 クラス] テキストボックスに「Dial」と入力します。

この値は ActionScript クラスの名前です。クラスがパッケージに含まれる場合、この値にはパッケージ名全体を指定する必要があります (例 : mx.controls.CheckBox)。

22. [OK] をクリックします。

23. ファイルを保存します。

Dial クラスファイルの作成

次に、新しい ActionScript ファイルとして Dial クラスファイルを作成する必要があります。

Dial クラスファイルを作成するには：

1. Flash で、[ファイル]-[新規] を選択し、[ActionScript ファイル] を選択します。
2. [ファイル]-[名前を付けて保存] を選択し、ファイルに **Dial.as** という名前を付けて、"Dial.fla" ファイルと同じフォルダに保存します。



任意のテキストエディタを使って、"Dial.as" ファイルを保存できます。

3. 次に示す Dial コンポーネントの ActionScript クラスコードを、新しく作成する "Dial.as" ファイルにコピーまたは入力してください。コンポーネントコードの各エレメントに早く慣れるには、コピーせず手作業で入力することをお勧めします。

コード内の各部分についてはコメントで説明しています。コンポーネントクラスファイルの各種エレメントの詳細については、[144 ページの「コンポーネントのクラスファイルの概要」](#)を参照してください。

```
// パッケージを読み込んで、
// クラスを直接的に参照できるようにする。
import mx.core.UIComponent;

// Event メタデータタグ
[Event("change")]
class Dial extends UIComponent
{
    // コンポーネントでは次のように宣言する。正当なコンポーネントとして
    // コンポーネントフレームワーク内で認識されるために必要。
    static var symbolName:String = "Dial";
    static var symbolOwner:Object = Dial;
    var className:String = "Dial";

    // 指針とダイアルのムービークリップ
    // （このコンポーネントのグラフィカル表現）
    private var needle:MovieClip;
    private var dial:MovieClip;
    private var boundingBox_mc:MovieClip;

    // プライベートメンバー変数 "__value" には、外部からは
    // 暗黙的な getter/setter メソッドでアクセス可能。
    // このプロパティを更新すると、指針の位置が
    // 値の設定時に更新される。
    private var __value:Number = 0;

    // このフラグは、ユーザーがマウスで指針を
    // ドラッグすると設定され、その後クリアされる。
    private var dragging:Boolean = false;
```

```

// コンストラクタ。
// どのクラスでも必須だが、v2 コンポーネントの場合は
// コンストラクタの内容が空で引数がないことが必要。
// すべての初期化処理は、必須の init() メソッド内で、
// クラスインスタンスが作成された後に実行される。
function Dial() {
}

// 初期化コード。
// init() メソッドは v2 コンポーネントに必須。また、ここでは
// super.init() として親クラスの init() を必ず呼び出すこと。
// init() メソッドは、UIComponent を継承するコンポーネントで必須。
function init():Void {
    super.init();
    useHandCursor = false;
    boundingBox_mc._visible = false;
    boundingBox_mc._width = 0;
    boundingBox_mc._height = 0;
}

// 起動時に必要とされる子オブジェクトの作成 :
// createChildren() メソッドは、
// UIComponent を継承するコンポーネントに必須。
public function createChildren():Void {
    dial = createObject("DialFinal", "dial", 10);
    size();
}

// draw() メソッドは v2 コンポーネントに必須。
// このメソッドは、コンポーネントが無効化された
// (invalidate() が呼び出された ) 後に呼び出される。
// これは、value の set() 関数内で再描画するよりよい方法。
// 他のプロパティがある場合でも再描画を
// 1 回でまとめて処理できるので、各変更箇所を
// 個別に再描画するよりも望ましい。この方法により、
// 効率を向上し、コード管理を集中化できる。
function draw():Void {
    super.draw();
    dial.needle._rotation = value;
}

// size() メソッドは、コンポーネントのサイズが変更されると
// 呼び出される。ここで、子についてもサイズを変更する。
// グラフィックのサイズを変更する。
// size() メソッドは、UIComponent を継承するコンポーネントで必須。
function size():Void {
    super.size();
    dial._width = width;
    dial._height = height;
    // 必要に応じて指針の再描画を実行させる。
    invalidate();
}

```

```

// これは value プロパティの getter/setter。
// [Inspectable] メタデータにより、このプロパティは
// プロパティインスペクタに表示される。getter/setter なので、
// 値が変更されたとき invalidate を呼び出してコンポーネントを
// 強制的に再描画させることができる。
[Bindable]
[ChangeEvent("change")]
[Inspectable(defaultValue=0)]
function set value (val:Number)
{
    __value = val;
    invalidate();
}

function get value ():Number
{
    return twoDigits(__value);
}

function twoDigits(x:Number):Number
{
    return (Math.round(x * 100) / 100);
}

// マウスの押下を受け付けるようコンポーネントに指示する。
function onPress()
{
    beginDrag();
}

// ダイアルが押されたとき、ドラッグフラグを設定する。
// マウスイベントにコールバック関数を割り当てる。
function beginDrag()
{
    dragging = true;
    onMouseMove = mouseMoveHandler;
    onMouseUp = mouseUpHandler;
}

// ドラッグが完了したらマウスイベントを削除し、
// フラグをクリアする。
function mouseUpHandler()
{
    dragging = false;
    delete onMouseMove;
    delete onMouseUp;
}

```

```

function mouseMoveHandler()
{
    // 角度を計算する。
    if (dragging) {
        var x:Number = _xmouse - width/2;
        var y:Number = _ymouse - height/2;

        var oldValue:Number = value;
        var newValue:Number = 90+180/Math.PI*Math.atan2(y, x);
        if (newValue<0) {
            newValue += 360;
        }
        if (oldValue != newValue) {
            value = newValue;
            dispatchEvent( {type:"change"} );
        }
    }
}
}

```

Dial コンポーネントのテストと書き出し

ここまでで、グラフィカルエレメントを含む Flash ファイル、基本クラス、および Dial コンポーネントの機能をすべて記述したクラスファイルを作成しました。次は、このコンポーネントをテストします。コンポーネントのテストは、作成作業中、特にクラスファイルの記述作業中に実行できるのが理想的です。作成中に最も手早くテストする方法は、コンポーネントをコンパイル済みクリップに変換し、そのクリップをコンポーネントの FLA ファイルで使用方法です。

コンポーネントの作成作業が完了したら、そのコンポーネントを SWC ファイルとして書き出します。詳細については、[180 ページの「コンポーネントの書き出しと配布」](#)を参照してください。

Dial コンポーネントをテストするには：

1. "Dial.fla" ファイルの [ライブラリ] で Dial コンポーネントを選択し、[ライブラリ] コンテキストメニューを開きます (Windows の場合は右クリック、Mac の場合は Control キーを押しながらクリック)。次に、[コンパイルされたクリップへの変換] を選択します。

コンパイルされたクリップが、Dial SWF という名前でライブラリに追加されます。



作成済みのコンパイルされたクリップが既にある場合 (2 度目、3 度目のテストを実行する場合など) は、[ライブラリのコンフリクトを解決] ダイアログボックスが表示されます。[既存のアイテムを置き換える] を選択して、新しいバージョンをドキュメントに追加します。

2. Dial SWF をメインタイムラインのステージにドラッグします。
3. サイズの変更と value プロパティの設定は、プロパティインスペクタまたは [コンポーネントインスペクタ] パネルで行います。value プロパティを設定すると、それに合わせて指針の位置が変化します。

4. 実行時に value プロパティをテストするには、ダイアルに **dial** というインスタンス名を付けて、メインタイムラインのフレーム 1 に次のコードを追加します。

```
// テキストフィールドの位置
var textXPos:Number = dial.width/2 + dial.x;
var textYPos:Number = dial.height/2 + dial.y;

// dial.value を表示するためのテキストフィールドを作成する
createTextField("dialValue", 10, textXPos, textYPos, 100, 20);

// change イベントを処理するためのリスナーを作成する
function change(evt){
    // 指針が移動したときに value プロパティを
    // テキストフィールドに表示する
    dialValue.text = dial.value;
}
dial.addEventListener("change", this);
```

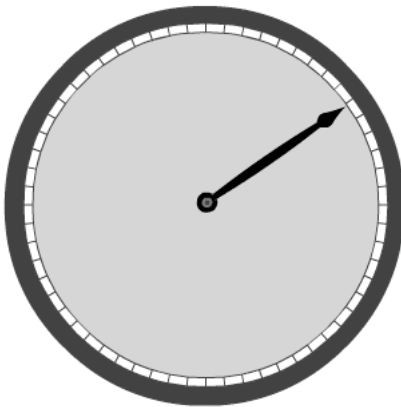
5. [制御]-[ムービープレビュー] を選択して、Flash Player でコンポーネントをテストします。

Dial コンポーネントを書き出すには：

1. "Dial.fla" ファイルの [ライブラリ] で Dial コンポーネントを選択し、[ライブラリ] コンテキストメニューを開きます (Windows の場合は右クリック、Mac の場合は Control キーを押しながらクリック)。次に、[SWC ファイル書き出し] を選択します。
2. SWC ファイルを保存する場所を選択します。

ユーザーレベル設定フォルダ内の "Components" フォルダに保存した場合は、[コンポーネント] パネルをリロードすればコンポーネントが表示され、Flash の再起動は必要ありません。

✕ 閉	フォルダの場所の詳細については、『Flash ユーザーガイド』の「Flash と共にインストールされる設定フォルダ」を参照してください。
--------	--



完成した Dial コンポーネント

親クラスの選択

コンポーネント作成時に最初に決めることは、バージョン 2 クラスのいずれかを継承するかかどうかです。バージョン 2 クラスを継承する場合は、Button、CheckBox、ComboBox、List などのコンポーネントクラスか、基本クラスである UIObject または UIComponent のいずれかを継承できます。Media コンポーネントを除くすべてのコンポーネントクラスは、基本クラスを継承しています。したがって、コンポーネントクラスを継承した場合、クラスは自動的に基本クラスを継承することになります。

2 つの基本クラスは、コンポーネントに必要な共通の機能を提供します。これらのクラスを継承することにより、基本的な各種メソッド、プロパティ、イベントがあらかじめ用意された状態からコンポーネントの作成を開始できます。

バージョン 2 フレームワークでは、UIObject、UIComponent、または他のクラスを作成することは必須ではありません。直接 MovieClip クラスを継承する場合でも、多くの強力なコンポーネントの機能、つまり、SWC ファイルまたはコンパイルされたクリップへの書き出し、ビルトインライブプレビューの使用、Inspectable プロパティの表示などを使用できます。ただし、作成するコンポーネントと Adobe バージョン 2 コンポーネントを併用し、Manager クラスを使用する場合は、UIObject または UIComponent を継承する必要があります。

次の表に、バージョン 2 基本クラスの概要を示します。

基本クラス	親クラス	説明
mx.core.UIObject	MovieClip	UIObject は、グラフィカルなオブジェクトすべての基本クラスです。UIObject は形状を持つことができ、それ自体を表示することも非表示にすることもできます。 UIObject は、次のような機能を備えています。 <ul style="list-style-type: none">• スタイルの編集• イベントの処理• 伸縮によるサイズの変更
mx.core.UIComponent	UIObject	UIComponent は、すべてのコンポーネントの基本クラスです。 UIComponent は、次のような機能を備えています。 <ul style="list-style-type: none">• フォーカスナビゲーションの作成• タブ順序の作成• コンポーネントの有効化と無効化• コンポーネントのサイズ変更• 低レベルのマウスイベントおよびキーボードイベントの処理

UIObject クラスについて

Adobe Component Architecture バージョン 2 に基づくコンポーネントは、UIObject クラスを継承します。UIObject は MovieClip クラスのサブクラスです。MovieClip クラスは、Flash 内のビジュアルオブジェクトを表現するすべてのクラスの基本クラスです。

UIObject には、スタイルとイベントを処理するためのメソッドが追加されています。UIObject では、描画の直前 (draw イベント。MovieClip.onEnterFrame イベントに相当)、ロードおよびアンロード時 (load および unload)、レイアウト変更時 (move および resize)、オブジェクトの非表示および表示時 (hide および reveal) に、リスナーに対してイベントが送出されます。

UIObject には、コンポーネントの位置とサイズを決定する読み取り専用の代替変数 (width、height、x、y) と、オブジェクトの位置およびサイズを変更するための move() メソッドおよび setSize() メソッドがあります。

UIObject クラスは、次の機能を実装します。

- スタイル
- イベント
- 伸縮によるサイズの変更

詳細については、『ActionScript 2.0 コンポーネントリファレンスガイド』を参照してください。

UIComponent クラスについて

UIComponent クラスは UIObject のサブクラスです (『ActionScript 2.0 コンポーネントリファレンスガイド』の「UIComponent クラス」を参照)。また、ユーザーの対話操作 (マウス入力とキーボード入力) を扱うコンポーネントすべての基本クラスです。UIComponent クラスのコンポーネントでは次の処理ができます。

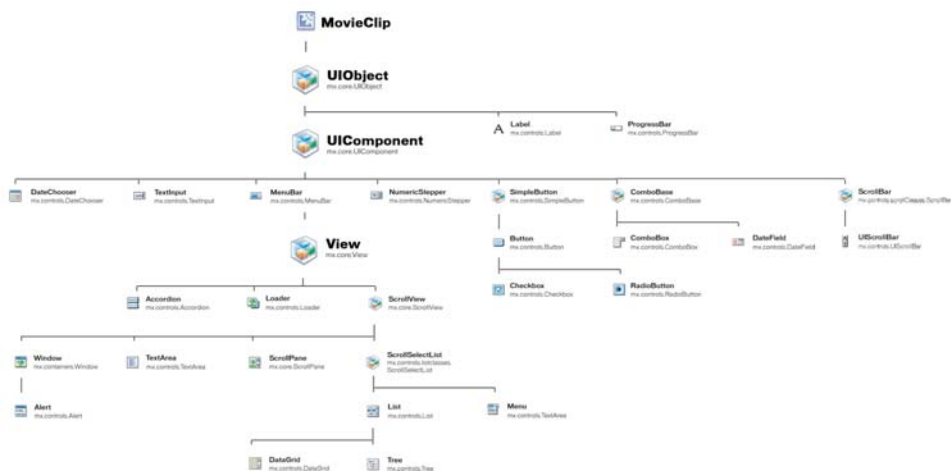
- フォーカスとキーボード入力の受け付け
- コンポーネントの有効化と無効化
- レイアウトによるサイズの変更

他のバージョン 2 クラスの継承について

コンポーネントを容易に構築できるよう、任意のクラスを継承できます。直接 UIObject クラスまたは UIComponent クラスを継承する必要はありません。Media コンポーネント以外のバージョン 2 コンポーネントのクラスを継承すると、自動的に UIObject および UIComponent も継承したことになります。コンポーネント辞書の一覧にあるコンポーネントクラスはすべて、継承して新しいコンポーネントクラスを作成できます。

たとえば、Button コンポーネントとほとんど同様に動作するコンポーネントを作成する場合、Button クラスを継承すればよく、Button クラスに相当するすべての機能を自力で基本クラスから作成する必要はありません。

次の図は、バージョン 2 コンポーネントの階層を示しています。



バージョン 2 コンポーネントの階層構造

"arch_diagram.swf" ファイルの FlashPaper バージョンのサンプルについては、Web サイトの Flash サンプルページ (www.adobe.com/go/learn_fl_tutorials_jp) を参照してください。

MovieClip クラスの継承について

コンポーネント作成時には、バージョン 2 クラスではなく、ActionScript MovieClip クラスを直接継承することもできます。ただし、UIObject や UIComponent に用意されている何らかの機能が必要になっても、その機能は自分で作成する必要があります。UIObject クラスおよび UIComponent クラスを開き (First Run/classes/mx/core)、それらがどのように構築されたかを調べることができます。

コンポーネントムービークリップの作成

コンポーネントを作成するには、必ずムービークリップシンボルを作成し、コンポーネントのクラスファイルにリンクする必要があります。

このムービークリップには、2 つのフレームと 2 つのレイヤーがあります。最初のレイヤーは Actions レイヤーで、フレーム 1 に stop() グローバル関数があります。もう 1 つのレイヤーは Assets レイヤーで、2 つのキーフレームがあります。フレーム 1 には、最終的なデザインのプレースホルダとなる境界ボックスやグラフィックが含まれています。フレーム 2 には、コンポーネントが使用するその他のアセット (グラフィックや基本クラスなど) がすべて含まれています。

新しいムービークリップシンボルの挿入

すべてのコンポーネントは MovieClip オブジェクトです。新しいコンポーネントを作成するには、まず、新しい FLA ファイルに新しいシンボルを挿入する必要があります。

新しいコンポーネントシンボルを追加するには：

1. Flash で、空の Flash ドキュメントを作成します。
2. [挿入]-[新規シンボル] を選択します。
[新規シンボルの作成] ダイアログボックスが表示されます。
3. シンボル名を入力します。コンポーネントに名前を付けます。コンポーネント名に含まれる各単語の先頭は大文字にします (たとえば、MyComponent のように)。
4. ムービークリップのビヘイビアを選択します。
5. [詳細] ボタンをクリックして、詳細な設定を表示します。
6. [ActionScript に書き出し] をオンに、[最初のフレームに書き出し] をオフにします。
7. リンケージ識別子を入力します。
8. [AS 2.0 クラス] テキストボックスに、ActionScript 2.0 のクラスへの完全修飾パスを入力します。
クラス名は、[コンポーネント] パネルに表示されるコンポーネント名と同じにする必要があります。たとえば、Button コンポーネントのクラスは mx.controls.Button です。



ファイル名の拡張子をパスに含めないでください。[AS 2.0 クラス] テキストボックスでは、ファイルシステム上にあるファイルとしての名前を指定するのではなく、パッケージ化されたクラスとしての場所を指定します。

目的の ActionScript ファイルがパッケージ内にある場合は、パッケージ名を含める必要があります。この値には、クラスパスからの相対パスか、パッケージの絶対パス (例: "mypackage.MyComponent") を指定します。

9. ほとんどの場合、[最初のフレームに書き出し] は選択解除する必要があります (デフォルトでは選択されています)。詳細については、[185 ページの「コンポーネント開発チェックリスト」](#)を参照してください。
10. [OK] をクリックします。

シンボルがライブラリに追加され、シンボル編集モードに切り替わります。このモードでは、ステージの左上隅の上にシンボル名が表示され、シンボルの基準点が十字カーソルで示されます。

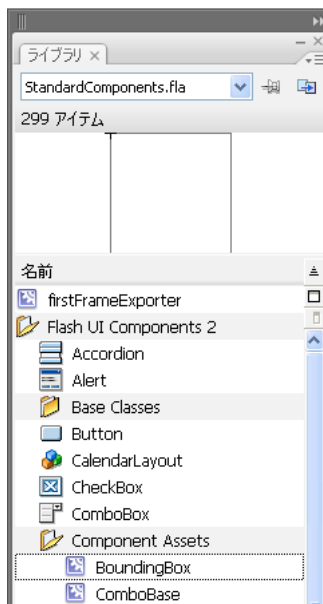
ムービークリップの編集

新しいシンボルを作成してそのシンボルにリンケージを定義すると、シンボルのタイムライン内でコンポーネントのアセットを定義できるようになります。

コンポーネントのシンボルには、2つのレイヤーが必要です。このセクションでは、どのようなレイヤーを挿入し、それらのレイヤーに何を追加するのかを説明します。

ムービークリップを編集するには：

1. レイヤー1の名前を **Actions** に変更し、フレーム1を選択します。
2. [アクション] パネルを開いて次のように入力し、`stop()` 関数を追加します。
`stop();`
このフレームには、グラフィカルアセットを追加しないでください。
3. レイヤーを追加し、**Assets** という名前を付けます。
4. Assets レイヤーのフレーム2を選択し、空白のキーフレームを挿入します。
これで、このレイヤーには空白のキーフレームが2つあることになります。
5. 次のいずれかの操作を行います。
 - コンポーネントに境界エリアを定義するビジュアルアセットがある場合は、そのシンボルをフレーム1にドラッグし、適切なレイアウトに調整します。
 - すべてのビジュアルアセットを実行時に作成するコンポーネントの場合は、**BoundingBox** シンボルをフレーム1のステージにドラッグして、適切なサイズに変更し、**boundingBox_mc** というインスタンス名を付けます。このシンボルは、"First Run/ComponentFLA" フォルダにある "StandardComponents.flc" ファイルのライブラリにあります。



6. 既存のコンポーネントを継承する場合は、そのコンポーネントとそのコンポーネントが継承する基本クラスのインスタンスを **Assets** レイヤーのフレーム 2 に配置します。

これを行うには、[コンポーネント] パネルからシンボルを選択し、ステージにドラッグします。基本クラスを継承する場合は、"Configuration/ComponentFLA" フォルダから "StandardComponents.flc" ファイルを開き、ライブラリからそのクラスをステージにドラッグします。

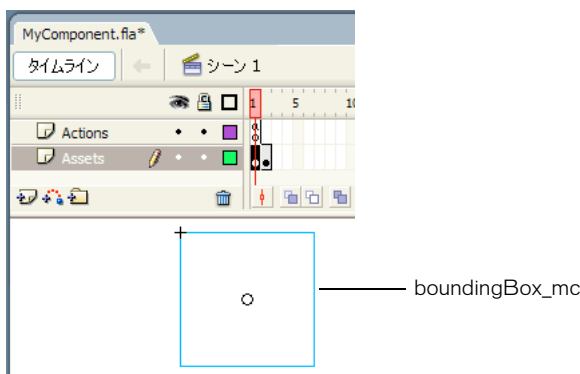


UIComponent をコンポーネントのライブラリにドラッグすると、ライブラリ内のフォルダ階層が変化します。ライブラリを再利用する場合や、他のコンポーネントのグループ (バージョン 2 コンポーネントなど) とライブラリを共用する場合は、整理してシンボルの重複を防ぐために、"StandardComponents.flc" のライブラリに合わせてフォルダ階層構造を再構成してください。

7. このコンポーネントで使用するグラフィカルアセットを、コンポーネントの **Assets** レイヤーのフレーム 2 に追加します。

コンポーネントで使用するすべてのアセット (別のコンポーネントやビットマップなどのメディアも含む) について、**Assets** レイヤーのフレーム 2 にそのインスタンスを配置する必要があります。

8. 完成したシンボルは次のようなものになります。



コンポーネントとしてのムービークリップの定義

ムービークリップシンボルは、[コンポーネント定義] ダイアログボックスで **ActionScript クラスファイル** とリンクする必要があります。Flash では、この関連付けによってコンポーネントのメタタグを検索する場所が決定されます。メタタグの詳細については、[149 ページの「コンポーネントメタデータの追加」](#)を参照してください。[コンポーネント定義] ダイアログボックスでは、これ以外のオプションも選択できます。

ムービークリップをコンポーネントとして定義するには：

1. ライブラリでムービークリップを選択し、[ライブラリ]コンテキストメニューから[コンポーネント定義]を選択します (Windows の場合は右クリック、Mac の場合は Control キーを押しながらクリック)。
2. [AS 2.0 クラス] を入力します。
パッケージに含まれているクラスの場合は、パッケージ名全体を入力します。
3. 必要に応じて、[コンポーネント定義]ダイアログボックスで他のオプションを指定します。
 - プラス (+) ボタンをクリックしてパラメータを定義します。
この操作は省略可能です。パラメータを指定するには、コンポーネントのクラスファイルに `Inspectable` メタデータタグを挿入するのが最良の方法です。ActionScript 2.0 クラスが指定されていない場合は、コンポーネントのパラメータをここで定義します。
 - [カスタム UI] を指定します。
これは [コンポーネントインスペクタ] パネルに表示される SWF ファイルです。このファイルはコンポーネントの FLA ファイルに埋め込むことも、外部 SWF ファイルを参照することもできます。
 - [ライブプレビュー] を指定します。
これは外部または埋め込みの SWF ファイルです。ここでライブプレビューを指定する必要はありません。コンポーネントムービークリップに境界ボックスを追加すれば、ライブプレビューが自動的に作成されます。[138 ページの「コンポーネントムービークリップの作成」](#)を参照してください。
 - 説明を入力します。
[リファレンス] パネルが Flash MX 2004 で削除されたため、[説明] フィールドの利用価値は下がっています。このフィールドは、FLA ファイルを Flash MX 形式で保存するときの後方互換性を維持するために提供されています。
 - アイコンを選択します。
コンポーネントのアイコンとして使用する PNG ファイルを指定します。ActionScript 2.0 クラスファイルで `IconFile` メタデータタグを指定している場合 (最良の方法)、このフィールドは無視されます。
 - [インスタンスのパラメータはロックされています] をオンまたはオフにします。
これをオフにすると、コンポーネントの各インスタンスごとに、コンポーネントのパラメータと異なる別のパラメータをユーザーが追加できます。通常、この設定はオンにしておいてください。このオプションは、Flash MX との後方互換性を維持するためのものです。
 - [コンポーネント] パネルに表示されるツールヒントを指定します。

ActionScript クラスファイルの作成

すべてのコンポーネントシンボルは、ActionScript 2.0 クラスファイルにリンクされています。リンクについては、[138 ページの「コンポーネントムービークリップの作成」](#)を参照してください。

ActionScript クラスファイルの編集には、Flash だけでなく、任意のテキストエディタや任意の統合開発環境 (IDE) を使用できます。

外部 ActionScript クラスでは、他のクラス (バージョン 2 コンポーネント、バージョン 2 基本クラス、または ActionScript MovieClip クラス) を継承します。作成するコンポーネントと最も近い機能を提供するクラスを継承するようにしてください。継承 (拡張) できる親クラスは 1 つだけです。ActionScript 2.0 では多重継承は許されません。

コンポーネントクラスファイルの簡単な例

次に示す例は、"MyComponent.as" という名前の簡単なクラスファイルです。このコンポーネントを作成する際は、このファイルを Flash のコンポーネントムービークリップにリンクします。

この例には、UIComponent クラスを継承する MyComponent というコンポーネントについて、読み込み、メソッド、宣言の最小限のセットが含まれています。この "MyComponents.as" ファイルは、"myPackage" フォルダに保存されます。

```
[Event("eventName")]

// パッケージを読み込む。
import mx.core.UIObject;

// クラスを宣言し、親クラスを継承する。
class mypackage.MyComponent extends UIObject {

    // このクラスにバインドするシンボル名を特定する。
    static var symbolName:String = "mypackage.MyComponent";

    // シンボル所有者の完全修飾パッケージ名を指定する。
    static var symbolOwner:Object = Object(mypackage.MyComponent);

    // 変数 className を宣言する。
    var className:String = "MyComponent";

    // 空のコンストラクタを定義する。
    function MyComponent() {
    }

    // 親クラスの init() メソッドを呼び出し、
    // 境界ボックスを非表示にする
    // ( オーサリング時にのみ使用するため )。
    function init():Void {
        super.init();
    }
}
```

```

        boundingBox_mc.width = 0;
        boundingBox_mc.height = 0;
        boundingBox_mc.visible = false;
    }

    function createChildren():Void{
        // createClassObject を呼び出してサブオブジェクトを作成する。
        size();
        invalidate();
    }

    function size(){
        // サイズ変更を処理するコードを記述する。
        super.size();
        invalidate();
    }

    function draw(){
        // 視覚的表現を処理するコードを記述する。
        super.draw();
    }
}

```

コンポーネントのクラスファイルの概要

次の手順は、コンポーネントクラスの `ActionScript` ファイルを作成する方法について説明しています。作成するコンポーネントのタイプによっては省略できる手順もあります。

コンポーネントのクラスファイルを記述するには：

1. (オプション) クラスを読み込みます。詳細については、[146 ページの「クラスの読み込み」](#)を参照してください。

これにより、パッケージ名をすべて記述しなくてもクラスを参照できるようになります。たとえば、`mx.controls.Button` と書く代わりに `Button` と記述できます。
2. `class` キーワードを使用してクラスを定義します。親クラスを継承する場合は、`extend` キーワードを使用します。詳細については、[146 ページの「クラスおよびサブクラスの定義」](#)を参照してください。
3. 変数 `symbolName`、`symbolOwner`、`className` を定義します。詳細については、[146 ページの「クラス名、シンボル名、および所有者名の指定」](#)を参照してください。

これらの変数は、バージョン 2 コンポーネントでのみ必要です。
4. メンバー変数を定義します。詳細については、[147 ページの「変数の定義」](#)を参照してください。

これらの変数は `getter/setter` メソッドで使用できます。

5. コンストラクタ関数を定義します。詳細については、[163 ページの「コンストラクタ関数について」](#)を参照してください。

6. `init()` メソッドを定義します。詳細については、[161 ページの「`init\(\)` メソッドの定義」](#)を参照してください。

UIComponent を継承するクラスでは、クラスが作成される際にこのメソッドが呼び出されます。MovieClip を継承するクラスの場合は、コンストラクタでこのメソッドを呼び出します。

7. `createChildren()` メソッドを定義します。詳細については、[161 ページの「`createChildren\(\)` メソッドの定義」](#)を参照してください。

UIComponent を継承するクラスでは、クラスが作成される際にこのメソッドが呼び出されます。MovieClip を継承するクラスの場合は、コンストラクタでこのメソッドを呼び出します。

8. `size()` メソッドを定義します。詳細については、[164 ページの「`size\(\)` メソッドの定義」](#)を参照してください。

UIComponent を継承するクラスでは、コンポーネントのサイズが変更されると、このメソッドが呼び出されます。また、オーサリング時にコンポーネントのライブプレビューをサイズ変更すると呼び出されます。

9. `draw()` メソッドを定義します。詳細については、[165 ページの「無効化について」](#)を参照してください。

UIComponent を継承するクラスでは、コンポーネントが無効化されるとこのメソッドが呼び出されます。

10. Metadata タグと宣言を追加します。詳細については、[149 ページの「コンポーネントメタデータの追加」](#)を参照してください。

タグと宣言の追加により、getter および setter プロパティが、Flash のプロパティインスペクタと [コンポーネントインスペクタ] パネルに表示されるようになります。

11. getter/setter メソッドを定義します。詳細については、[148 ページの「getter/setter メソッドを使用したパラメータの定義」](#)を参照してください。

12. (オプション) コンポーネント内で使用するすべてのスキンエレメントおよびリンケージの変数を作成します。詳細については、[167 ページの「スキンの割り当てについて」](#)を参照してください。
これによって、ユーザーがコンポーネントのパラメータを変更して別のスキンエレメントを設定できるようになります。

クラスの読み込み

クラスファイルを読み込むと、コード内の各所に完全修飾名でクラス名を記述する必要がなくなります。それにより、より簡潔で読みやすいコードを記述できます。クラスを読み込むには、クラスファイルの冒頭で `import` ステートメントを使用します。次のように記述します。

```
import mx.core.UIObject;
import mx.core.ScrollView;
import mx.core.ext.UIObjectExtensions;
```

```
class MyComponent extends UIComponent{
```

また、ワイルドカード文字 (*) を使用して、特定のパッケージ内のすべてのクラスを読み込むこともできます。たとえば、次のステートメントを使用すると、`mx.core` パッケージ内のクラスがすべて読み込まれます。

```
import mx.core.*;
```

読み込んででもスクリプト内で使用しないクラスは、最終的な SWF ファイルのバイトコードには含まれません。結果として、ワイルドカードを使用してパッケージ全体を読み込んだ場合でも、必要以上に大きな SWF ファイルは作成されません。

クラスおよびサブクラスの定義

コンポーネントクラスファイルの定義は、他のクラスファイルと同様に行います。クラス名を示すには、`class` キーワードを使用します。クラス名は、クラスファイル名を兼ねている必要があります。スーパークラス名を示すには、`extends` キーワードを使用します。詳細については、『ActionScript 2.0 の学習』の第 6 章の「カスタムクラスファイルの記述」を参照してください。

```
class MyComponentName extends UIComponent{
}
```

クラス名、シンボル名、および所有者名の指定

Flash が正しい ActionScript クラスおよびパッケージを見つけられるように、また、コンポーネントのネーミングが保たれるように、作成したコンポーネントの ActionScript クラスファイル内では、変数 `symbolName`、`symbolOwner`、および `className` を必ず設定してください。

変数 `symbolOwner` は、シンボルを表す `Object` 型の参照です。コンポーネントが自分自身の `symbolOwner` である場合や、`symbolOwner` が読み込まれている場合は、完全修飾名で指定する必要はありません。

これらの変数について次の表で説明します。

変数	データ型	説明
symbolName	String	ActionScript クラスの名前 (例 : ComboBox)。 この名前は、シンボルのリンケージ識別子と一致させる必要があります。 この変数は静的変数として定義する必要があります。
symbolOwner	Object	完全修飾クラス名 (例 : mypackage.MyComponent)。 symbolOwner 値は Object 型の変数なので、値は引用符で囲まずに指定します。 この名前は、[リンケージプロパティ] ダイアログボックスの [AS 2.0 クラス] と一致させる必要があります。 この変数は、createClassObject() メソッドへの内部的な呼び出しで使用されます。 この変数は静的変数として定義する必要があります。
className	String	コンポーネントクラス名。これにパッケージ名は含まれません。Flash 開発環境の中に、これに対応する設定はありません。 この変数の値は、スタイルプロパティを設定するときに使用できます。

次の例では、変数 symbolName、symbolOwner、className を MyButton クラスに追加しています。

```
class MyButton extends mx.controls.Button {
    static var symbolName:String = "MyButton";
    static var symbolOwner = myPackage.MyButton;
    var className:String = "MyButton";
}
```

変数の定義

次のコードは、"Button.as" ファイル (mx.controls.Button) からの抜粋です。クラスファイル内で使用する変数 btnOffset を定義しています。また、変数 __label と __labelPlacement も定義しています。変数 __label および __labelPlacement は名前の先頭に 2 つのアンダースコアがついています。これは、getter/setter メソッド内で使用されるときや、最終的にコンポーネント内のプロパティおよびパラメータとして使用されるときに、名前の競合が起こらないようにするためです。詳細については、『ActionScript 2.0 の学習』の「getter/setter メソッドを使用したパラメータの定義」を参照してください。

```
/**
 * ボタンが押されたときラベルやアイコンの表示に使用するオフセットの数値
 */
var btnOffset:Number = 0;

/**
 * @private
 * 値が指定されていないときにラベルに表示されるテキスト
 */
```

```

    var __label:String = "default value";

/**
 * @private
 * デフォルトのラベル位置
 */
    var __labelPlacement:String = "right";

```

getter/setter メソッドを使用したパラメータの定義

コンポーネントのパラメータを定義するには、パラメータを "inspectable" にするパブリックメンバ変数を追加するのが最も簡単な方法です。この操作を実行するには、コンポーネントインスペクタで `Inspectable` タグを使用するか、次のとおり `Inspectable` 変数を追加します。

```

[Inspectable(defaultValue="strawberry")]
public var flavorStr:String;

```

ただし、コンポーネントを使用するコードで `flavorStr` プロパティが変更された場合、そのコンポーネントでは通常、プロパティの変更に反応してそれ自身を更新するためのアクションを実行する必要があります。たとえば、`flavorStr` が "cherry" に設定された場合、このコンポーネントでは、それ自身を再描画してデフォルトのイチゴのイメージからサクランボのイメージに変更します。

通常のメンバ変数の値が変更された場合は、コンポーネントに自動的に通知されることはありません。

`getter/setter` メソッドは、コンポーネントのプロパティに対する変更を検知するためのわかりやすい方法です。通常の変数は `var` を使用して宣言するのに対し、`getter/setter` メソッドの宣言は次のように記述します。

```

private var __flavorStr:String = "strawberry";

[Inspectable(defaultValue="strawberry")]

public function get flavorStr():String{
    return __flavorStr;
}
public function set flavorStr(newFlavor:String) {
    __flavorStr = newFlavor;
    invalidate();
}

```

`invalidate()` を呼び出すことにより、コンポーネントが新しいフレーバーで自身を再描画します。これが、`flavorStr` プロパティに通常のメンバ変数ではなく `getter/setter` メソッドを使用することのメリットです。[164 ページの「draw\(\) メソッドの定義」](#)を参照してください。

getter/setter メソッドを定義するときは、次の点に注意してください。

- メソッド名の先頭は `get` または `set` で開始し、1 つスペースを空けてからプロパティ名を続けます。
- プロパティの値を格納する変数に `getter` または `setter` と同じ名前を付けることはできません。慣例として、`getter` および `setter` の変数名には、先頭が 2 つのアンダースコアで始まる名前を使用します。

通常 `getter` および `setter` は、プロパティインスペクタおよび [コンポーネントインスペクタ] パネルに表示されるプロパティを定義するためのタグと組み合わせて使用されます。

`getter/setter` メソッドの詳細については、『*ActionScript 2.0 の学習*』の「`getter` メソッドと `setter` メソッドについて」を参照してください。

コンポーネントメタデータの追加

外部 *ActionScript* クラスファイルにコンポーネントメタデータタグを追加することで、コンポーネントのパラメータ、データバインディングプロパティ、イベントについて、情報をコンパイラに伝えることができます。メタデータタグは、Flash オーサリング環境においてさまざまな目的で使用されます。

メタデータタグは、外部 *ActionScript* クラスファイル内でのみ使用できます。FLA ファイル内ではメタデータタグを使用できません。

メタデータは、クラス宣言または個々のデータフィールドに関連付けられます。属性の値が `String` である場合は、その属性を引用符で囲む必要があります。

メタデータステートメントは、*ActionScript* ファイル内でその次にある行にバインドされます。コンポーネントプロパティを定義するときは、プロパティ宣言の直前の行にメタデータタグを追加してください。唯一の例外は `Event` メタデータタグです。コンポーネントイベントを定義するときは、クラス定義の外側にメタデータタグを追加してください。こうすることで、イベントがクラス全体にバインドされます。

次の例では、`Inspectable` タグにより `flavorStr` パラメータ、`colorStr` パラメータ、および `shapeStr` パラメータが定義されます。

```
[Inspectable(defaultValue="strawberry")]
public var flavorStr:String;
[Inspectable(defaultValue="blue")]
public var colorStr:String;
[Inspectable(defaultValue="circular")]
public var shapeStr:String;
```

これらのパラメータは、プロパティインスペクタ、および [コンポーネントインスペクタ] パネルの [パラメータ] タブに、すべて `String` 型として表示されます。

メタデータタグ

ActionScript クラスファイル内で使用できるメタデータタグを、次の表に示します。

タグ	説明
Inspectable	プロパティを [コンポーネントインスペクタ] パネルとプロパティインスペクタに公開します。 150 ページの「Inspectable タグについて」 を参照してください。
InspectableList	プロパティインスペクタおよび [コンポーネントインスペクタ] パネルのリストに表示する Inspectable プロパティのサブセットを指定します。コンポーネントのクラスに InspectableList 属性を追加しないと、プロパティインスペクタには Inspectable パラメータがすべて表示されます。 153 ページの「InspectableList タグについて」 を参照してください。
イベント	コンポーネントイベントを定義します。 153 ページの「Event タグについて」 を参照してください。
Bindable	[コンポーネントインスペクタ] パネルの [バインディング] タブにプロパティを表示します。 154 ページの「Bindable タグについて」 を参照してください。
ChangeEvent	データバインディングを実行させるイベントを指定します。 155 ページの「ChangeEvent タグについて」 を参照してください。
Collection	[コンポーネントインスペクタ] パネルに表示する collection 属性を識別します。 156 ページの「Collection タグについて」 を参照してください。
IconFile	[コンポーネント] パネル内でこのコンポーネントを表すアイコンのファイル名を指定します。 157 ページの「IconFile タグについて」 を参照してください。
ComponentTask	1つ以上の関連する JSFL ファイルのファイル名を指定し、オーサリング環境でタスクを実行します。 158 ページの「ComponentTask タグについて」 を参照してください。

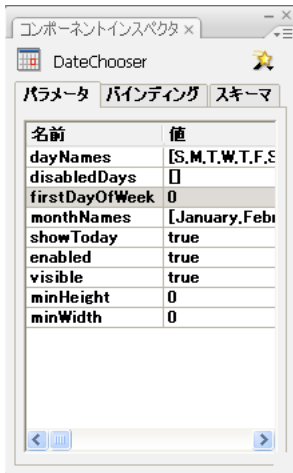
コンポーネントのメタデータタグについては、次のセクションでさらに詳しく説明します。

Inspectable タグについて

Inspectable タグを使用すると、ユーザーによる編集が可能な (または "inspectable" つまり検査可能な) パラメータを指定し、そのパラメータを [コンポーネントインスペクタ] パネルおよびプロパティインスペクタに表示できます。これにより、Inspectable プロパティとそれを実装する ActionScript コードを同じ場所で保守できます。コンポーネントプロパティを表示するには、コンポーネントのインスタンスをステージにドラッグして、[コンポーネントインスペクタ] パネルの [パラメータ] タブを選択します。

Collection パラメータも Inspectable です。詳細については、[156 ページの「Collection タグについて」](#)を参照してください。

次の図では、DateChooser コンポーネントの場合について [コンポーネントインスペクタ] パネルの [パラメータ] タブを示しています。



また、プロパティインスペクタの [パラメータ] タブには、コンポーネントプロパティのサブセットが表示されます。

オーサリング環境に表示されるパラメータは、Inspectable タグに基づいて決定されます。このタグのシンタックスは次のとおりです。

```
[Inspectable(value_type=value[,attribute=value,...])]  
property_declaration name:type;
```

次の例では、enabled パラメータを Inspectable として定義しています。

```
[Inspectable(defaultValue=true, verbose=1, category="Other")]  
var enabled:Boolean;
```

また、Inspectable タグでは、型付けの厳格でない次のような属性もサポートしています。

```
[Inspectable("danger", 1, true, maybe)]
```

メタデータステートメントをプロパティにバインドするには、そのプロパティの変数宣言の直前に配置する必要があります。

次の表に、Inspectable タグの属性を示します。

属性	データ型	説明
defaultValue	String または Number	(オプション)Inspectable プロパティのデフォルト値。
enumeration	String	(オプション) そのプロパティで許容される正当な設定値のカンマ区切りリストを指定します。
listOffset	Number	(オプション) Flash MX コンポーネントとの後方互換性のために追加されています。List 値へのデフォルトのインデックスとして使用されます。
name	String	(オプション) プロパティの名前。たとえば " フォントの幅 " のように指定します。指定しないと、_fontWidth などのプロパティ名が使用されます。
type	String	(オプション) タイプ指定子。省略すると、プロパティのタイプが使用されます。許容値は次のとおりです。 <ul style="list-style-type: none">• Array• Boolean• Color• Font Name• List• 数値 (Number)• Object• String
variable	String	(オプション) Flash MX コンポーネントとの後方互換性のために追加されています。このパラメータをバインドする対象の変数を指定します。
verbose	Number	(オプション) verbose 属性を 1 に設定すると、その Inspectable プロパティはプロパティインスペクタには表示されませんが、[コンポーネントインスペクタ] パネルには表示されます。この属性は、あまり頻繁に変更されることがないプロパティによく使用されます。

これらの属性に必須のものはありません。メタデータタグとして Inspectable を使用できます。

親クラスで Inspectable タグが付けられているすべてのプロパティは、継承したクラスでも自動的に Inspectable となります。そうしたプロパティの一部を継承クラスで非表示にするには、InspectableList タグを使用します。

InspectableList タグについて

InspectableList タグを使用すると、Inspectable プロパティのうちプロパティインスペクタに表示するサブセットを指定できます。サブクラス化したコンポーネントで、親クラスから継承した属性を隠蔽するには、InspectableList と Inspectable とを組み合わせで使用します。コンポーネントのクラスに InspectableList タグを追加しないと、Inspectable パラメータがすべて (コンポーネントの親クラスの Inspectable パラメータも含め) プロパティインスペクタに表示されます。

InspectableList のシンタックスは次のとおりです。

```
[InspectableList("attribute1"[,...])]
// クラス定義
```

InspectableList タグは、クラス全体に適用されるため、クラス定義の直前に置く必要があります。

次の例のようにコーディングすると、flavorStr プロパティおよび colorStr プロパティがプロパティインスペクタに表示されます。親クラスから継承した他の Inspectable プロパティは除外されます。

```
[InspectableList("flavorStr","colorStr")]
class BlackDot extends DotParent {
    [Inspectable(defaultValue="strawberry")]
    public var flavorStr:String;
    [Inspectable(defaultValue="blue")]
    public var colorStr:String;
    ...
}
```

Event タグについて

Event タグを使用すると、コンポーネントで発生するイベントを定義できます。

このタグのシンタックスは次のとおりです。

```
[Event("event_name")]
```

たとえば、click イベントを定義するには、次のようなコードを使用します。

```
[Event("click")]
```

クラスの特定のメンバーではなくクラス全体にイベントをバインドするには、ActionScript ファイル内のクラス定義の外部に Event ステートメントを追加します。

次に、UIObject クラス用の Event メタデータの例を示します。この例では、resize イベント、move イベント、および draw イベントが処理されます。

```
...
import mx.events.UIEvent;
[Event("resize")]
[Event("move")]
[Event("draw")]
class mx.core.UIObject extends MovieClip {
    ...
}
```

特定のインスタンスをブロードキャストするには、dispatchEvent() メソッドを呼び出します。[165 ページの「dispatchEvent\(\) メソッドの使用」](#)を参照してください。

Bindable タグについて

コンポーネントは、データバインディングによって相互に接続されます。データバインディングを視覚的に行うには、[コンポーネントインスペクタ] パネルの [バインディング] タブを使用します。ここではコンポーネントのバインディングの追加、表示、および削除を実行できます。

データバインディングはどのようなコンポーネントでも使用できますが、主な用途は、ユーザーインターフェイスコンポーネントを外部データソース (Web サービスや XML ドキュメントなど) に接続することです。これらのデータソースは、プロパティを備えたコンポーネントとして使用できます。これらのプロパティは、他のコンポーネントのプロパティにバインドすることができます。

プロパティを [コンポーネントインスペクタ] パネルの [バインディング] タブに表示するには、ActionScript クラス内のプロパティの前に Bindable タグを使用します。var や getter/setter メソッドを使用してプロパティを宣言できます。getter および setter 両方のメソッドがあるプロパティの場合は、いずれか一方のメソッドに [Bindable] タグを適用すれば済みます。

Bindable タグのシンタックスは次のとおりです。

```
[Bindable "readonly"|"writeonly",type="datatype"]
```

属性はいずれもオプションです。

次の例では、flavorStr 変数を [コンポーネントインスペクタ] パネルの [バインディング] タブでアクセス可能なプロパティとして定義しています。

```
[Bindable]
public var flavorStr:String = "strawberry";
```

Bindable タグは、プロパティへのアクセスの種類、およびそのプロパティのデータ型を指定する 3 つのオプションを取ります。それらのオプションを次の表に示します。

オプション	説明
readonly	[コンポーネントインスペクタ] パネルでバインディングを作成するときに、このプロパティをソースとして使用するバインディングのみを作成できることを示します。ただし、 ActionScript を使用してバインディングを作成する場合は、そのような制限はありません。 [Bindable("readonly")]
writable	[コンポーネントインスペクタ] パネルでバインディングを作成するときに、このプロパティをバインディング先としてのみ使用できることを示します。ただし、 ActionScript を使用してバインディングを作成する場合は、そのような制限はありません。 [Bindable("writable")]
type="datatype"	データバインディングでプロパティにどの型を使用するのかを示します。 Flash の他の機能では、宣言された型を使用します。 このオプションを指定しないと、 ActionScript コード内に宣言されているプロパティのデータ型が、データバインディングに使用されます。 次の例では、x の実際の型は Object ですが、データバインディングでは DataProvider 型として扱うように指定しています。 [Bindable(type="DataProvider")] var x: Object;

すべてのコンポーネントのすべてのプロパティはデータバインディングに使用できます。**Bindable** タグは、どのプロパティをバインディング用に [コンポーネントインスペクタ] パネルに表示するかを制御するものに過ぎません。プロパティの前に **Bindable** タグを付けなくても、そのプロパティをデータバインディングで使用することはできますが、その場合はバインディングの作成に **ActionScript** を使用する必要があります。

ChangeEvent タグを使用する場合は、**Bindable** タグが必要です。

Flash オーサリング環境でのデータバインディングの作成については、『**Flash ユーザーガイド**』の「データバインディング」を参照してください。

ChangeEvent タグについて

ChangeEvent タグは、コンポーネントで特定のプロパティの値が変化するたびにイベントが発生するというのを、データバインディング処理に伝えるために使用します。データバインディング処理では、このイベントに反応して、該当するプロパティをソースとするすべてのバインディングが実行されます。ただし、そのコンポーネント内に適切な **ActionScript** コードを記述しなければイベントは発生しません。使用するイベントは、クラスで宣言される **Event** メタデータのリストに記載されている必要があります。

var や getter/setter メソッドを使用してプロパティを宣言できます。getter および setter 両方のメソッドがあるプロパティの場合は、いずれか一方のメソッドに `ChangeEvent` タグを適用すれば済みます。

`ChangeEvent` タグのシンタックスは次のとおりです。

```
[Bindable]
[ChangeEvent("event")]
property_declaration or getter/setter function
```

次の例では、バインド可能な `flavorStr` プロパティの値が変わった場合、コンポーネントが `change` イベントを発生させます。

```
[Bindable]
[ChangeEvent("change")]
public var flavorStr:String;
```

メタデータに指定されたイベントが発生すると、プロパティの変更がバインディングに対して通知されます。

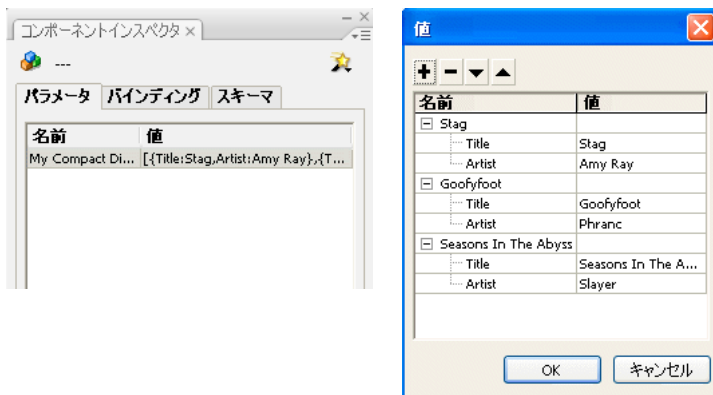
次の例に示すように、複数のイベントをタグ内に登録することもできます。

```
[ChangeEvent("change1", "change2", "change3")]
```

このプロパティに対する変更は、これらのイベントのいずれかによって示されます。これらのイベントがすべて発生する必要はありません。

Collection タグについて

`Collection` タグは、オブジェクトの配列を表現するために使用します。オーサリング時に [値] ダイアログボックスでアイテムを追加、削除、および変更できます。オブジェクトのタイプは、`collectionItem` 属性で指定します。コレクションプロパティには、別のクラスで定義したコレクションアイテム群が含まれます。このクラスとは、`mx.utils.CollectionImpl` クラス、またはそのサブクラスです。個々のオブジェクトに対しては、`collectionClass` 属性で指定するクラスのメソッドによってアクセスします。



[コンポーネントインスペクタ] パネルのコレクションプロパティと、虫めがねアイコンをクリックすると表示される [値] ダイアログボックス。

Collection タグのシンタックスは次のとおりです。

```
[Collection (name="name", variable="varname",  
    collectionClass="mx.utils.CollectionImpl",  
    collectionItem="coll-item-classname", identifier="string")]  
public var varname:mx.utils.Collection;
```

次の表に、Collection タグの属性を示します。

属性	データ型	説明
name	String	(必須) [コンポーネントインスペクタ] パネルに表示される、コレクションの名前。
variable	String	(必須) 基礎となる Collection オブジェクトを指す ActionScript 変数。たとえば、Collection パラメータに Columns という名前を付けた場合でも、基礎となる variable 属性の名前は __columns などとすることができます。
collectionClass	String	(必須) コレクションプロパティ用にインスタンス化されるクラスのタイプ。通常は mx.utils.CollectionImpl ですが、mx.utils.CollectionImpl を継承するクラスを指定することもできます。
collectionItem	String	(必須) コレクション内に格納するコレクションアイテムのクラス。このクラス自体にも、メタデータによって公開される独自の Inspectable プロパティがあります。
identifier	String	(必須) コレクションアイテムクラスに含まれる Inspectable プロパティのうち、ユーザーが [値] ダイアログボックスを通じて新しいコレクションアイテムを追加したときに、Flash によってデフォルトの識別子として使用されるプロパティの名前。ユーザーが新しいコレクションアイテムを作成するたびに、identifier に一意のインデックスを付けたアイテム名が設定されます。たとえば、identifier=name と指定した場合、[値] ダイアログボックスには、name0、name1、name2 のように表示されます。

詳細については、[187 ページ](#)の「**コレクションプロパティ**」を参照してください。

IconFile タグについて

Flash オーサリング環境の [コンポーネント] パネルには、コンポーネントを表すアイコンを追加することができます。詳細については、[184 ページ](#)の「**アイコンの追加**」を参照してください。

ComponentTask タグについて

Flash JavaScript (JSFL) ファイルを指定して、Flash オーサリング環境内からコンポーネントのタスクを実行できます。それらのタスクは、`taskName` 属性に基づいて自動的にコンポーネントインスペクタウィザードメニュー ([自動選択ツール] をクリック) に表示されます。ComponentTask タグを使用して、コンポーネントと JSFL ファイルの関連付けを定義し、JSFL ファイルを必要とする他のファイルを関連付けます。JSFL ファイルは、Flash オーサリング環境で JavaScript API とやり取りします。

X
中

JSFL タスクファイル、および ComponentTask タグで宣言された必須の依存ファイルは、コンポーネントを SWC ファイルとして書き出す際に、コンポーネントの FLA ファイルと同じフォルダに格納する必要があります。

ComponentTask タグのシンタックスは次のとおりです。

```
[ComponentTask [taskName,taskFile [,otherFile[,...]]]
```

`taskName` 属性と `taskFile` 属性は必須です。`otherFile` 属性はオプションです。

次の例は、`SetUp.jsfl` と `AddNewSymbol.jsfl` を、`myComponent` という名前のコンポーネントクラスに関連付けています。`AddNewSymbol.jsfl` は `testXML.xml` ファイルを必要とし、`otherFile` 属性で指定されます。

```
[ComponentTask("Do Some Setup","SetUp.jsfl")]
[ComponentTask("Add a new Symbol","AddNewSymbol.jsfl","testXML.xml")]
class myComponent{
    //...
}
```

次の表に、ComponentTask タグの属性を示します。

属性	データ型	説明
taskName	String	(必須) 文字列としてのタスク名。この名前は、[コンポーネントインスペクタ] の [スキーマ] タブに表示される [タスク] ポップアップメニューで示されます。
taskFile	String	(必須) オーサリング環境内でタスクを実装する JSFL ファイルの名前。このファイルは、コンポーネントを SWC ファイルとして書き出す際に、コンポーネント FLA と同じフォルダに格納する必要があります。
otherFile	String	(オプション) XML ファイルなど、JSFL ファイルが必要とする 1 つ以上のファイル名。こうしたファイルは、コンポーネントを SWC ファイルとして書き出す際に、コンポーネント FLA と同じフォルダに格納する必要があります。

コンポーネントのパラメータの定義

コンポーネントを構築する際には、そのコンポーネントの外観および動作を定義するパラメータを追加できます。最もよく使用するパラメータは、オーサリングパラメータとして [コンポーネントインスペクタ] パネルおよびプロパティインスペクタに表示されます。また、**ActionScript** を使用してすべての **Inspectable** パラメータと **Collection** パラメータを設定することもできます。これらのプロパティは、コンポーネントクラスファイルで **Inspectable** タグを使用して定義します。詳細については、[150 ページの「Inspectable タグについて」](#)を参照してください。

次の例では、**JellyBean** クラスファイルにいくつかのコンポーネントパラメータを設定し、**Inspectable** タグを使用して、それらのコンポーネントパラメータを [コンポーネントインスペクタ] パネルに公開しています。

```
class JellyBean{
    // 文字列パラメータ
    [Inspectable(defaultValue="strawberry")]
    public var flavorStr:String;

    // 文字列リストパラメータ
    [Inspectable(enumeration="sour,sweet,juicy,rotten",defaultValue="sweet")]
    public var flavorType:String;

    // 配列パラメータ
    [Inspectable(name="Flavors", defaultValue="strawberry,grape,orange",
        verbose=1, category="Fruits")]
    var flavorList:Array;

    // オブジェクトパラメータ
    [Inspectable(defaultValue="belly:flop,jelly:drop")]
    public var jellyObject:Object;

    // color パラメータ
    [Inspectable(defaultValue="#ffffff")]
    public var jellyColor:Color;

    // setter
    [Inspectable(defaultValue="default text")]
    function set text(t:String) {
    }
}
```

パラメータには、次の型のうちいずれかを使用できます。

- Array
- Object
- List
- String
- 数値 (Number)

- Boolean
- Font Name
- Color

×
中

この JellyBean クラスの例は現実的なものではありません。実際の例を確認するには、Flash と共にインストールされる、< 言語 >/First Run/Classes/mx/controls ディレクトリの "Button.as" クラスファイルを参照してください。

コア関数について

コンポーネントのクラスファイルでは、`init()`、`createChildren()`、コンストラクタ関数、`draw()`、`size()` という 5 つの関数を定義する必要があります。UIComponent を継承するコンポーネントの場合、クラスファイル内の 5 つの関数は次の順序で呼び出されます。

■ `init()`

`init()` 関数の呼び出し中に初期化が実行されます。たとえば、この時点でインスタンスメンバの変数を設定することができ、コンポーネントの境界ボックスを非表示にできます。

`init()` が呼び出されると、`width` プロパティと `height` プロパティが自動的に設定されます。[161 ページの「init\(\) メソッドの定義」](#)を参照してください。

■ `createChildren()`

タイムラインでフレーム再生として呼び出されます。この間に、コンポーネントのユーザーはメソッドとプロパティを呼び出して、コンポーネントを設定できます。コンポーネントが作成する必要のある任意のサブオブジェクトが、`createChildren()` 関数内で作成されます。[161 ページの「createChildren\(\) メソッドの定義」](#)を参照してください。

■ コンストラクタ関数

コンポーネントのインスタンスを作成するために呼び出されます。コンポーネントコンストラクタ関数は、初期化によるコンフリクトを避けるため、一般に空のままになっています。[163 ページの「コンストラクタ関数について」](#)を参照してください。

■ `draw()`

プログラムで作成または編集するコンポーネントのビジュアルエレメントは、`draw` 関数内で実行する必要があります。[164 ページの「draw\(\) メソッドの定義」](#)を参照してください。

■ `size()`

この関数は、実行時にコンポーネントのサイズを変更するとき、およびコンポーネントの更新された `width` および `height` プロパティが渡されるときに必ず呼び出されます。コンポーネントのサブオブジェクトは、`size()` 関数内のコンポーネントに関する更新された `width` と `height` プロパティを基準にして、サイズ変更または移動できます。[164 ページの「size\(\) メソッドの定義」](#)を参照してください。

これらのコアコンポーネント関数は、この後のセクションで詳細に説明します。

init() メソッドの定義

Flash では、クラスの作成時に `init()` メソッドが呼び出されます。このメソッドは、コンポーネントをインスタンス化するとき一度だけ呼び出されます。

`init()` メソッドは、次の操作を行うために使用します。

- `super.init()` を呼び出します。
この操作は必須です。
- `boundingBox_mc` を非表示にします。

```
boundingBox_mc.width = 0;  
boundingBox_mc.height = 0;  
boundingBox_mc.visible = false;
```
- インスタンスのメンバー変数を作成します。

`width` パラメータ、`height` パラメータ、および `clip` パラメータは、このメソッドが呼び出されるまでは正しく設定されていません。

`init()` メソッドは `UIObject` のコンストラクタから呼び出されるため、制御のフローは `UIObject` のコンストラクタにたどり着くまで親のコンストラクタへと順にさかのぼります。`UIObject` のコンストラクタでは、最下位のサブクラスで定義されている `init()` メソッドを呼び出します。基本クラスの初期化が完了するよう、各クラスに実装する `init()` では `super.init()` を呼び出す必要があります。`init()` メソッドを実装しても `super.init()` を呼び出していないと、その `init()` メソッドからはどの基本クラスも呼び出されないことになり、基本クラスが使用可能な状態にならない可能性があります。

createChildren() メソッドの定義

コンポーネントで `createChildren()` メソッドを実装するのは、コンポーネント内で他のコンポーネントなどのサブオブジェクトを作成するためです。コンポーネントのサブオブジェクトをインスタンス化するには、`createChildren()` メソッド内でサブオブジェクトのコンストラクタを呼び出すのではなく、`createClassObject()` または `createObject()` を呼び出します。

`createChildren()` メソッド内では、`size()` を呼び出して、すべてのサブオブジェクトのサイズを最初から適切な値に設定することをお勧めします。また、`createChildren()` メソッド内で画面を更新するには `invalidate()` を呼び出してください。詳細については、[165 ページの「無効化について」](#)を参照してください。

`createClassObject()` メソッドは次のように記述します。

```
createClassObject(className, instanceName, depth, initObject)
```

次の表ではパラメータについて説明します。

パラメータ	データ型	説明
<i>className</i>	Object	クラス名
<i>instanceName</i>	String	インスタンス名
<i>depth</i>	Number	インスタンスの深度
<i>initObject</i>	Object	初期化プロパティを含んでいるオブジェクトです。

`createClassObject()` を呼び出すには、作成するサブオブジェクトの名前と型、および初期化パラメータを呼び出しの中で指定する必要があるため、サブオブジェクトに関する情報を知っておく必要があります。

次の例では、`createClassObject()` を呼び出して、コンポーネント内で使用する新しい **Button** オブジェクトを作成しています。

```
up_mc.createClassObject(mx.controls.Button, "submit_btn", 1);
```

`createClassObject()` の呼び出しでプロパティを設定するには、*initObject* パラメータの一部としてそのプロパティを追加します。次の例では、`label` プロパティの値を設定しています。

```
form.createClassObject(mx.controls.CheckBox, "cb", 0, {label:"Check this"});
```

次の例では、**TextInput** コンポーネントと **SimpleButton** コンポーネントを作成しています。

```
function createChildren():Void {
    if (text_mc == undefined)
        createClassObject(TextInput, "text_mc", 0, { preferredWidth: 80,
            editable:false });
        text_mc.addEventListener("change", this);
        text_mc.addEventListener("focusOut", this);

    if (mode_mc == undefined)
        createClassObject(SimpleButton, "mode_mc", 1, { falseUpSkin:
            modeUpSkinName, falseOverSkin: modeOverSkinName, falseDownSkin:
            modeDownSkinName });
        mode_mc.addEventListener("click", this);
        size()
        invalidate()
}
```

コンストラクタ関数について

コンストラクタ関数はコンポーネントクラスと同じ名前であるため、他の関数と区別できます。たとえば、次のコードは **ScrollBar** コンポーネントのコンストラクタ関数を示します。

```
function ScrollBar() {  
}
```

この場合、新しいスクロールバーをインスタンス化すると、この **ScrollBar()** コンストラクタが呼び出されます。

通常、コンポーネントのコンストラクタは空にしておいてください。コンストラクタでプロパティを設定すると、初期化呼び出しの順序によってはデフォルト値が上書きされてしまう場合があります。

UIComponent または **UIObject** を継承するコンポーネントの場合、**init()** メソッド、**createChildren()** メソッド、**size()** メソッドは自動的に呼び出されるため、コンストラクタ関数は空にしておくことができます。次のように記述します。

```
class MyComponent extends UIComponent{  
    ...  
    // これがコンストラクタ関数  
    function MyComponent(){  
    }  
}
```

すべてのバージョン 2 コンポーネントには、コンストラクタが呼び出された後に呼び出される **init()** 関数を定義する必要があります。初期化コードは、コンポーネントの **init()** 関数の中に記述します。詳細については、次のセクションを参照してください。

MovieClip を継承するコンポーネントの場合は、**init()** メソッド、**createChildren()** メソッド、およびコンポーネントを配置するメソッドを、コンストラクタ関数から呼び出すことをお勧めします。次に例を示します。

```
class MyComponent extends MovieClip{  
    ...  
    function MyComponent(){  
        init()  
    }  
  
    function init():Void{  
        createChildren();  
        layout();  
    }  
    ...  
}
```

コンストラクタの詳細については、『**ActionScript 2.0 の学習**』の「コンストラクタ関数の記述」を参照してください。

draw() メソッドの定義

draw() メソッド内にコードを記述すると、コンポーネントのビジュアルエレメントを作成または修正できます。つまり、draw() メソッドでは、コンポーネント自身をその状態を表す変数に合わせて描画します。draw() メソッドが前回呼び出された時点から現在までに複数のプロパティやメソッドが呼び出されている可能性があるため、それらすべてを draw() 本体の中で考慮する必要があります。

ただし、draw() メソッドを直接呼び出すことはお勧めしません。代わりに invalidate() メソッドを呼び出すことで、draw() に対する呼び出しをキューに格納して一度に処理させるようにしてください。この方法により、効率を向上し、コード管理を集中化できます。詳細については、[165 ページの「無効化について」](#)を参照してください。

draw() メソッド内では、Flash 描画 API を呼び出して境界線や罫線などのグラフィカルエレメントを描画できます。また、プロパティ値の設定やメソッドの呼び出しも可能です。clear() メソッドを呼び出して可視のオブジェクトを削除することもできます。

次のコードは Dial コンポーネントの例の一部です ([127 ページの「初めてのコンポーネントの作成」](#)を参照)。draw() メソッドで指針の回転を value プロパティに設定しています。

```
function draw():Void {  
    super.draw();  
    dial.needle._rotation = value;  
}
```

size() メソッドの定義

コンポーネントのサイズが componentInstance.setSize() メソッドによって実行時に変更されると、size() 関数が呼び出され、width プロパティと height プロパティが渡されます。size() メソッドをコンポーネントのクラスファイルで使用すると、コンポーネントのコンテンツを配置できます。

size() メソッドでは、少なくとも親クラスの size() メソッド (super.size()) を呼び出す必要があります。

次のコードは Dial コンポーネントの例の一部です ([127 ページの「初めてのコンポーネントの作成」](#)を参照)。size() メソッドで width パラメータと height パラメータを使用して、dial ムービークリップのサイズを変更しています。

```
function size():Void {  
    super.size();  
    dial._width = width;  
    dial._height = height;  
    invalidate();  
}
```

size() メソッド内では、draw() メソッドを直接呼び出すのではなく invalidate() メソッドを呼び出して、コンポーネントを再描画するよう指示します。詳細については、次のセクションを参照してください。

無効化について

コンポーネントでの表示の更新は、通常は直接に実行しないことをお勧めします。その代わりに、新しいプロパティ値のコピーを保存し、変更内容を示すフラグを設定してから、`invalidate()` メソッドを呼び出すようにしてください。変化したのはオブジェクトの外観のみで、サブオブジェクトのサイズと位置は変化していない場合に、このメソッドを呼び出します。このメソッドからは `draw()` メソッドが呼び出されます。

コンポーネントをインスタンス化する過程では、少なくとも 1 回は無効化メソッドを呼び出す必要があります。このメソッドを使用する最も一般的な場所は、`createChildren()` メソッドまたは `layoutChildren()` メソッド内です。

イベントの送出

コンポーネントで、親クラスから継承するイベント以外のイベントをブロードキャストする場合は、コンポーネントのクラスファイルで `dispatchEvent()` メソッドを呼び出す必要があります。

`dispatchEvent()` は `mx.events.EventDispatcher` クラスで定義されているメソッドで、`UIObject` を継承するすべてのコンポーネントが継承します。詳細については、『[ActionScript 2.0 コンポーネントリファレンスガイド](#)』の「[EventDispatcher クラス](#)」を参照してください。

また、新しく追加する個々のイベントごとに、クラスファイルの冒頭に `Event` メタデータタグを追加する必要があります。詳細については、[153 ページ](#)の「[Event タグについて](#)」を参照してください。



Flash アプリケーションにおけるコンポーネントイベントの扱いについては、[67 ページ](#)、[第 4 章](#)の「[コンポーネントイベントの処理](#)」を参照してください。

dispatchEvent() メソッドの使用

イベントをブロードキャストするには、コンポーネントの `ActionScript` クラスファイルの本文中に `dispatchEvent()` メソッドを使用します。`dispatchEvent()` メソッドは次のように記述します。

```
dispatchEvent(eventObj)
```

`eventObj` パラメータは、イベントの内容を表す `ActionScript` オブジェクトです。詳細については、このセクションの例を参照してください。

`dispatchEvent()` メソッドを呼び出す前に、次のようにこのメソッドを宣言する必要があります。

```
private var dispatchEvent:Function;
```

また、`dispatchEvent()` に渡すイベントオブジェクトも作成する必要があります。イベントオブジェクトには、リスナーがイベントの処理に使用するための、イベントに関する情報が含まれています。

イベントを送出する前に、次のようにしてイベントオブジェクトを明示的に構築できます。

```
var eventObj = new Object();
eventObj.type = "myEvent";
eventObj.target = this;
dispatchEvent(eventObj);
```

type プロパティと target プロパティの値を設定するショートカットシンタックスを使用すれば、次のように 1 行でイベントを送出することもできます。

```
ancestorSlide.dispatchEvent({type:"revealChild", target:this});
```

前の例では、target プロパティは暗黙的に設定されるので省略可能です。

Flash のマニュアルの中の各イベントの説明には、イベントプロパティ (オプションおよび必須) の一覧が記載されています。たとえば、ScrollBar.scroll イベントは、detail プロパティの他、type プロパティおよび target プロパティを取ります。詳細については、『ActionScript 2.0 コンポーネントリファレンスガイド』のイベントの説明を参照してください。

一般的なイベント

次の表に、さまざまなクラスによってブロードキャストされる一般的なイベントの一覧を示します。各コンポーネントは、自身にとって意味のあるイベントについてはブロードキャストを試みる必要があります。次のリストは、全コンポーネント用のイベントの完全なリストではありません。他のコンポーネントが再利用する可能性のあるイベントのみを列挙したものです。一部のイベントについてはパラメータを示していませんが、すべてのイベントには、暗黙のパラメータとして、イベントのブロードキャスト元オブジェクトへの参照が含まれます。

イベント	用途
click	Button コンポーネントで使用される他、マウスクリックがクリック以外の意味を持たないすべての場合に使用されます。
change	List、ComboBox、および他のテキスト入力コンポーネントにより使用されます。
scroll	スクロール (スクロールポップアップメニュー上にある " バンパー " のスクロール) 処理を実行する ScrollBar および他のコントロールにより使用されます。

さらに、すべてのコンポーネントは、基本クラスから継承した次のイベントをブロードキャストします。

UIComponent	説明
イベント	
load	コンポーネントが、サブオブジェクトを作成またはロードしています。
unload	コンポーネントが、サブオブジェクトをアンロードしています。
focusIn	この時点でコンポーネントが入力フォーカスを取得しました。HTML と同等のコンポーネントの中には、focus もブロードキャストするものがあります (ListBox、ComboBox、Button、Text)。また、すべてのコンポーネントは DOMFocusIn をブロードキャストします。
focusOut	コンポーネントが入力フォーカスを失いました。
move	コンポーネントが新しい場所に移動されました。
resize	コンポーネントのサイズが変更されました。

一般的なキーイベントを次の表に示します。

キーイベント	説明
keyDown	キーが押されました。code プロパティにはキーのコードが含まれます。また、ascii プロパティには、押されたキーの ASCII コードが含まれます。低レベルの Key オブジェクトに対する検査はしないでください。このイベントは必ずしも Key オブジェクトによって発生するとは限らないためです。
keyUp	キーが離されました。

スキンの割り当てについて

ユーザーインターフェイス (UI) コンポーネントは、アタッチされたムービークリップだけで構成されています。つまり、UI コンポーネント用のアセットはすべて UI コンポーネントムービークリップの外部にあってよいため、これらのアセットは他のコンポーネントにも使用できます。たとえば、チェックボックスの機能を必要とするようなコンポーネントを作成する場合は、既存の CheckBox コンポーネントアセットを再利用できます。

CheckBox コンポーネントでは、各種の状態 (FalseUp、FalseDown、Disabled、Selected など) を表示するのに別々のムービークリップを使用します。ただし、これらの状態は、" スキン " と呼ばれるカスタムクリップに関連付けることもできます。実行時には、新旧両方のムービークリップが SWF ファイルの中に書き出されます。古い状態は不可視になり、新しいムービークリップに取って代わられます。このようにスキンを変更することは " スキンの適用 " と呼ばれます。スキンの適用はオーサリング時にも実行時にも可能です。

コンポーネントにスキンを適用するには、コンポーネントで使用するすべてのスキンエレメント (ムービークリップシンボル) 用に変数を作成し、それにシンボルのリンケージ識別子を代入します。これにより、コンポーネント内のパラメータを変更するだけで、異なるスキンエレメントを設定できます。次のように記述します。

```
var falseUpIcon = "mySkin";
```

次の例では、**CheckBox** コンポーネントのさまざまな状態に対応するスキン変数を示しています。

```
var falseUpSkin:String = "";
var falseDownSkin:String = "";
var falseOverSkin:String = "";
var falseDisabledSkin:String = "";
var trueUpSkin:String = "";
var trueDownSkin:String = "";
var trueOverSkin:String = "";
var trueDisabledSkin:String = "";
var falseUpIcon:String = "CheckFalseUp";
var falseDownIcon:String = "CheckFalseDown";
var falseOverIcon:String = "CheckFalseOver";
var falseDisabledIcon:String = "CheckFalseDisabled";
var trueUpIcon:String = "CheckTrueUp";
var trueDownIcon:String = "CheckTrueDown";
var trueOverIcon:String = "CheckTrueOver";
var trueDisabledIcon:String = "CheckTrueDisabled";
```

スタイルについて

スタイルを使用して、コンポーネント内のすべてのグラフィックをクラスに登録して、登録したクラスに実行時のグラフィックの配色を制御させることができます。コンポーネントの実装には、スタイルをサポートするための特別なコードは何も必要ありません。基本クラス (UIObject および UIComponent) およびスキンにのみスタイルが実装されます。

新しいスタイルをコンポーネントに追加するには、コンポーネントクラスで `getStyle("styleName")` を呼び出します。インスタンス、カスタムスタイルシート、またはグローバルスタイルシートにスタイルが設定されている場合は、その値が抽出されます。設定されていない場合は、グローバルスタイルシートのスタイルにデフォルト値を設定する必要がある場合があります。

スタイルの詳細については、[82 ページの「スタイルによるコンポーネントのカラーとテキストの変更」](#)を参照してください。

スタイルへのスキンの登録

次の例では、Shape という名前のコンポーネントを作成しています。このコンポーネントには、円または四角形の 2 つのスキンのうち、いずれかのシェイプを表示します。この 2 つのスキンは、themeColor スタイルに登録されています。

スキンをスタイルに登録するには：

1. 新しい ActionScript ファイルを作成し、次のコードをそのファイルにコピーします。

```
import mx.core.UIComponent;

class Shape extends UIComponent{

    static var symbolName:String = "Shape";
    static var symbolOwner:Object = Shape;
    var className:String = "Shape";

    var themeShape:String = "circle_skin"

    function Shape(){
    }

    function init(Void):Void{
        super.init();
    }

    function createChildren():Void{
        setSkin(1, themeShape);
        super.createChildren();
    }
}
```

2. ファイルを "Shape.as" として保存します。
3. 新しい Flash ドキュメントを作成し、"Shape.as" と同じフォルダに、"Shape.fla" として保存します。
4. ステージ上に円を描き、それを選択してから F8 キーを押し、ムービークリップに変換します。円に、名前とリンケージ識別子 **circle_skin** を付与します。
5. circle_skin ムービークリップを開き、フレーム 1 に次の ActionScript を配置して、シンボルに themeColor というスタイル名を登録します。
`mx.skins.ColoredSkinElement.setColorStyle(this, "themeColor");`
6. コンポーネントの新しいムービークリップを作成します。
ムービークリップに名前を付け、リンケージ識別子 **Shape** を指定します。

7. 2つのレイヤーを作成します。stop() 関数を1つ目のレイヤーの1つ目のフレームに配置します。シンボル circle_skin を2つ目のフレームに配置します。

これは、コンポーネントムービークリップです。詳細については、[138 ページの「コンポーネントムービークリップの作成」](#)を参照してください。

8. StandardComponents.fla を外部ライブラリとして開き、UIComponent ムービークリップを、ステージ上の Shape ムービークリップの2つ目のフレーム (circle_skin を持つ) にドラッグします。

9. StandardComponents.fla を閉じます。

10. ライブラリで Shape ムービークリップを選択し、[ライブラリ] コンテキストメニューから [コンポーネント定義] を選択します (Windows の場合は右クリック、Mac の場合は Control キーを押しながらクリック)。次に、AS 2.0 クラス名として「Shape」を入力します。

11. ステージ上で Shape コンポーネントを持つムービークリップをテストします。

テーマカラーを変更するには、インスタンス上にスタイルを設定します。次のコードは、shape というインスタンス名を持つ Shape コンポーネントの色を、赤に変更します。

```
shape.setStyle("themeColor",0xff0000);
```

12. ステージ上に四角形を描き、ムービークリップに変換します。

リンケージ名「square_skin」を入力し、[最初のフレームに書き出し] チェックボックスがオンになっていることを確認します。



このムービークリップはコンポーネント上に配置されていないため、初期化の前にスキンを使用するためには [最初のフレームに書き出し] がオンに設定されている必要があります。

13. square_skin ムービークリップを開き、フレーム1に次の ActionScript を配置して、シンボルに themeColor というスタイル名を登録します。

```
mx.skins.ColoredSkinElement.setColorStyle(this, "themeColor");
```

14. メインタイムラインのステージ上の Shape コンポーネントのインスタンスに次のコードを配置します。

```
onClipEvent(initialize){  
    themeShape = "square_skin";  
}
```

15. ステージ上で Shape を持つムービークリップをテストします。結果として、赤色の四角形が表示されます。

新しいスタイル名の登録

新しいスタイル名を作成し、それがカラースタイルである場合、StyleManager.as ファイルの colorStyles オブジェクトにその新しい名前を追加します ("First Run\Classes\mx\styles\StyleManager.as")。次の例では shapeColor スタイルを追加しています。

```
// 継承されたカラースタイルの初期化
static var colorStyles:Object =
{
    barColor: true,
    trackColor: true,
    borderColor: true,
    buttonColor: true,
    color: true,
    dateHeaderColor: true,
    dateRollOverColor: true,
    disabledColor: true,
    fillColor: true,
    highlightColor: true,
    scrollTrackColor: true,
    selectedDateColor: true,
    shadowColor: true,
    strokeColor: true,
    symbolBackgroundColor: true,
    symbolBackgroundDisabledColor: true,
    symbolBackgroundPressedColor: true,
    symbolColor: true,
    symbolDisabledColor: true,
    themeColor:true,
    todayIndicatorColor: true,
    shadowCapColor:true,
    borderCapColor:true,
    focusColor:true,
    shapeColor:true
};
```

次のように、各ムービークリップのフレーム 1 の円および四角形のスキンに、新しいスタイル名を登録します。

```
mx.skins.ColoredSkinElement.setColorStyle(this, "shapeColor");
```

次のように、インスタンスにスタイルを設定することで、カラーを新しいスタイル名で変更できます。

```
shape.setStyle("shapeColor",0x00ff00);
```

複数の既存コンポーネントを組み込んだコンポーネントの作成

このセクションでは、Label、TextInput および Button コンポーネントを組み合わせる簡単な Login コンポーネントを作成します。このチュートリアルでは、既存コンポーネントのコンパイル済みでない Flash (FLA) ライブラリシンボルを新しいコンポーネントに追加して組み込む方法を説明します。完成済みのコンポーネントファイルのサンプルについては、Web サイトの Flash サンプルページ (www.adobe.com/go/learn_fl_tutorials_jp) を参照してください。次のサンプルファイルがあります。

- Login fla
- Login.as
- Login.swf

Login が提供する機能は、名前とパスワードを入力するためのインターフェイスです。Login の API には name および password の 2 つのプロパティがあり、name および password という TextInput フィールドのストリング値を設定および取得するために使用します。また、"Login" というラベルの付いたボタンがユーザーにクリックされると、Login コンポーネントでは "click" イベントを送出します。

- [172 ページの「Login の Flash \(FLA\) ファイルの作成」](#)
- [175 ページの「Login クラスファイル」](#)
- [178 ページの「Login コンポーネントのテストと書き出し」](#)

Login の Flash (FLA) ファイルの作成

まず、新しいコンポーネントシンボルを格納する Flash (FLA) ファイルを作成することから始めます。

Login の FLA ファイルを作成するには：

1. Flash で [ファイル]-[新規] を選択し、新しいドキュメントを作成します。
2. [ファイル]-[名前を付けて保存] を選択し、ファイル名を "Login fla" として保存します。
3. [挿入]-[新規シンボル] を選択します。Login という名前を付け、[ムービークリップ] タイプラジオボタンを選択します。

[新規シンボルの作成] ダイアログボックスの [リンケージ] セクションが開いていない場合は、[詳細] ボタンをクリックして開きます。
4. [ActionScript に書き出し] をオンに、[最初のフレームに書き出し] をオフにします。
5. リンケージ識別子を入力します。

デフォルトのリンケージ識別子は Login です。このデフォルト値を使用するものとして後の手順を示します。

6. [AS 2.0 クラス] テキストボックスに「Login」と入力します。この値はコンポーネントのクラス名です。

クラスをパッケージに含める場合は、パッケージ名全体を入力します。たとえば、mx.controls.CheckBox は、mx.controls パッケージの CheckBox クラスであることを示します。

7. [OK] をクリックします。

Flash がシンボル編集モードで開きます。

8. 新しいレイヤーを挿入します。上側のレイヤーに **Actions**、下側のレイヤーに **Assets** という名前を付けます。

9. Assets レイヤーのフレーム 2 を選択し、F6 キーを押してキーフレームを挿入します。

これでコンポーネントムービークリップの構造 (Actions レイヤーと Assets レイヤー) が作成されました。Actions レイヤーにはキーフレームが 1 つ、Assets レイヤーにはキーフレームが 2 つになります。

10. Actions レイヤーのフレーム 1 を選択し、F9 キーを押して [アクション] パネルを開きます。stop() グローバル関数を入力します。

これにより、ムービークリップがフレーム 2 に進むのを防ぎます。

11. [ファイル]-[読み込み]-[外部ライブラリを開く] を選択し、"Configuration/ComponentFLA" フォルダから "StandardComponents.flc" ファイルを選択します。

- Windows : ¥Program Files¥Adobe¥Adobe Flash CS3¥言語 ¥Configuration¥ComponentFLA¥StandardComponents.flc
- Macintosh : HD/ アプリケーション /Adobe Flash CS3/Configuration/ComponentFLA/StandardComponents.flc



フォルダの場所の詳細については、『Flash ユーザーガイド』の「Flash と共にインストールされる設定フォルダ」を参照してください。

12. Assets レイヤーのフレーム 2 を選択します。"StandardComponents.flc" のライブラリで、"Flash UI Components 2" フォルダを開きます。Assets レイヤーのフレーム 2 に、Button、Label、および TextInput コンポーネントシンボルをドラッグします。

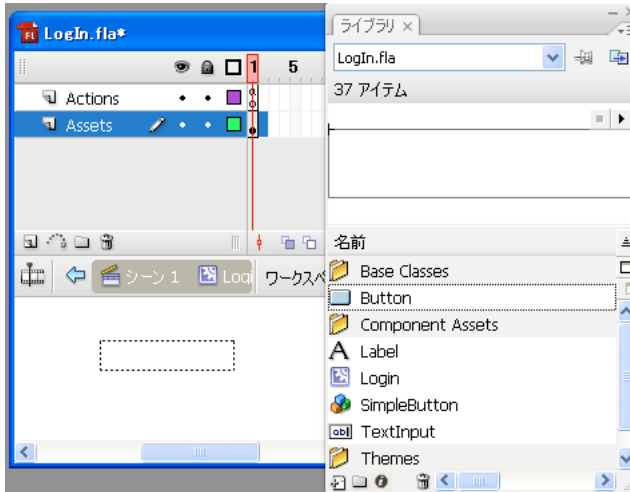
これらのコンポーネントが依存するアセットは、自動的に "Login.flc" のライブラリにコピーされます。

コンポーネントアセットは、すべて Assets レイヤーのフレーム 2 に追加されます。Actions レイヤーのフレーム 1 に stop() グローバル関数があるため、フレーム 2 のアセットは、ステージ上に配置しても表示されません。

アセットをフレーム 2 に追加するのは、次の 2 つの理由によります。

- アセットがすべて自動的にライブラリにコピーされ、動的にインスタンス化できる、またはメソッド、プロパティ、およびイベントにアクセスできるようにするため。

- フレームにアセットを配置することにより、ムービーがストリーミングされるときにアセットがよりスムーズにロードされ、ライブラリのアセットを1番目のフレームの前に書き出すよう設定する必要がなくなるため。この方法により、ダウンロードの遅延や長時間の停止を引き起こす可能性のある、初期のデータ転送の問題を回避できます。



Button コンポーネントラベルを、"StandardComponents.fla" のライブラリから "LogIn.fla" の Assets レイヤーにドラッグ

13. "StandardComponents.fla" のライブラリを閉じます。
14. Assets レイヤーでフレーム 1 を選択します。"Component Assets" フォルダ内の "LogIn.fla" ライブラリから BoundingBox ムービークリップをステージにドラッグします。
15. BoundingBox インスタンスに `boundingBox_mc` という名前を付けます。
16. [情報] パネルを使用して、この BoundingBox のサイズを LogInFinal ムービークリップのサイズ (340, 150) に変更し、0、0 の位置に移動します。

この BoundingBox インスタンスは、コンポーネントのライブプレビューを作成し、オーサリング中にコンポーネントのサイズを変更するために使用します。境界ボックスのサイズは、コンポーネント内のグラフィカルエレメントをすべて囲むように設定する必要があります。

✕
h

コンポーネント (バージョン 2 コンポーネントを含む) を継承する場合は、継承元コンポーネントのコードが既に使用しているインスタンス名を参照するので、継承元コンポーネントで使用しているインスタンス名を一切変更せずそのまま使用する必要があります。たとえば、インクルードするバージョン 2 コンポーネントで `boundingBox_mc` というインスタンス名が使用されている場合、この名前を変更しないでください。独自に作成するコンポーネントについては、同じスコープ内で既に使用されている名前とコンフリクトしない一意の名前であれば任意のインスタンス名を使用できます。

17. ライブラリで **LogIn** ムービークリップを選択し、[ライブラリ] コンテキストメニューから [コンポーネント定義] を選択します (Windows の場合は右クリック、Mac の場合は **Control** キーを押しながらクリック)。
18. [AS 2.0 クラス] テキストボックスに「**LogIn**」と入力します。

この値は **ActionScript** クラスの名前です。クラスがパッケージに含まれる場合、この値にはパッケージ名全体を指定する必要があります。たとえば、`mx.controls.CheckBox` は、`mx.controls` パッケージの `CheckBox` クラスであることを示します。
19. [OK] をクリックします。
20. ファイルを保存します。

LogIn クラスファイル

次に、**LogIn** コンポーネントの **ActionScript** クラスのコードを示します。コード内の各部分についてはコメントで説明しています。コンポーネントクラスファイルの各種エレメントの詳細については、[144 ページの「コンポーネントのクラスファイルの概要」](#)を参照してください。

このファイルを作成するには、Flash で新しい **ActionScript** ファイルを作成するか、任意のテキストエディタを使用します。このファイルには "LogIn.as" という名前を付けて、"LogIn.fla" ファイルと同じフォルダに保存します。

次に示す **LogIn** コンポーネントの **ActionScript** クラスコードを、新しく作成する "LogIn.as" ファイルにコピーまたは入力してください。コンポーネントコードの各エレメントに早く慣れるには、コピーせず手作業で入力することをお勧めします。

```
/* このクラスから直接参照できるよう
   パッケージを読み込む。*/
import mx.core.UIComponent;
import mx.controls.Label;
import mx.controls.TextInput;
import mx.controls.Button;

// Event メタデータタグ
[Event("change")]
[Event("click")]
class LogIn extends UIComponent
{
    /* これらのメンバー変数の宣言は必須。正当なコンポーネントとして
       コンポーネントフレームワーク内で認識されるために必要。*/
    static var symbolName:String = "LogIn";
    static var symbolOwner:Object = LogIn;
    var className:String = "LogIn";

    // このコンポーネントのグラフィカル表現。
    private var name_label:MovieClip;
    private var password_label:MovieClip;
```

```

private var name_ti:MovieClip;
private var password_ti:MovieClip;
private var login_btn:MovieClip;
private var boundingBox_mc:MovieClip;
private var startDepth:Number = 10;

/* プライベートメンバー変数には getter/setter で外部からアクセス可能。
   これらは InputText name および password のストリング値を表す。*/
private var __name:String;
private var __password:String;

/* コンストラクタ :
   どのクラスでも必須だが、v2 コンポーネントの場合は
   コンストラクタの内容が空で引数がないことが必要。
   すべての初期化処理は、必須の init メソッド内で、
   クラスインスタンスが作成された後に実行される。*/
function Login() {
}

/* 初期化コード。
   init メソッドは v2 コンポーネントに必須。また、ここでは
   super.init() として親クラスの init() を必ず呼び出すこと。
   init メソッドは、UIComponent を継承するコンポーネントで必須。*/
function init():Void {
    super.init();
    boundingBox_mc._visible = false;
    boundingBox_mc._width = 0;
    boundingBox_mc._height = 0;
}

/* 起動時に必要とされる子オブジェクトの作成 :
   createChildren メソッドは、
   UIComponent を継承するコンポーネントに必須。*/
public function createChildren():Void {
    name_label = createObject("Label", "name_label", this.startDepth++);
    name_label.text = "Name:";
    name_label._width = 200;
    name_label._x = 20;
    name_label._y = 10;

    name_ti = createObject("TextInput", "name_ti",
        this.startDepth++, {_width:200, _height:22, _x:20, _y:30});
    name_ti.html = false;
    name_ti.text = __name;
    name_ti.tabIndex = 1;

    /* このテキストインプットフィールドにフォーカスが来るように設定する。
    [ 制御 ]-[ キーボードショートカットを無効 ] が
    Flash デバッガで選択されていないならば、必ず選択する。そうしないと
    テスト時にフォーカスを設定できない場合がある。*/
    name_ti.setFocus();
}

```



```

        name_label = createObject("Label", "password_label",
this.startDepth++, {_width:200,_height:22,_x:20,_y:60});
        name_label.text = "Password:";

        password_ti = createObject("TextInput", "password_ti",
this.startDepth++, {_width:200,_height:22,_x:20,_y:80});
        password_ti.html = false;
        password_ti.text = __password;
        password_ti.password = true;
        password_ti.tabIndex = 2;

        login_btn = createObject("Button", "login_btn",
this.startDepth++, {_width:80,_height:22,_x:240,_y:80});
        login_btn.label = "LogIn";
        login_btn.tabIndex = 3;
        login_btn.addEventListener("click", this);

        size();

    }

    /* draw メソッドは v2 コンポーネントに必須。
        このメソッドは、コンポーネントが無効化された
        (invalidate() が呼び出された) 後に呼び出される。
        再描画を 1 回でまとめて処理できるので、各変更箇所を
        個別に再描画するよりも望ましい。この方法により、
        効率を向上し、コード管理を集中化できる。*/
    function draw():Void {
        super.draw();
    }

    /* size メソッドは、コンポーネントのサイズが変更されると
        呼び出される。ここで、子についてもサイズを変更する。
        size メソッドは、UIComponent を継承するコンポーネントで必須。*/
    function size():Void {
        super.size();
        // 必要に応じて再描画を実行させる。
        invalidate();
    }

    /* イベントハンドラ :
        LogIn ボタンがマウスクリックを受け取ると呼び出される。
        このイベントには、このコンポーネントのスコープ外からアクセスする
        必要があるため、クリックイベントは dispatchEvent を使用して送出する。*/
    public function click(evt){
        // メンバー変数を、入力フィールドの内容によって更新する。
        __name = name_ti.text;
        __password = password_ti.text;
        // ボタンによるクリックイベントが生じた場合は送出する。
        dispatchEvent({type:"click"});
    }

```

```

/* これは name プロパティの getter/setter。
   [Inspectable] メタデータにより、このプロパティは
   プロパティインスペクタに表示され、デフォルト値を
   設定可能。getter/setter を使用すると、invalidate を呼び出して、
   値が変更された場合に強制的にコンポーネントを再描画できる。*/
[Bindable]
[ChangeEvent("change")]
[Inspectable(defaultValue="")]
function set name(val:String){
    __name = val;
    invalidate();
}

function get name():String{
    return(__name);
}

[Bindable]
[ChangeEvent("change")]
[Inspectable(defaultValue="")]
function set password(val:String){
    __password=val;
    invalidate();
}

function get password():String{
    return(__password);
}
}

```

LogIn コンポーネントのテストと書き出し

ここまでで、グラフィカルエレメントを含む Flash ファイル、基本クラス、および LogIn コンポーネントの機能をすべて記述したクラスファイルを作成しました。次は、このコンポーネントをテストします。

コンポーネントのテストは、作成作業中、特にクラスファイルの記述作業中に実行できるのが理想的です。作成中に最も早くテストする方法は、コンポーネントをコンパイル済みクリップに変換し、そのクリップをコンポーネントの FLA ファイルで使用する的方法です。

コンポーネントの作成作業が完了したら、そのコンポーネントを SWC ファイルとして書き出します。詳細については、[180 ページの「コンポーネントの書き出しと配布」](#)を参照してください。

LogIn コンポーネントをテストするには：

1. "LogIn.fla" ファイルのライブラリで LogIn ムービークリップを選択し、[ライブラリ] コンテキストメニューの [コンパイルされたクリップへの変換] を選択します (Windows の場合は右クリック、Mac の場合は Control キーを押しながらクリック)。

コンパイルされたクリップが、LogIn SWF という名前でライブラリに追加されます。テストするためだけにムービークリップをコンパイルしています。それ以外の目的で使用する場合は、このセクションの後に示されている手順を実行して、LogIn ムービークリップを書き出します。

× ⚠	作成済みのコンパイルされたクリップが既にある場合 (2 度目、3 度目のテストを実行する場合など) は、[ライブラリのコンフリクトを解決] ダイアログボックスが表示されます。[既存のアイテムを置き換える] を選択して、新しいバージョンをドキュメントに追加します。
--------	---

2. LogIn SWF をメインタイムラインのフレーム 1 のステージにドラッグします (ムービークリップ タイムラインではなく、メインタイムラインのシーン 1 にドラッグしていることを確認します)。

name プロパティと password プロパティの設定は、[パラメータ] タブまたは [コンポーネントインスペクタ] パネルで行います。これは、ユーザーが何か入力する前に、" 名前をここに入力 " などのデフォルトのテキストを表示させる場合に便利です。name プロパティや password プロパティを設定すると、name および password の InputText サブコンポーネントでデフォルトのテキストが実行時に変更されます。

実行時に value プロパティをテストするには、ステージ myLogin の LogIn インスタンスに名前を付けて、メインタイムラインのフレーム 1 に次のコードを追加します。

```
// ログインの値を表示するためのテキストフィールドを作成する。
createTextField("myLoginValues",10,10,10,340,40)
myLoginValues.border = true;
// ログインコンポーネントインスタンスで送出されるクリックイベントのイベントハンドラ。
function click(evt){
/* ここで認証が実行されます。
   たとえば名前とパスワードは、名前とパスワードの認証を実行して、ユーザーにセッション
   ID および許可ロールを返す Web サービスに渡される。*/
  myLoginValues.text = "Processing...\r";
  myLoginValues.text += "Name: " + myLogin.name + " Password: " +
  myLogin.password;
}

myLogin.addEventListener("click",this);
```

3. [制御]-[ムービープレビュー] を選択して、Flash Player でコンポーネントをテストします。

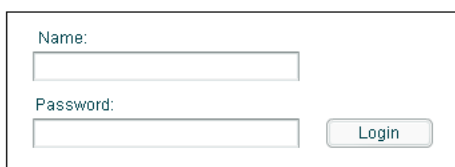
× ⚠	元のドキュメント内でこのコンポーネントのテストを行ったため、2 つのシンボルに対して同じリンケージ識別子が存在することを示す警告メッセージが表示される場合があります。このコンポーネントは引き続き動作します。実際には、他のドキュメント内の新しいコンポーネントを使用します。この場合、リンケージ識別子は一意となります。
--------	---

LogIn コンポーネントを書き出すには：

1. "LogIn.fla" ファイルのライブラリで LogIn ムービークリップを選択し、[ライブラリ] コンテキストメニューの [コンポーネント定義] を選択します (Windows の場合は右クリック、Mac の場合は Control キーを押しながらクリック)。
2. [コンポーネント] パネルの [表示] を確認します。
3. [OK] をクリックします。
4. "LogIn.fla" ファイルのライブラリで LogIn ムービークリップを再び選択し、[ライブラリ] コンテキストメニューの [SWC ファイル書き出し] を選択します (Windows の場合は右クリック、Mac の場合は Control キーを押しながらクリック)。
5. SWC ファイルを保存する場所を選択します。

ユーザーレベル設定フォルダ内の "Components" フォルダに保存した場合は、[コンポーネント] パネルをリロードすればコンポーネントが表示され、Flash の再起動は必要ありません。

× ❗	フォルダの場所の詳細については、『Flash ユーザーガイド』の「Flash と共にインストールされる設定フォルダ」を参照してください。
--------	--



完成した LogIn コンポーネント

コンポーネントの書き出しと配布

Flash では、コンポーネントはコンポーネントパッケージ (SWC ファイル) として書き出されます。コンポーネントは SWC ファイルまたは FLA ファイルの形式で配布できます。FLA 形式によるコンポーネントの配布については、Adobe DevNet にある

www.adobe.com/jp/support/flash/applications/creating_comps/creating_comps12.html の記事を参照してください。

コンポーネントを配布するには、SWC ファイルとして書き出すのが最善の方法です。SWC ファイルには、すべての ActionScript、SWF ファイル、およびコンポーネントを使用するために必要なその他のオプションファイルが含まれているからです。また、SWC ファイルは、コンポーネントを作成しながら、そのコンポーネントを使用するアプリケーションの作成を並行して進める場合にも便利です。

SWC ファイルは、Flash、Macromedia Dreamweaver MX 2004、および Macromedia Director MX 2004 で使用するコンポーネントの配布に使用できます。

コンポーネントを他の開発者のために開発する場合でも、または自分だけが使用するために開発する場合でも、コンポーネント開発の過程で SWC ファイルをテストすることは重要です。たとえば、コンポーネントの SWC ファイルで、FLA ファイルには見られない問題が発生する可能性があります。このセクションでは、SWC ファイルについて説明した上で、Flash での SWC の読み込みおよび書き出しについても説明します。

SWC ファイルについて

SWC ファイルは、Flash オーサリングツールで生成される、zip 形式に似たファイルです (PKZIP アーカイブ形式でパッケージ化および展開されます)。

次の表に、SWC ファイルのコンテンツの説明を示します。

ファイル	説明
catalog.xml	(必須) コンポーネントパッケージおよびその個々のコンポーネントのコンテンツをリストして、SWC ファイル内の他のファイルへのディレクトリとして機能します。
ActionScript (AS) ファイル	Flash でコンポーネントを作成すると、ソースコードは、そのコンポーネントのクラス宣言を含む ActionScript ファイルとして保存されます。 コンパイラでは、コンポーネントが継承されるときに、このソースコードを使用して型チェックを行います。実装用 SWF ファイルにコンパイル済みバイトコードが既に含まれているので、オーサリングツールでは AS ファイルのコンパイルを行いません。 ソースコードによっては、関数本体をいっさい含まず、型チェック専用に提供されるクラス定義が含まれているものもあります。
SWF ファイル	(必須) コンポーネントを実装する SWF ファイル。1つの SWF ファイル内に複数のコンポーネントを定義することもできます。Flash 8 以降でコンポーネントを作成した場合、SWF ファイルごとに1つのコンポーネントだけが書き出されます。
ライブプレビュー用 SWF ファイル	(オプション) 指定した場合、これらの SWF ファイルはオーサリングツールのライブプレビューに使用されます。省略した場合、代わりにコンポーネントを実装する SWF ファイルがライブプレビューに使用されます。ほとんどの場合、ライブプレビュー SWF ファイルは省略可能です。ライブプレビュー SWF ファイルを含める必要があるのは、動的なデータ (たとえば、Web サービスの呼び出し結果を示すテキストフィールド) によってコンポーネントの外観が左右される場合のみです。
SWD ファイル	(オプション) SWD ファイルは実装用 SWF ファイルに対応しており、SWF ファイルのデバッグに使用されます。SWD ファイルの名前は、常に SWF ファイルの名前と同じですが、拡張子 .swd が付けられます。

ファイル	説明
PNG ファイル	(オプション) 18x18、8 ビット / ピクセルのアイコンが含まれている PNG ファイル。オーサリングツールのユーザーインターフェイス内にコンポーネントアイコンを表示する目的に使用します。アイコンを何も指定しないと、デフォルトのアイコンが表示されます。詳細については、 184 ページの「アイコンの追加」 を参照してください。
プロパティインスペクタ SWF ファイル	(オプション) オーサリングツールでカスタムのプロパティインスペクタとして使用する SWF ファイル。このファイルを省略すると、デフォルトのプロパティインスペクタがユーザーに表示されます。

Flash 環境で SWC ファイルを生成した後、オプションで SWC ファイル内に他のファイルをインクルードすることもできます。たとえば、コンポーネントのソースコードへのアクセスをユーザーに許可する場合は、ReadMe ファイルまたは FLA ファイルをインクルードすることもできます。ファイルを追加するには、Adobe Extension Manager (www.adobe.com/jp/exchange/em_download/を参照) を使用します。

SWC ファイルは 1 つのディレクトリに展開されるので、コンフリクトを防ぐために、各コンポーネントには一意のファイル名を付ける必要があります。

SWC ファイルの書き出し

Flash では、ムービークリップを SWC ファイルとして書き出すことにより、SWC ファイルの書き出しができます。SWC ファイルを書き出す際には、Flash アプリケーションのテスト中と同様に、コンパイル時のエラーがレポートされます。

SWC ファイルの書き出しは、次の 2 つの目的で実行します。

- 完成したコンポーネントを配布するため
- 開発中にテストするため

完成済みコンポーネントの SWC ファイルの書き出し

コンポーネントを使用するために必要な ActionScript、SWF ファイル、その他のオプションファイルをすべて含む SWC ファイルとして、コンポーネントを書き出すことができます。

完成済みコンポーネントの SWC ファイルを書き出すには：

1. Flash のライブラリで、コンポーネントムービークリップを選択します。
2. 右クリック (Windows の場合)、または Control キーを押しながらクリック (Mac の場合) して、[ライブラリ] コンテキストメニューを開きます。
3. [ライブラリ] コンテキストメニューから [SWC ファイル書き出し] を選択します。
4. SWC ファイルを保存します。

開発中における SWC ファイルのテスト

開発中の各段階で、コンポーネントを SWC ファイルとして書き出し、アプリケーション内でテストすることをお勧めします。ユーザーレベル設定フォルダ内の "Components" フォルダに SWC ファイルを書き出した場合は、[コンポーネント] パネルをリロードすればよく、Flash を終了して再起動する必要はありません。

開発中に SWC ファイルをテストするには：

1. Flash のライブラリで、コンポーネントムービークリップを選択します。
2. 右クリック (Windows の場合)、または Control キーを押しながらクリック (Mac の場合) して、[ライブラリ] コンテキストメニューを開きます。
3. [ライブラリ] コンテキストメニューから [SWC ファイル書き出し] を選択します。
4. ユーザーレベル設定フォルダ内の "Components" フォルダに移動します。

Configuration/Components



フォルダの場所の詳細については、『Flash ユーザーガイド』の「Flash と共にインストールされる設定フォルダ」を参照してください。

5. SWC ファイルを保存します。
6. [コンポーネント] パネルのオプションメニューから [リロード] を選択します。
コンポーネントが [コンポーネント] パネルに表示されます。
7. コンポーネントを [コンポーネント] パネルからドキュメントにドラッグします。

Flash へのコンポーネント SWC ファイルの読み込み

コンポーネントを他の開発者に配布するときは、配布を受けた開発者がすぐにインストールして使用できるように、次の指示を含めてください。

SWC ファイルを読み込むには：

1. SWC ファイルを "Configuration/Components" ディレクトリにコピーします。
2. Flash を再起動します。

コンポーネントのアイコンが [コンポーネント] パネルに表示されます。

コンポーネント開発の最後の手順

コンポーネントを作成してパッケージ化の準備が整ったら、アイコンとツールヒントを追加できます。必要な手順がすべて完了したことを確認するには、[185 ページの「コンポーネント開発チェックリスト」](#)も参照してください。

アイコンの追加

Flash オーサリング環境の [コンポーネント] パネルには、コンポーネントを表すアイコンを追加することができます。

コンポーネント用のアイコンを追加するには：

1. 新規にイメージを作成します。
イメージは、18 x 18 ピクセルの寸法にして PNG 形式で保存する必要があります。アルファ透明度については必ず 8 ビットとし、左上のピクセルは、マスキングがサポートされるよう透明にする必要があります。
2. コンポーネントの ActionScript クラスファイルで、クラス定義の前に次の定義を追加します。
`[IconFile("component_name.png")]`
3. FLA ファイルと同じディレクトリにイメージを追加します。Flash で SWC ファイルを書き出すと、アーカイブのルートレベルでイメージがインクルードされます。

ツールチップの追加

Flash オーサリング環境の [コンポーネント] パネルで、コンポーネント名またはアイコン上にマウスを移動させると、ツールヒントが表示されます。

ツールヒントは [コンポーネント定義] ダイアログボックスで定義します。コンポーネントの FLA ファイルの [ライブラリ] オプションメニューからこのダイアログボックスにアクセスできます (Windows の場合は右クリック、Mac の場合は Control キーを押しながらクリック)。

[コンポーネント定義] ダイアログボックスでツールヒントを追加するには：

1. 作成したコンポーネントの FLA ファイルが Flash で開かれ、ライブラリが表示されている状態にします ([ウィンドウ]-[ライブラリ])。
2. [ライブラリ] オプションメニューをクリックします (Windows の場合は右クリック、Mac の場合は Control キーを押しながらクリック)。
[ライブラリ] オプションメニューは、[ライブラリ] タイトルバーの右側にある、3 本の線と下向き矢印のアイコンです。
3. [コンポーネント定義] オプションを選択します。
4. [コンポーネント定義] ダイアログボックスの [オプション] で、[コンポーネント] パネルの [表示] を選択します。
[ツールヒントテキスト] ボックスが編集可能になります。

5. 作成したコンポーネントに対するツールヒントのテキストを [ツールヒントテキスト] ボックスに入力します。
6. [OK] をクリックして変更を保存します。

コンポーネント開発チェックリスト

コンポーネントを設計するときは、次の点を考慮してください。

- ファイルのサイズは可能な限り小さく保ちます。
- 機能に汎用性を持たせることによって、コンポーネントの再利用性が最大限に高まるようにします。
- オブジェクトの周囲の境界線を描画するには、グラフィカルエレメントではなく `RectBorder` クラス (`mx.skins.halo.RectBorder`) を使用します。詳細については、『*ActionScript 2.0 コンポーネントリファレンスガイド*』の「*RectBorder クラス*」を参照してください。
- タグベースのスキンを使用します。
- 変数 `symbolName`、`symbolOwner`、`className` を定義します。
- 初期設定値は受け継がれます。現在のバージョンではオブジェクトがスタイルプロパティを持つようになったため、スタイルとプロパティに初期設定値を指定できます。このため、ユーザーがデフォルト状態をオーバーライドしない限り、オブジェクトを構築する際に初期化コードで初期設定値を設定する必要はありません。
- シンボルを定義する際には、必要でない限り [最初のフレームに書き出し] を選択することは避けます。Flash では、コンポーネントを Flash アプリケーションで使う直前にロードするため、このオプションを選択した場合、コンポーネントが親の先頭フレーム内にプリロードされてしまいます。コンポーネントを先頭フレームで通常プリロードしないのは、Web という条件を考慮するからです。プリローダーが起動するより前にコンポーネントがロードされることになるので、プリローダーの意味がなくなります。これは望ましいことではありません。
- ムービークリップで複数のフレームを作成しないようにします (Assets レイヤーの 2 つのフレームは除く)。
- 必ず `init()` メソッドと `size()` メソッドを実装し、それぞれの中で `Super.init()` と `Super.size()` を呼び出します。また、これらのメソッド内で必要以上の処理は実行しないようにします。
- `_root.myVariable` などの絶対参照を避けます。
- `attachMovie()` ではなく `createClassObject()` を使用します。
- `draw()` メソッドを呼び出す場合は、このメソッドを明示的に呼び出すのではなく、`invalidate()` メソッドと `invalidateStyle()` メソッドを使用します。
- 作成するコンポーネントに Flash コンポーネントを組み込む場合は、"Configuration/ComponentFLA" フォルダから、"StandardComponents.fla" ファイルのライブラリにあるコンパイル済みでないムービーシンボルを使用します。

コレクションプロパティ

Flash で新規カスタムコンポーネントを作成する場合、プロパティ値をユーザーが編集できるようにすることが可能です。こうしたプロパティは、コレクションプロパティと呼ばれます。ユーザーはプロパティ値を [値] ダイアログボックスで編集することができます。このダイアログボックスは、コンポーネントの [パラメータ] タブ内のテキストボックスから開きます。

通常、コンポーネントには、コンポーネントユーザーが求める一定の要件を満たすだけの柔軟性が備わっていますが、特定のタスク向けの機能も含まれています。コンポーネントの柔軟性のためには、コンポーネント内のプロパティにも柔軟性が求められます (つまり、コンポーネントによっては、プロパティの値だけでなくコンポーネントのユーザーによってもプロパティの変更が可能です)。

コレクションプロパティを使用することにより、1つのオブジェクトモデルの中で一定でない個数の編集可能プロパティを作成することが可能になります。Flash では、Collection クラスを使用することにより、[コンポーネントインスペクタ] パネルでこうしたプロパティを管理することができます。具体的には、Collection クラスは、関連性のあるオブジェクト (コレクションアイテムと呼ばれる) で構成されるグループを管理するためのヘルパークラスです。コンポーネントのプロパティをコレクションアイテムとして定義し、[コンポーネントインスペクタ] パネルを通じて操作できるとすると、ユーザーはオーサリング時に [値] ダイアログボックスでコレクションアイテムを追加、削除、および変更できます。



コレクションおよびコレクションアイテムを定義する方法は次のとおりです。

- コンポーネントの ActionScript ファイルで Collection メタデータタグを使用してコレクションプロパティを定義します。詳細については、[156 ページの「Collection タグについて」](#)を参照してください。
- 異なる Inspectable プロパティを持つ別の ActionScript ファイルで、コレクションアイテムをクラスとして定義します。

Flash でコレクションを使用すると、関連性のあるアイテムのグループをプログラム上で管理することが容易になります。旧バージョンの Flash では、関連性のあるアイテムのグループを管理するにはコンポーネントの作者がプログラムで複数の配列を同期させる必要がありました。

[値] ダイアログボックスだけでなく、プログラムから Collection のインスタンスと値を管理するために Collection インターフェイスおよび Iterator インターフェイスも用意されています。『ActionScript 2.0 コンポーネントリファレンスガイド』の「Collection インターフェイス」および「Iterator インターフェイス」を参照してください。

この章では、次のセクションについて説明します。

コレクションプロパティの定義.....	188
簡単なコレクションの例	189
コレクションアイテムのクラス定義.....	191
プログラムによるコレクション情報へのアクセス	192
コレクションが定義されているコンポーネントの SWC ファイルへの書き出し	194
コレクションプロパティが定義されているコンポーネントの使用	195

コレクションプロパティの定義

コレクションプロパティを定義するには、コンポーネントの ActionScript ファイルで Collection タグを使用します。詳細については、[156 ページの「Collection タグについて」](#)を参照してください。



この項では、コンポーネントの作成方法、およびコンポーネントの Inspectable プロパティの作成方法を理解していることを前提に説明します。

コレクションプロパティを定義するには：

1. コンポーネントの FLA ファイルを作成します。[138 ページの「コンポーネントムービークリップの作成」](#)を参照してください。
2. ActionScript クラスを作成します。[143 ページの「ActionScript クラスファイルの作成」](#)を参照してください。
3. 作成した ActionScript クラスに Collection メタデータタグを挿入します。詳細については、[156 ページの「Collection タグについて」](#)を参照してください。

4. コンポーネントの `ActionScript` ファイルに、コレクション用の `get` および `set` メソッドを定義します。

5. [ウィンドウ]-[サンプルライブラリ]-[クラス] を選択してコンポーネントのライブラリに `UtilsClasses` をドラッグし、ユーティリティクラスを `FLA` ファイルに追加します。

`UtilsClasses` には、`Collection` インターフェイス用の `mx.utils.*` パッケージが含まれています。



`UtilsClasses` は、`ActionScript` クラスではなく `FLA` ファイルに関連付けられるため、コンポーネントの `ActionScript` クラスを表示してシンタックスをチェックすると、コンパイルエラーがスローされます。

6. コレクションアイテムプロパティを含むクラスのコードを記述します。

[191 ページの「コレクションアイテムのクラス定義」](#)を参照してください。

簡単なコレクションの例

次に示すのは、`"MyShelf.as"` という簡単なコンポーネントクラスファイルの例です。`UIObject` クラスを継承するコンポーネントについて、読み込み、メソッド、宣言の最小限のセット、およびコレクションプロパティが含まれています。

このサンプルのようにして `mx.utils.*` を読み込むと、`mx.utils` からのクラス名を完全修飾名で指定する必要がなくなります。たとえば、`mx.utils.Collection` は `Collection` と表現できます。

```
import mx.utils.*;
// 標準的なクラス宣言
class MyShelf extends mx.core.UIObject
{
// 全クラスに必須の変数
    static var symbolName:String = "MyShelf";
    static var symbolOwner:Object = Object(MyShelf);
    var className:String = "MyShelf";

// Collection メタデータタグと属性
    [Collection(variable="myCompactDiscs",name="My Compact Discs",collectionClass="mx.utils.CollectionImpl",collectionItem="CompactDisc", identifier="Title")]

// コレクションの get メソッドと set メソッド
    public function get MyCompactDiscs():mx.utils.Collection
    {
        return myCompactDiscs;
    }
    public function set MyCompactDiscs(myCDs:mx.utils.Collection):Void
    {
        myCompactDiscs = myCDs;
    }
}
```

```

// プライベートクラスメンバー
private var myCompactDiscs:mx.utils.Collection;

// コレクションアイテムクラスへの参照はコーディングが必須
// これは、コンパイラでこのクラスを依存クラスとして強制的に
// SWC に追加する必要があるため
private var collItem:CompactDisc;

// mx.utils.CollectionImpl クラスへの参照はコーディングが必須
// これは、コンパイラでこのクラスを依存クラスとして強制的に
// SWC に追加する必要があるため
private var coll:mx.utils.CollectionImpl;

// 全クラスに必須のメソッド
function init(Void):Void {
    super.init();
}
function size(Void):Void {
    super.size();
}
}

```

このクラスに付随するテスト用 FLA ファイルを作成するには：

1. Flash で [ファイル]-[新規] を選択し、Flash ドキュメントを作成します。
 2. [挿入]-[新規シンボル] を選択します。名前、リンケージ識別子、および **MyShelf** という AS 2.0 クラス名を入力します。
 3. [最初のフレームに書き出し] をオフにして [OK] をクリックします。
 4. **MyShelf** シンボルをライブラリから選択し、[ライブラリ] パネルのオプションメニューから [コンポーネント定義] を選択します。ActionScript 2.0 クラス名として「**MyShelf**」と入力します。
 5. [ウィンドウ]-[サンプルライブラリ]-[クラス] を選択し、UtilClasses を "MyShelf.fla" のライブラリにドラッグします。
 6. **MyShelf** シンボルの [タイムライン] で、1つのレイヤーに **Assets** という名前をつけます。レイヤーをもう1つ作成し、**Actions** という名前をつけます。
 7. Actions レイヤーのフレーム1に stop() 関数を配置します。
 8. Assets レイヤーのフレーム2を選択し、[挿入]-[タイムライン]-[キーフレーム] を選択します。
 9. "Configuration/ComponentFLA" フォルダから "StandardComponents.fla" を開き、UIObject のインスタンスを Assets レイヤーのフレーム2の **MyShelf** ステージにドラッグします。
- 前のクラスファイルを見るとわかるように、**MyShelf** は UIObject の拡張クラスであるため、コンポーネントの FLA ファイルに UIObject を含める必要があります。

10. Assets レイヤーのフレーム 1 で、棚を描画します。

学習の目的で使用する MyShelf コンポーネントの視覚的表現に過ぎないので、簡単な長方形でかまいません。

11. ライブラリで MyShelf ムービークリップを右クリックし、[コンパイルされたクリップに変換] を選択します。

これで、MyShelf SWF ファイル (ライブラリに追加されるコンパイルされたクリップ) を "MyShelf.fla" ファイルにドラッグし、コンポーネントをテストできるようになります。コンポーネントを再コンパイルするたびに、[ライブラリのコンフリクトを解決] ダイアログボックスが表示されます。これは、コンポーネントの前のバージョンが既にライブラリに存在するためです。[既存のアイテムを置き換える] を選択してアイテムを置き換えます。

✕
❗
ここで、既に CompactDisc クラスが作成されている必要があります。このクラスがないと、コンパイルされたクリップへの変換時にコンパイルエラーが発生します。

コレクションアイテムのクラス定義

コレクションアイテムのプロパティのコードは、別の ActionScript クラスに記述します。その方法は次のとおりです。

- UIObject または UIComponent のいずれも拡張しないクラスを定義します。
- すべてのプロパティを Inspectable タグを使用して定義します。
- すべてのプロパティを変数として定義します。get (getter) メソッドと set (setter) メソッドは使用しません。

次に、コレクションアイテムクラスファイルの簡単な例を示します。ファイル名は "CompactDisc.as" です。

```
class CompactDisc{
    [Inspectable(type="String", defaultValue="Title")]
    var title:String;
    [Inspectable(type="String", defaultValue="Artist")]
    var artist:String;
}
```

"CompactDisc.as" クラスファイルを表示するには、[189 ページの「簡単なコレクションの例」](#)を参照してください。

プログラムによるコレクション情報へのアクセス

Flash では、Collection インターフェイスと Iterator インターフェイスを通じて、プログラムによってコレクションデータにアクセスできます。Collection インターフェイスを使用すると、コレクション内のアイテムを追加、変更、および削除できます。Iterator インターフェイスを使用すると、ループ処理によってコレクション内のアイテムを確認できます。

Collection インターフェイスと Iterator インターフェイスは、次の 2 つの場合に使用します。

- 192 ページの「コンポーネントクラス (AS) ファイル内のコレクション情報へのアクセス」
- 193 ページの「コレクションアイテムへの Flash アプリケーションによる実行時アクセス」

また、上級開発者は、プログラムによってコレクションの作成、内容の設定、アクセス、および削除も実行できます。詳細については、『ActionScript 2.0 コンポーネントリファレンスガイド』の「Collection インターフェイス」を参照してください。

コンポーネントクラス (AS) ファイル内のコレクション情報へのアクセス

コンポーネントのクラスファイルには、オーサリング時または実行時に定義したコレクションアイテムを操作するためのコードを記述できます。

コンポーネントクラスファイル内のコレクションアイテム情報にアクセスするには、次のいずれかの方法を使用します。

- Collection タグには `variable` という属性が含まれているので、ここに `mx.utils.Collection` タイプの変数を指定し、この変数を使用してコレクションにアクセスします。次に例を示します。

```
[Collection(name="LinkButtons", variable="__linkButtons",  
    collectionClass="mx.utils.CollectionImpl", collectionItem="ButtonC",  
    identifier="ButtonLabel")]  
public var __linkButtons:mx.utils.Collection;
```

- `Collection.iterator()` メソッドを呼び出して、コレクションアイテムの `Iterator` インターフェイスにアクセスします。次に例を示します。

```
var itr:mx.utils.Iterator = __linkButtons.iterator();
```

- `Iterator` インターフェイスを使用して、コレクション内のアイテムを順に確認します。
`Iterator.next()` メソッドは `Object` を返すので、コレクションアイテムのタイプを定義する必要があります。次に例を示します。

```
while (itr.hasNext()) {  
    var button:ButtonC = ButtonC(itr.next());  
    ...  
}
```


- アプリケーションの必要に応じて、コレクションアイテムのプロパティにアクセスします。次に例を示します。

```
item.label = button.ButtonLabel;

if (button.ButtonLink != undefined) {
    item.data = button.ButtonLink;
}
else {
    item.enabled = false;
}
```

コレクションアイテムへの Flash アプリケーションによる実行時アクセス

コレクションプロパティが定義されているコンポーネントを Flash アプリケーションで使用する場合は、実行時にコレクションプロパティにアクセスできます。この例では、[値] ダイアログボックスを使用してコレクションプロパティにいくつかのアイテムを追加し、実行時には Collection API および Iterator API を使用してそれらのアイテムを表示します。

実行時にコレクションアイテムにアクセスするには：

1. 前の手順で作成した "MyShelf.fla" ファイルを開きます。

189 ページの「[簡単なコレクションの例](#)」を参照してください。

この例では、MyShelf コンポーネントと CompactDisc コレクションを使用します。

2. [ライブラリ] パネルを開いてコンポーネントをステージ上にドラッグし、インスタンス名を指定します。
この例では、myShelf というインスタンス名を使用します。
3. コンポーネントを選択し、[コンポーネントインスペクタ] パネルを開いて、[パラメータ] タブを表示します。コレクションプロパティを含む行をクリックし、行の右側にある虫めがねのアイコンをクリックします。[値] ダイアログボックスが表示されます。
4. [値] ダイアログボックスを使用して、コレクションプロパティに値を入力します。
5. ステージ上のコンポーネントを選択した状態で [アクション] パネルを開き、次のコードを入力して、コンポーネントにアタッチします。

```
onClipEvent (mouseDown) {
    import mx.utils.Collection;
    import mx.utils.Iterator;
    var myColl:mx.utils.Collection;
    myColl = _parent.myShelf.MyCompactDiscs;

    var itr:mx.utils.Iterator = myColl.getIterator();
    while (itr.hasNext()) {
        var cd:CompactDisc = CompactDisc(itr.next());
```

```

        var title:String = cd.Title;
        var artist:String = cd.Artist;
        trace("Title: " + title + " Artist: " + artist);
    }
}

```

コレクションにアクセスするには、シンタックス `componentName.collectionVariable` を使用します。イテレータにアクセスしてコレクションアイテムを順に確認するには、`componentName.collectionVariable.getIterator()` を使用します。

6. [制御]-[ムービープレビュー]を選択してシェルフをクリックし、[出力]パネルでコレクションデータを確認します。

コレクションが定義されているコンポーネントの SWC ファイルへの書き出し

コレクションが定義されているコンポーネントを配布する場合は、SWC ファイルに次の依存ファイルが含まれている必要があります。

- Collection インターフェイス
- コレクション実装クラス
- コレクションアイテムクラス
- Iterator インターフェイス

これらのファイルのうち、コードで一般的に使用されるのは、Collection インターフェイスと Iterator インターフェイスです。これらのインターフェイスは、コードで依存クラスとしてマークされます。Flash では、SWC ファイルと出力された SWF ファイルに依存ファイルが自動的に追加されます。

ただし、コレクション実装クラス、つまり `mx.utils.CollectionImpl` とコンポーネント固有のコレクションアイテムクラスは、SWC ファイルに自動的に追加されません。

コレクション実装クラスとコレクションアイテムクラスを SWC ファイルに追加するには、コンポーネントの ActionScript でプライベート変数を定義します。次に例を示します。

```

// コレクションアイテムクラス
private var collItem:CompactDisc;
// コレクション実装クラス
private var coll:mx.utils.CollectionImpl;

```

SWC ファイルの詳細については、[181 ページの「SWC ファイルについて」](#)を参照してください。

コレクションプロパティが定義されているコンポーネントの使用

コレクションプロパティが定義されているコンポーネントを使用する場合は、通常 [値] ダイアログボックスを使用してコレクション内のアイテムを設定します。

コレクションプロパティが定義されているコンポーネントを使用するには：

1. コンポーネントをステージに追加します。
2. プロパティインスペクタを使用してコンポーネントのインスタンス名を指定します。
3. [コンポーネントインスペクタ] パネルを開いて [パラメータ] タブを表示します。
4. コレクションプロパティを含む行をクリックし、行の右側にある虫めがねのアイコンをクリックします。
[値] ダイアログボックスが表示されます。
5. [追加](+) ボタンをクリックすると、コレクションアイテムを定義できます。
6. [追加](+) ボタン、[削除](-) ボタン、矢印ボタンを使用すると、コレクションアイテムを追加、変更、移動、削除できます。
7. [OK] をクリックします。

コレクションにプログラムを使用してアクセスする方法については、[193 ページの「コレクションアイテムへの Flash アプリケーションによる実行時アクセス」](#)を参照してください。

索引

A

ActionScript クラスファイル 143

C

className 変数 146

Collection タグ 156

components

creating subobjects 161

ComponentTask タグ

JavaScript (JSFL) 158

createClassObject() method 161

CSSStyleDeclaration 87, 88

D

DataGrid コンポーネント

DataSet へのバインディング

(チュートリアル) 29

列の追加 (チュートリアル) 31

DataSet コンポーネント、XMLConnector および

DataGrid へのバインディング

(チュートリアル) 29

defaultPushButton プロパティ 62

Delegate クラス (チュートリアル) 76

DepthManager クラス、概要 62

Dial コンポーネント 127, 172

draw() メソッド、定義 164

E

Event メタデータタグ 153

F

Flash JavaScript (JSFL)、

ComponentTask タグ 158

Flash MX のエディションと使用可能なコン

ポーネント 12

Flash Player

コンポーネント 17

サポート 66

Flash ファイル、ActionScript ファイル、

SWC ファイルの各アイコン 126

FocusManager クラス、フォーカスナビゲーションの

作成 61

G

getter/setter メソッド 148

H

Halo テーマ 108

handleEvent コールバック関数 72

I

import ステートメント 146

init() メソッド、定義 161

Inspectable パラメータ 150

invalidate() メソッド 165

L

Label コンポーネント

チュートリアル 39

M

MovieClip クラス、継承 138

O

on() イベントハンドラ 80

P

prototype 107

S

Sample テーマ 108

ScrollPane コンポーネント

チュートリアル 39

size() メソッド、定義 164

StandardComponents のライブラリ 140

subobjects, creating 161

SWC ファイル

書き出し 182

コレクションプロパティの書き出し 194

コンパイルされたクリップ 19

説明 19

テスト 183

ファイル形式 181

読み込み 183

symbolName 変数 146

symbolOwner 変数 146

T

TextInput コンポーネント (チュートリアル) 39

U

UIComponent クラス

概要 137

コンポーネントの継承 18

UIObject クラス

説明 137

W

Web サービス、接続 (チュートリアル) 28

WebService クラス (チュートリアル) 28

X

XMLConnector コンポーネント

DataSet コンポーネントへのバインディング
(チュートリアル) 29

外部 XML ファイルのロード (チュートリアル) 31

スキーマの指定 (チュートリアル) 29

あ

アイコン、コンポーネント用 184

アクセシビリティ 20

値ダイアログボックス 187

アップグレード、バージョン 1 コンポーネント 66

い

イベント

各コンポーネント名も参照

一般的 166

イベントオブジェクト 79

スコープの委譲 76

説明 67

送出 165

ハンドラ関数 67

メタデータ 153

イベントリスナー、「リスナー」を参照

インスタンス

スタイル宣言 82

スタイルを設定 85

インストール、コンポーネント 12

う

ウィザード 158

お

親クラス、選択 136

か

書き出し、コンポーネント 180

カスタマイズ

テキスト 82

カスタマイズ、色とテキスト、スタイルシートの使用
82

カラー

カスタマイズ 82

スタイルプロパティの設定 93

く

クラス

- UIComponent 137
- UIObject 137
- 親クラスの選択 136
- 継承 137
 - コンポーネントの継承 18
- 作成、「コンポーネントの作成」を参照
- 参照の作成 (チュートリアル) 25
- 定義 146
- 読み込み 146
- クラススタイルシート 82
- クラスのキーワード 146
- クラスファイル
 - 説明 144
 - 例 143, 189
- グリッド、「DataGrid コンポーネント」を参照
- グローバルスタイル宣言 87

け

- 継承、クラス 137
- 継承、バージョン 2 コンポーネント 18

こ

- 高度なアンチエイリアスはサポートされない 19
- コードヒント、トリガ 58
- コレクションアイテム 191
- コレクションプロパティ
 - クラスの定義 191
 - コンポーネントの書き出し 194
 - 使用 195
 - 定義 188
 - プログラムによるアクセス 192
 - 例 189
- コンパイルされたクリップ
 - 説明 19
 - ライブラリパネル 54
- コンポーネント 143
 - 各コンポーネント名も参照
 - ActionScript クラス 143
 - className 変数 146
 - draw() メソッドの定義 164
 - Flash MX の各エディションで使用可能 12
 - Flash Player サポート 17
 - Flash ドキュメントへの追加 50
 - getter/setter メソッド 148
 - init() メソッドの定義 161

- invalidation、説明 165
- size() メソッドの定義 164
- SWC としてのコンポーネントの書き出し 182
- SWC ファイルの書き出し 182
- SWC ファイルの読み込み 183
- symbolOwner 変数 146
- アーキテクチャ 17
- アイコンの追加 184
- アプリケーションでの使用 (チュートリアル) 21
- 一般的なイベント 166
- イベント 67
- インストール 12
- 親クラスの選択 136
- 開発チェックリスト 185
- 書き出しと配布 180
- カテゴリ、説明 16
- クラスの概要 144
- クラスファイルの例 143
- 継承 18
- 継承、クラス 137
- 構造 126
- コレクションを使用したクラスファイルの例 189
- コンポーネントの作成例 127、172
- 削除 57
- 実行時の追加 52
- シンボル名の選択 146
- スキンの割り当て 167
- スタイル 168
- スタイルへのスキンの登録 169
- 送出、イベント 165
- ソースファイル 125
- ツールチップの追加 184
- テスト、SWC ファイル 183
- パラメータの定義 159
- プリロード 64
- プレビュー 63
- 変数の定義 147
- ムービークリップの作成 138
- ムービークリップの編集 140
- メタデータタグ 149
- メタデータ、ComponentTask タグ 158
- ロード 65
- コンポーネントインスペクタ
 - バインディングタブ 30
 - パラメータの設定 54
- コンポーネントウィザード 158
- コンポーネントのクラスファイル、
「クラスファイル」を参照
- コンポーネントパネル 50

コンポーネントパラメータ
各コンポーネント名も参照
Inspectable 150
設定 55
説明 54
定義 159
表示 55

さ

削除、コンポーネント 57
サブクラス、スキンの変更に使用 107
サブコンポーネント、スキンの適用 103

し

システム要件、コンポーネント 8
情報源、Macromedia に関する 9

す

スーパークラスのキーワード 146
スキン
各コンポーネント名も参照
コンポーネントへの適用 102
サブコンポーネントへの適用 103
編集 99
変数の作成 167
リンケージ識別子 97
スキンの適用、コンポーネント 96
スキンのプロパティ
設定 97
プロトタイプ内で変更 107
スクリーン
ActionScript とのやり取り 59
ActionScript の使用 58
インスタンス名 59
基準点 59
コンポーネントの使用 61
スクリーンリーダー、アクセシビリティ 20
スタイル
各コンポーネント名も参照
インスタンスで設定 85
カスタムの設定 88
グローバルに設定 87
コンポーネントの作成 168
使用 (チュートリアル) 27
設定 82
説明 82
優先順位の決定 92

スタイルシート
カスタム 82
クラス 82
スタイル宣言
カスタム 88
クラスに設定 89
グローバル 87
デフォルトのクラス 89
スタイルプロパティ、カラー 93

そ

送出、イベント 165

た

対象となる読者 8
タグ、「メタデータ」を参照
タブ 61

て

ディスパッチャー (イベントブロードキャスター) 68
データ型、インスタンス用の設定 (チュートリアル) 27
データグリッド、「DataGrid コンポーネント」を参照
データバインディング、XML ファイル
(チュートリアル) 29
テーマ
作成 112
説明 108
適用 114
テキスト、カスタマイズ 82
テスト、SWC ファイル 183

な

9 スライスはサポートされない 19

は

バージョン 1 コンポーネント、アップグレード 66
バージョン 2 コンポーネント
Flash Player 17
継承 18
メリット 15
バインディングタブ、サンプルアプリケーション
(チュートリアル) 30
パッケージ 17
パラメータ、「コンポーネントパラメータ」を参照
ハンドラ関数 67

ひ

ビットマップキャッシュはサポートされない 19
表記規則 9
ヒントテキスト、追加 184

ふ

プリロード、コンポーネント 64
プレビュー、コンポーネント 63
ブロードキャスター 68
プロパティインスペクタ 54

へ

ベストプラクティス、コンポーネント開発 185
変数、定義 147

ま

マニュアル
概要 8
用語のガイド 9
マニュアル内の用語 9

む

ムービークリップ
コンポーネントとしての定義 141
作成 138

め

メタデータ
Collection タグ 156
ComponentTask タグ 158
Event タグ 153
Inspectable タグ 150
説明 149
タグ、リスト 150

ら

ライブプレビュー機能 63
ライブラリ
StandardComponents 140
コンパイルされたクリップ 54
ライブラリパネル 54

り

リスナー
オブジェクト 69
関数 72
コンポーネントインスタンスとの併用
(チュートリアル) 36
コンポーネントとの併用(チュートリアル) 32
スコープ 74
説明 68
リンケージ識別子、スキン 97

れ

列
追加(チュートリアル) 31

ろ

ロード、コンポーネント 65

