

使用 ACTIONSCRIPT™ 3.0 组件

© 2007 Adobe Systems Incorporated。保留所有权利。

使用 ActionScript™ 3.0 组件

如果本指南是随包括最终用户协议的软件分发的，那么，本指南及其所描述的软件将按照该许可协议提供，而且必须遵照该许可协议的条款来使用或复制。除非该许可协议允许，否则，未经 **Adobe Systems Incorporated** 书面许可，不得以任何形式或任何方法（电子、机械、录制或其它方法）对本指南中的任何部分进行复制、存储到检索系统中或者进行传播。请注意，本指南中的内容受版权法保护，即使它不随包括最终用户许可协议的软件一起分发也是如此。

本指南的内容仅供参考，如有更改恕不另行通知，而且不应被理解为 **Adobe Systems Incorporated** 的承诺。对于本指南的信息内容中可能出现的任何错误或不确切之处，**Adobe Systems Incorporated** 不承担任何责任。

请注意，您可能希望包括在您的项目中的现有插图或图像可能受版权法的保护。如果在未经授权的情况下将这些材料包括到您的新作品中，则可能会侵害版权所有者的权利。请确保从版权所有那里获得了任何必需的许可。

在范例模板中公司名称的任何引用都仅用于演示目的，而绝不涉及任何实际的组织。

Adobe、**Adobe** 徽标、**ActionScript**、**Flash**、**Flash Player** 和 **Flash Video** 是 **Adobe Systems Incorporated** 在美国和 / 或其它国家（地区）的注册商标或商标。

Macintosh 是 **Apple Inc.** 在美国和其它国家（地区）的注册商标。**Windows** 是 **Microsoft Corporation** 在美国和 / 或其它国家（地区）的注册商标或商标。其它所有商标都是其各自所有者的财产。

本产品包括 **Apache Software Foundation** (<http://www.apache.org/>) 开发的软件。由 **Fraunhofer IIS** 和 **Thomson Multimedia** (<http://www.iis.fhg.de/amm/>) 许可的 **MPEG Layer-3** 音频压缩技术。无法使用软件中的 **MP3** 压缩音频来进行实时广播。如果您需要 **MP3** 解码器来进行实时广播，请自行负责获取此 **MP3** 技术许可。语音压缩和解压缩技术已得到 **Nellymoser, Inc.** (www.nellymoser.com) 的许可。**Flash CS3** 视频采用了 **On2 TrueMotion** 视频技术。

© 1992-2005 **On2 Technologies, Inc.** 保留所有权利。<http://www.on2.com>。本产品包括 **OpenSymphony Group** (<http://www.opensymphony.com/>) 开发的软件。



Sorenson Spark™ 视频压缩和解压缩技术由 **Sorenson Media, Inc.** 授权。

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

美国政府最终用户须知。按照 48 C.F.R. §2.101 中定义的条款，由“商业计算机软件”和“商业计算机软件文档”组成的软件和文档是“商业项目”，48 C.F.R. §12.212 或 48 C.F.R. §227.7202 中也具有此类条款（如果适用）。与 48 C.F.R. §12.212 或 48 C.F.R. §227.7202-1 到 227.7202-4（如果适用）保持一致，商业计算机软件和商业计算机软件文档将以如下方式授权给美国政府最终用户：(a) 仅作为商业项目 (b) 仅具有依照本协议中的条款和条件授予所有其他最终用户的权利。依据美国版权法保留未公布的权限。**Adobe Systems Incorporated**, 345 Park Avenue, San Jose, CA 95110-2704, USA。对于美国政府最终用户，**Adobe** 同意遵守所有适用的平等机会法规，其中包括（如果适用的话）Executive Order 11246（修订版）、1974 年的 **Vietnam Era Veterans Readjustment Assistance Act** 第 402 部分 (38 USC 4212)、1973 年 **Rehabilitation Act**（修订版）第 503 部分的规定以及 41 CFR 中第 60-1 到 60-60、60-250 和 60-741 部分的规则。前一句中提到的确认性行动条款和规则可以引用至本协议。

目 录

简介	11
目标读者	11
系统要求	12
关于本文档	12
印刷惯例	12
本手册中使用的术语	12
其它资源	13
第 1 章：关于 ActionScript 3.0 组件	15
使用组件的优点	16
组件类型	17
在文档中添加和删除	19
删除组件	21
查找版本	21
ActionScript 3.0 事件处理模型	22
一个简单的应用程序	23
应用程序的设计	23
创建 Greetings 应用程序	24
运行后续示例	30
第 2 章：使用组件	31
组件体系结构	31
ActionScript 3.0 基于 FLA 的组件	32
基于 SWC 的组件	33
ActionScript 3.0 组件 API	34
使用组件文件	34
组件文件的存储位置	34
组件源文件的存储位置	35
组件源文件和类路径	36
修改组件文件	36
调试组件应用程序	37
设置参数和属性	38
在 ActionScript 中设置组件属性	39

库	40
调整组件大小	42
实时预览	43
处理事件	43
关于事件侦听器	44
关于事件对象	44
使用显示列表	45
向显示列表添加组件	46
移动显示列表中的组件	46
从显示列表中删除组件	47
使用 FocusManager	48
使用基于 List 的组件	49
使用 DataProvider	50
创建 DataProvider	50
使用 dataProvider 参数	50
使用 ActionScript	52
操作 DataProvider	55
使用 CellRenderer	59
设置单元格格式	59
定义自定义 CellRenderer 类	60
CellRenderer 属性	65
为 DataGrid 对象的列应用 CellRenderer	65
为可编辑单元格定义 CellRenderer	66
将图像、SWF 文件或影片剪辑用作 CellRenderer	66
使组件具有辅助功能	66
 第 3 章：使用 UI 组件	 69
使用 Button 组件	69
与 Button 进行用户交互	70
Button 参数	70
创建具有 Button 组件的应用程序	71
使用 CheckBox 组件	73
与 CheckBox 进行用户交互	73
CheckBox 参数	74
创建具有 CheckBox 组件的应用程序	74
使用 ColorPicker 组件	76
与 ColorPicker 组件进行用户交互	76
ColorPicker 参数	76
创建具有 ColorPicker 组件的应用程序	77
使用 ComboBox 组件	78
与 ComboBox 组件进行用户交互	79
ComboBox 参数	79
创建具有 ComboBox 组件的应用程序	80

使用 DataGridView 组件	82
与 DataGridView 组件进行用户交互	82
DataGridView 参数	84
创建具有 DataGridView 组件的应用程序	84
使用 Label 组件	88
与 Label 组件进行用户交互	88
Label 参数	88
创建具有 Label 组件的应用程序	88
使用 List 组件	90
与 List 组件进行用户交互	90
List 参数	91
创建具有 List 的应用程序	92
使用 NumericStepper	95
用户与 NumericStepper 的交互	95
NumericStepper 参数	96
创建具有 NumericStepper 的应用程序	96
使用 ProgressBar	98
用户与 ProgressBar 的交互	98
ProgressBar 参数	99
创建具有 ProgressBar 的应用程序	99
使用 RadioButton	104
用户与 RadioButton 的交互	104
RadioButton 参数	105
创建具有 RadioButton 的应用程序	105
使用 ScrollPane 组件	107
用户与 ScrollPane 的交互	108
ScrollPane 参数	108
创建具有 ScrollPane 的应用程序	109
使用 Slider	110
用户与 Slider 组件的交互	111
Slider 参数	111
创建具有 Slider 的应用程序	111
使用 TextArea	113
用户与 TextArea 的交互	114
TextArea 参数	114
创建具有 TextArea 的应用程序	115
使用 TextInput	116
用户与 TextInput 的交互	117
TextInput 参数	117
创建具有 TextInput 的应用程序	117
使用 TileList	120
与 TileList 的用户交互	120
TileList 参数	121
创建具有 TileList 的应用程序	121

使用 UILoader.....	123
与 UILoader 的用户交互.....	123
UILoader 参数.....	123
创建具有 UILoader 的应用程序.....	124
使用 UIScrollView.....	125
与 UIScrollView 的用户交互.....	125
UIScrollView 参数.....	125
创建具有 UIScrollView 的应用程序.....	125
第 4 章：自定义 UI 组件.....	129
关于 UI 组件自定义.....	130
设置样式.....	130
了解样式设置.....	131
访问组件的默认样式.....	131
在组件实例上设置和获取样式.....	131
使用 TextFormat 设置文本属性.....	132
为组件的所有实例设置样式.....	132
为所有组件设置样式.....	133
关于外观.....	133
创建新外观.....	136
为所有实例创建外观.....	136
为部分实例创建外观.....	136
自定义 Button.....	137
对 Button 使用样式.....	138
对 Button 使用外观.....	139
自定义 CheckBox.....	140
对 CheckBox 使用样式.....	140
对 CheckBox 使用外观.....	141
自定义 ColorPicker.....	142
对 ColorPicker 使用样式.....	142
对 ColorPicker 使用外观.....	143
自定义 ComboBox.....	144
对 ComboBox 使用样式.....	145
对 ComboBox 使用外观.....	146
自定义 DataGrid.....	147
对 DataGrid 使用样式.....	147
为单个列设置样式.....	147
设置标题样式.....	150
对 DataGrid 使用外观.....	151
自定义 Label.....	153
对 Label 使用样式.....	153
对 Label 使用外观.....	153

自定义 List	154
对 List 使用样式	154
对 List 使用外观	155
自定义 NumericStepper	157
对 NumericStepper 使用样式	157
对 NumericStepper 使用外观	158
自定义 ProgressBar	159
对 ProgressBar 使用样式	159
对 ProgressBar 使用外观	160
自定义 RadioButton	161
对 RadioButton 使用样式	161
对 RadioButton 使用外观	162
自定义 ScrollPane	163
对 ScrollPane 使用样式	163
对 ScrollPane 使用外观	164
自定义 Slider	164
对 Slider 使用样式	164
对 Slider 使用外观	165
自定义 TextArea	166
对 TextArea 使用样式	166
对 TextArea 使用外观	167
自定义 TextInput	168
对 TextInput 使用样式	168
对 TextInput 使用外观	169
自定义 TileList	170
对 TileList 使用样式	170
对 TileList 使用外观	171
自定义 UILoader	172
自定义 UIScrollBar	172
对 UIScrollBar 使用样式	173
对 UIScrollBar 使用外观	173
第 5 章：使用 FLVPlayback 组件	175
使用 FLVPlayback 组件	175
创建具有 FLVPlayback 组件的应用程序	177
FLVPlayback 组件参数	179
指定 source 参数	180
使用实时预览	182
全屏支持	182
用于播放多个 FLV 文件的布局对齐方式	183
自动播放渐进式下载的 FLV 文件	183

使用提示点	183
使用“Flash 视频提示点”对话框	184
在 ActionScript 中使用提示点	185
添加 ActionScript 提示点	186
侦听 cuePoint 事件	186
查找提示点	187
搜索导航提示点	188
启用和禁用嵌入式 FLV 文件提示点	188
删除 ActionScript 提示点	189
播放多个 FLV 文件	190
使用多个视频播放器	190
从 Flash Media Server 流式加载 FLV 文件	192
关于本机带宽检测或无带宽检测	192
关于非本机带宽检测	193
自定义 FLVPlayback 组件	193
选择预先设计的外观	194
分别在各个 FLV 回放自定义用户界面组件内设置外观	195
按钮组件	196
BufferingBar 组件	197
SeekBar 和 VolumeBar 组件	197
连接 FLV 回放自定义用户界面组件	200
创建新外观	202
使用外观布局	202
缓冲栏	204
搜索栏和音量栏	205
背景剪辑和前景剪辑	206
修改外观行为	206
使用 SMIL 文件	207
<smil>	208
<head>	209
<meta>	210
<layout>	210
<root-layout>	211
<body>	212
<video>	213
<ref>	213
<switch>	214
 第 6 章：使用 FLVPlayback 字幕组件	 215
使用 FLVPlaybackCaptioning 组件	215
将字幕添加到 FLVPlayback 组件	215
设置 FLVPlaybackCaptioning 组件参数	217
指定 source 参数	217
显示字幕	218

使用 Timed Text 字幕	218
将提示点用于字幕	220
了解 FLVPlaybackCaptioning 提示点标准	220
了解如何为嵌入的事件提示点创建字幕	221
支持带有嵌入的事件提示点的多语言轨道	222
播放多个带有字幕的 FLV 文件	222
自定义 FLVPlaybackCaptioning 组件	223
 附录 A: Timed Text 标签	 225
 索引	 231

简介

Adobe® Flash® CS3 Professional 是标准的创作工具，可以制作出极富感染力的 Web 内容。组件是制作这些内容的丰富 Internet 应用程序的构建块。“组件”是带有参数的影片剪辑，在 Flash 中进行创作时或在运行时，可以使用这些参数以及 **ActionScript™** 方法、属性和事件自定义此组件。设计这些组件的目的是为了让开发人员重复使用和共享代码，以及封装复杂功能，使设计人员无需编写 **ActionScript** 就能够使用和自定义这些功能。

使用组件可以轻松而快速地构建功能强大且具有一致外观和行为的应用程序。本手册介绍如何使用 **ActionScript 3.0** 组件构建应用程序。《**ActionScript 3.0** 语言和组件参考》中介绍了每种组件的应用程序编程接口 (API)。

您可以使用 Adobe 创建的组件，下载其他开发人员创建的组件，还可以创建自己的组件。

本章包含以下各节：

目标读者	11
系统要求	12
关于本文档	12
印刷惯例	12
本手册中使用的术语	12
其它资源	13

目标读者

本手册的目标读者是要构建 **Flash** 应用程序并希望使用组件加快开发速度的开发人员。您应当已经熟悉如何开发 **Flash** 应用程序和编写 **ActionScript**。

如果对编写 **ActionScript** 还不够熟练，您可以向文档添加组件，在“属性”检查器或“组件”检查器中设置其参数，然后使用“行为”面板处理其事件。例如，您无需编写任何 **ActionScript** 代码，就可以将“转到网页”行为附加到一个 **Button** 组件，用户点击此按钮时会在 Web 浏览器中打开一个 URL。

如果希望创建功能更加强大的应用程序，则可通过动态方式创建组件，使用 **ActionScript** 在运行时设置属性和调用方法，还可使用事件侦听器模型来处理事件。

有关详细信息，请参阅第 31 页的第 2 章“使用组件”。

系统要求

Flash 组件的系统要求与 Flash 的系统要求完全相同。

使用 Flash CS3 组件的 SWF 文件必须用 Adobe® Flash® Player 9.0.28.0 或更高版本查看，而且必须以 ActionScript 3.0 发布（可以通过“文件”>“发布设置”中的“Flash”选项卡进行设置）。

关于本文档

本文档详细说明了如何使用组件开发 Flash 应用程序。本文档的目标读者需已掌握 Flash 和 ActionScript 3.0 的一般性知识。Flash 和相关产品的专用文档将单独提供。

本文档以 PDF 格式和在线帮助的形式提供。若要查看在线帮助，请启动 Flash，然后选择“帮助”>“Flash 帮助”>“使用 ActionScript 3.0 组件”。

有关 Flash 的信息，请参阅下列文档：

- 《使用 Flash》
- 《ActionScript 3.0 编程》
- 《ActionScript 3.0 语言和组件参考》

印刷惯例

本手册使用以下印刷惯例：

- *斜体字体* 表示应被替换的值（例如，在文件夹路径中）。
- 代码字体 (Code font) 表示 ActionScript 代码，其中包括方法和属性名称。
- *斜体代码字体* 表示应被替换的代码项目（例如，一个 ActionScript 参数）。
- **粗体字体** 表示您输入的值。

本手册中使用的术语

在本手册中使用以下术语：

在运行时 代码在 Flash Player 中运行时。

在创作时 在 Flash 创作环境中工作时。

其它资源

除了这些手册中的内容以外，Adobe 还在 Adobe 开发人员中心和 Adobe 设计中心上提供定期更新的文章、设计思路和示例。

访问 www.adobe.com/go/learn_fl_samples_cn 可以找到其它组件示例。

Adobe 开发人员中心

在 Adobe 开发人员中心中，您可以找到有关 **ActionScript** 的最新信息、有关实际应用程序开发的文章以及有关新出现的重大问题的信息。开发人员中心的网址为

www.adobe.com/go/flash_devcenter_cn。

Adobe 设计中心

提供了有关数字设计和动画图形的最新信息。在该中心，可以浏览主流艺术家的杰作，了解设计新趋势，并可以通过教程、关键工作流程和高级技巧使您的技能更加精湛。请每月访问该中心两次，查看其中新的教程和文章以及可激发灵感的艺术作品。设计中心的网址为

www.adobe.com/go/fl_designcenter_cn。

关于 ActionScript 3.0 组件

Adobe® Flash® CS3 Professional 组件是带参数的影片剪辑，您可以修改它们的外观和行为。组件可以是一个简单的用户界面控件（如 `RadioButton` 或 `CheckBox`），也可以包含内容（如 `List` 或 `DataGrid`）。

组件使您可以方便而快速地构建功能强大且具有一致外观和行为的应用程序。您可以使用 Flash 组件实现这些控件，而不用创建自定义按钮、组合框和列表。只需将这些组件从“组件”面板拖到应用程序文档中即可。您还可以方便地自定义这些组件的外观和直观感受，从而适合您的应用程序设计。

即使对 ActionScript 没有深入的理解您也可以进行所有这些工作，还可以使用 ActionScript 3.0 修改组件的行为或实现新的行为。每个组件都有一组唯一的 ActionScript 方法、属性和事件，它们构成了此组件的“应用程序编程接口”（API）。API 允许您在应用程序运行时创建并操作组件。

API 还允许您创建自己的新的自定义组件。您可以从 Adobe Exchange（网址为 http://www.adobe.com/go/flash_exchange_cn）下载由 Flash 社区成员构建的组件。有关创建组件的信息，请参阅 www.adobe.com/go/learn_fl_creating_components_cn。

ActionScript 3.0 组件体系结构包括所有组件基于的类、允许您自定义外观的外观和样式、事件处理模型、焦点管理、辅助功能接口等等。

提醒

Adobe Flash CS3 包括 ActionScript 2.0 组件以及 ActionScript 3.0 组件。不能将这两组组件混合。对于给定的应用程序，您只能使用其中的一组。根据您打开的是 ActionScript 2.0 文件还是 ActionScript 3.0 文件，Flash CS3 将显示 ActionScript 2.0 组件或 ActionScript 3.0 组件。创建新的 Flash CS3 文档时，必须指定一个 Flash 文件（ActionScript 3.0 或 ActionScript 2.0）。打开现有文档时，Flash 会检查“发布设置”以确定要使用哪组组件。有关 ActionScript 2.0 组件的信息，请参阅《使用 ActionScript 2.0 组件》。

有关 Flash ActionScript 3.0 组件的完整列表，请参阅第 17 页的“组件类型”。

本章包含以下各节：

使用组件的优点	16
组件类型	17
在文档中添加和删除	19
查找版本	21
ActionScript 3.0 事件处理模型	22
一个简单的应用程序	23

使用组件的优点

组件使您可以将应用程序的设计过程和编码过程分开。通过使用组件，开发人员可以创建设计人员在应用程序中能用到的功能。开发人员可以将常用功能封装到组件中，而设计人员可以通过更改组件的参数来自定义组件的大小、位置和行为。通过编辑组件的图形元素或外观，还可以更改组件的外观。

组件之间共享核心功能，如样式、外观和焦点管理。将第一个组件添加至应用程序时，此核心功能大约占用 20 千字节的大小。当您添加其它组件时，添加的组件会共享初始分配的内存，降低应用程序大小的增长。

本部分概括介绍了 ActionScript 3.0 组件的一些优点。

ActionScript 3.0 的强大功能提供了一种强大的、面向对象的编程语言，这是 Flash Player 功能发展过程中重要的一步。该语言的设计意图是，在可重用代码的基础上构建丰富的 Internet 应用程序。ActionScript 3.0 基于 ECMAScript（编写脚本的国际标准化语言）。它符合 ECMAScript (ECMA-262) 第 3 版语言规范 (ECMAScript (ECMA-262) edition 3 language specification)。有关 ActionScript 3.0 的详细介绍，请参阅《ActionScript 3.0 编程》。有关该语言的参考信息，请参阅《ActionScript 3.0 语言和组件参考》。

基于 placeStateFLA 的用户界面组件提供对外观的轻松访问，以便在创作时进行方便的自定义。这些组件还提供样式（包括外观样式），您可以利用样式来自定义组件的某些外观，并在运行时加载外观。有关详细信息，请参阅第 129 页的第 4 章“自定义 UI 组件”和《ActionScript 3.0 语言和组件参考》。

新的 FVLPlayback 组件添加 FLVPlaybackCaptioning 组件及全屏支持、改进的实时预览、允许您添加颜色和 Alpha 设置的外观，以及改进的 FLV 下载和布局功能。

“属性”检查器和“组件”检查器允许您在 Flash 中进行创作时更改组件参数。有关详细信息，请参阅第 19 页的“在文档中添加和删除”和第 38 页的“设置参数和属性”。

ComboBox、List 和 TileList 组件的新集合对话框允许您通过用户界面填充它们的数据Provider 属性。有关详细信息，请参阅第 50 页的“创建 DataProvider”。


ActionScript 3.0 事件模型允许您的应用程序侦听事件并调用事件处理函数进行响应。有关详细信息，请参阅第 22 页的“[ActionScript 3.0 事件处理模型](#)”和第 43 页的“[处理事件](#)”。

管理器类：提供了一种在应用程序中处理焦点和管理样式的简便方法。有关详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》。

UIComponent 基类为扩展它的组件提供核心方法、属性和事件。所有的 ActionScript 3.0 用户界面组件继承自 UIComponent 类。有关详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 UIComponent 类。

在基于 placeCityUI StateFLA 的组件中使用 SWC 可提供 ActionScript 定义（作为组件的时间轴内部的资源），用以加快编译速度。

便于扩展的类层次体系结构使用 ActionScript 3.0，可以创建唯一的命名空间，按需要导入类，并且可以方便地创建子类来扩展组件。有关详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》。



Flash CS3 既支持基于 FLA 的组件，又支持基于 SWC 的组件。有关详细信息，请参阅第 31 页的“[组件体系结构](#)”。

组件类型

在安装 Flash CS3 时安装 Flash 组件。

ActionScript 3.0 组件包括下列用户界面 (UI) 组件：

UIComponent		
Button	List	TextArea
CheckBox	NumericStepper	TextInput
ColorPicker	RadioButton	TileList
ComboBox	ProgressBar	UILoader
DataGrid	ScrollPane	UIScrollBar
Label	Slider	

除了用户界面组件，Flash ActionScript 3.0 组件还包括下列组件和支持类：

- **FLVPlayback 组件** (fl.video.FLVPlayback)，它是基于 SWC 的组件。
FLVPlayback 组件使您可以轻松将视频播放器包括在 Flash 应用程序中，以便通过 HTTP 从 Adobe® Flash® Video Streaming Service (FVSS) 或从 Adobe 的 Macromedia® Flash® Media Server (FMS) 播放渐进式视频流。有关详细信息，请参阅第 175 页的第 5 章“[使用 FLVPlayback 组件](#)”。

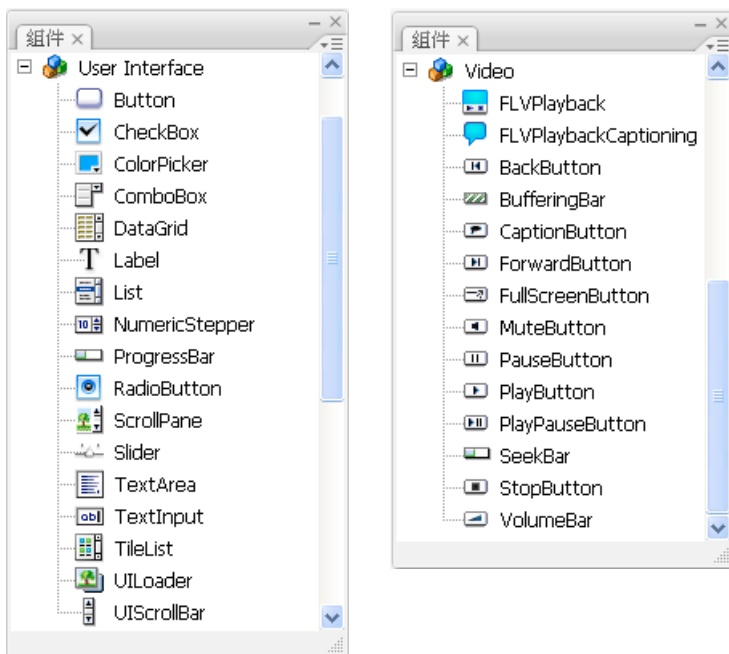
- FLVPlayback 自定义 UI 组件，基于 FLA，同时使用于 FLVPlayback 组件的 ActionScript 2.0 和 ActionScript 3.0 版本。有关详细信息，请参阅第 175 页的第 5 章“使用 FLVPlayback 组件”。
- FLVPlayback Captioning 组件，为 FLVPlayback 提供关闭的字幕。第 215 页的第 6 章“使用 FLVPlayback 字幕组件”

有关 ActionScript 3.0 组件类及其支持类的完整列表，请参阅《ActionScript 3.0 语言和组件参考》。

查看 Flash 组件：

您可以按如下步骤在“组件”面板中查看 Flash ActionScript 3.0 组件。

1. 启动 Flash。
2. 创建新的 Flash 文件 (ActionScript 3.0) 或打开现有的 Flash 文档（其“发布设置”中指定了 ActionScript 3.0）。
3. 如果“组件”面板没有打开，请选择“窗口”>“组件”打开它。



带有用户界面组件和视频组件的“组件”面板

用户界面组件和视频组件分别显示以节省空间。“组件”面板包含显示的所有组件。

您还可以从 Adobe Exchange（网址为 http://www.adobe.com/go/flash_exchange_cn）下载其它的组件。要安装从 Exchange 下载的组件，请下载并安装 Adobe® Extension Manager，网址为 http://www.adobe.com/go/exchange_cn。单击“Adobe Exchange 主页”链接并查找“Extension Manager”链接。

所有组件都可以在 Flash 的“组件”面板上显示。若要在 Windows® 或 Macintosh® 计算机上安装组件，请遵循以下步骤。

在基于 Windows 的计算机上或 Macintosh 计算机上安装组件：

1. 退出 Flash。
2. 将包含组件的 SWC 或 FLA 文件放在硬盘上的以下文件夹中：
 - 在 Windows 中：
C:\Program Files\Adobe\Flash CS3\language\Configuration\Components
 - 在 Macintosh 上：
Macintosh HD:Applications:Adobe Flash CS3:Configuration:Components
3. 启动 Flash。
4. 如果“组件”面板尚未打开，请选择“窗口”>“组件”，以在“组件”面板中查看组件。有关组件文件的详细信息，请参阅第 34 页的“使用组件文件”

在文档中添加和删除

将基于 FLA 的组件从“组件”面板拖到舞台上时，Flash 会将一个可编辑的影片剪辑导入到库中。将基于 SWC 的组件拖到舞台上时，Flash 会将一个已编译的剪辑导入到库中。将组件导入到库中后，您可以将组件的实例从“库”面板或“组件”面板拖到舞台。

在创作时添加组件

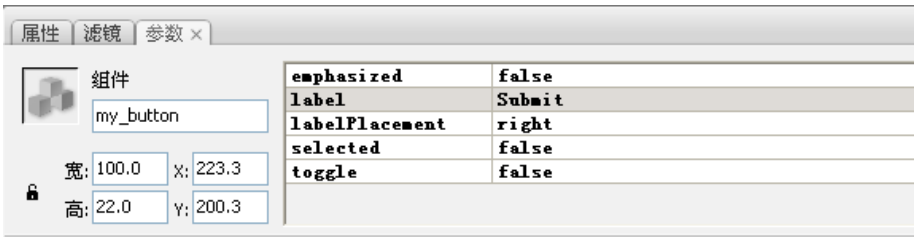
通过从“组件”面板拖动组件，可以将组件添加到文档中。在“属性”检查器的“参数”选项卡或“组件”检查器中的“参数”选项卡中可以设置组件每个实例的属性。

使用“组件”面板向 Flash 文档添加组件：

1. 选择“窗口”>“组件”。
2. 双击“组件”面板中的组件，或将组件拖到舞台。
3. 在舞台上选择该组件。
4. 如果看不到“属性”检查器，请选择“窗口”>“属性”>“属性”。
5. 在“属性”检查器中，输入组件实例的实例名称。

6. 单击“参数”选项卡，然后为实例指定参数。

下图显示了 **Button** 组件的“属性”检查器。



“属性”检查器中的组件设置

有关详细信息，请参阅第 38 页的“设置参数和属性”。

7. 通过编辑宽度 (W:) 和高度 (H:) 的值，按需更改组件的大小。

有关调整特定组件类型大小的详细信息，请参阅第 129 页的第 4 章“自定义 UI 组件”。

8. 选择“控制” > “测试影片”或按 **Ctrl+Enter** 编译文档并查看设置的结果。

您还可以更改组件的颜色和文本格式，方法是设置组件的样式属性，或通过编辑组件的外观自定义其外观。有关这些主题的详细信息，请参阅第 129 页的第 4 章“自定义 UI 组件”。

如果在创作时将组件拖到舞台，使用实例名称（例如，myButton）即可引用该组件。

使用 ActionScript 在运行时添加组件

若要使用 **ActionScript** 在运行时将组件添加到文档，当编译 **SWF** 文件时，组件必须先位于应用程序的“库”（“窗口” > “库”）中。若要将组件添加到“库”中，请将组件从“组件”面板拖到“库”面板中。有关库的详细信息，请参阅第 40 页的“库”。

您还必须导入组件的类文件，以使应用程序可以使用组件的 **API**。组件类文件安装在包含一个或多个类的包中。若要导入组件类，请使用 `import` 语句并指定包名称和类名称。例如，您可以使用下列 `import` 语句导入 **Button** 类：

```
import fl.controls.Button;
```

有关组件所位于包的信息，请参阅《**ActionScript 3.0 语言和组件参考**》。有关组件源文件位置的信息，请参阅第 34 页的“使用组件文件”。

若要创建组件的一个实例，必须调用组件的 **ActionScript** 构造函数方法。例如，下面的语句创建一个名为 `aButton` 的 **Button** 实例：

```
var aButton:Button = new Button();
```

最后一个步骤是调用静态的 `addChild()` 方法将组件实例添加到舞台或应用程序容器。例如，下面的语句添加 `aButton` 实例：

```
addChild(aButton);
```

此时，您可以使用组件的 **API** 动态指定组件的大小和在舞台上的位置、侦听事件，并设置属性以修改组件的行为。有关特定组件的 **API** 的详细信息，请参阅 [《ActionScript 3.0 语言和组件参考》](#)。

有关 `addChild()` 方法的详细信息，请参阅第 45 页的“使用显示列表”。

删除组件

在创作时若要从“舞台”删除组件实例，只需选择该组件，然后按 **Delete** 键即可。这会从“舞台”删除实例，但不会从应用程序中删除该组件。

在您将组件放置在舞台上或“库”中之后，若要从 **Flash** 文档删除组件，您必须从“库”中删除组件及与它关联的“资源”。从“舞台”中删除组件是不够的。如果您未从“库”中删除组件，则在您编译时组件会包括在应用程序中。

从文档中删除组件：

1. 在“库”面板中，选择组件的元素。
2. 单击“库”面板底部的“删除”按钮，或从“库”面板菜单中选择“删除”。

重复这些步骤以删除所有与组件关联的资源。

有关在应用程序运行时如何从组件的容器中删除组件的信息，请参阅第 47 页的“从显示列表中删除组件”。

查找版本

Flash ActionScript 3.0 组件有一个 **version** 属性，如果您需要将此属性提供给 Adobe 技术支持或需要知道正在使用的组件的版本，则可以显示 **version** 属性。

显示用户界面组件的版本号：

1. 创建一个新的 **Flash** 文件 (ActionScript 3.0) 文档。
2. 将组件拖到“舞台”上，然后为组件指定实例名称。例如，将一个 **ComboBox** 组件拖到“舞台”上，然后将其命名为 **aCb**。
3. 按 **F9** 键或选择“窗口” > “动作”以打开“动作”面板。
4. 单击主时间轴的第 1 帧，然后将下面的代码添加到“动作”面板中。

```
trace(aCb.version);
```

版本号（与下图中的版本号相似）应当出现在“输出”面板中。



对于 FLVPlayback 和 FLVPlaybackCaptioning 组件，您必须引用类名而不是实例名，因为版本号存储在类常量中。

显示 FLVPlayback 和 FLVPlaybackCaptioning 组件的版本号：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将 FLVPlayback 和 FLVPlaybackCaptioning 组件拖到 “库” 面板中。
3. 按 “F9” 键或选择 “窗口” > “动作” 以打开 “动作” 面板。
4. 单击主时间轴的第 1 帧，然后将下面的代码添加到 “动作” 面板中。

```
import fl.video.*;
trace("FLVPlayback.VERSION: " + FLVPlayback.VERSION);
trace("FLVPlaybackCaptioning.VERSION: " + FLVPlaybackCaptioning.VERSION);
```

版本号（与下图中的版本号相似）应当出现在 “输出” 面板中。



FLVPlayback 和 FLVPlaybackCaptioning 版本号

ActionScript 3.0 事件处理模型

ActionScript 3.0 引入了单个事件处理模型，替换以前版本的 ActionScript 中存在的不同事件处理机制。该新事件模型基于文档对象模型 (DOM) 第 3 级事件规范。

对于具有使用 ActionScript 2.0 addListener() 方法经验的开发人员，指出 ActionScript 2.0 事件侦听器模型和 ActionScript 3.0 事件模型之间的区别是会有帮助的。下面的列表描述了这两种事件模型之间的一些主要区别：

- 若要在 ActionScript 2.0 中添加事件侦听器，在某些情况下使用 addListener()，其它情况下则使用 addEventListener()，而在 ActionScript 3.0 中，所有情况下都使用 addEventListener()。
- ActionScript 2.0 中没有事件流，这意味着 addListener() 方法只能在广播事件的对象上进行调用，而在 ActionScript 3.0 中，addEventListener() 方法可以在作为事件流一部分的任意对象上进行调用。
- 在 ActionScript 2.0 中，事件侦听器可以是函数、方法或对象，而在 ActionScript 3.0 中，事件侦听器只能是函数或方法。
- ActionScript 3.0 中不再支持 on(event) 语法，因此无法将 ActionScript 事件代码附加到影片剪辑。您只能使用 addEventListener() 添加事件侦听器。

下面的示例（侦听名为 `aButton` 的 **Button** 组件上的 `MouseEvent.CLICK` 事件）演示基本的 **ActionScript 3.0** 事件处理模型：

```
aButton.addEventListener(MouseEvent.CLICK, clickHandler);
function clickHandler(event:MouseEvent):void {
    trace("clickHandler detected an event of type: " + event.type);
    trace("the event occurred on: " + event.target.name);
}
```

有关 **ActionScript 3.0** 事件处理的详细信息，请参阅《**ActionScript 3.0 编程**》。有关 **ActionScript 3.0** 组件事件处理的详细信息，请参阅[第 43 页的“处理事件”](#)。

一个简单的应用程序

本节将指导您使用 **Flash** 组件和 **Flash** 创作工具完成创建简单的 **ActionScript 3.0** 应用程序的步骤。提供的示例既有 **FLA** 文件（其时间轴上包含 **ActionScript** 代码），也有外部的 **ActionScript** 类文件（带有只包含“库”中组件的 **FLA** 文件）。通常，您会希望使用外部类文件开发较大的应用程序，以便在类和应用程序之间共享代码，并使应用程序更易维护。有关使用 **ActionScript 3.0** 编程的详细信息，请参阅《**ActionScript 3.0 编程**》。

应用程序的设计

我们的第一个 **ActionScript** 组件应用程序的示例是标准的“**Hello World**”应用程序的一个变体，因此其设计相当简单：

- 将此应用程序称为“**Greetings**”。
 - 它使用 **TextArea** 显示最初为“**Hello World**”的一个问候。
 - 使用 **ColorPicker** 允许您更改文本的颜色。
 - 使用三个 **RadioButton** 允许您将文本的大小设置为小、较大或最大。
 - 使用 **ComboBox** 允许您从下拉列表选择其它问候。
 - 应用程序使用“组件”面板中的组件，而且还通过 **ActionScript** 代码创建应用程序元素。
- 完成这些定义后，您可以开始构建应用程序。

创建 Greetings 应用程序

下列步骤使用 Flash 创作工具创建 FLA 文件、将组件放置在“舞台”上、向“时间轴”添加 ActionScript 代码，从而创建 Greetings 应用程序。

在 FLA 文件中创建 Greetings 应用程序：

1. 选择“文件”>“新建”。
2. 在“新建文档”对话框中，选择“Flash 文件 (ActionScript 3.0)”，然后单击“确定”。
打开一个新的 Flash 窗口。
3. 选择“文件”>“保存”，将 Flash 文件命名为 **Greetings.fla**，然后单击“保存”按钮。
4. 在 Flash 组件面板中，选择一个 TextArea 组件，并将其拖到“舞台”上。
5. 在“属性”窗口中，选择“舞台”上的 TextArea 后，请键入 **aTa** 作为实例名，然后输入下列信息：
 - 输入 **230** 作为 W 值（宽）。
 - 输入 **44** 作为 H 值（高）。
 - 输入 **165** 作为 X 值（水平位置）。
 - 输入 **57** 作为 Y 值（垂直位置）。
 - 在“参数”选项卡上输入“**Hello World!**”作为文本参数。
6. 将 ColorPicker 组件拖到舞台上，放在 TextArea 的左侧，并为其指定实例名称“txtCp”。
在“属性”检查器中输入下列信息：
 - 输入 **96** 作为 X 值。
 - 输入 **72** 作为 Y 值。
7. 将三个 RadioButton 组件拖到“舞台”上，分别为组件指定实例名称 **smallRb**、**largerRb** 和 **largestRb**。在“属性”检查器中为它们输入下列信息：
 - 为每个组件输入 **100** 作为 W 值，输入 **22** 作为 H 值。
 - 输入 **155** 作为 X 值。
 - 输入 **120** 作为 smallRb 的 Y 值，输入 **148** 作为 largerRb 的 Y 值，输入 **175** 作为 largestRb 的 Y 值。
 - 输入 **fontRbGrp** 作为每个组件的 groupName 参数。
 - 在组件的“参数”选项卡输入 **Small**、**Larger** 和 **Largest** 作为标签。

8. 将一个 **ComboBox** 拖到“舞台”上，并为其指定实例名称 **msgCb**。在“属性”检查器中为其输入下列信息：

- 输入 **130** 作为 **W** 值。
- 输入 **265** 作为 **X** 值。
- 输入 **120** 作为 **Y** 值。
- 在“参数”选项卡上，输入“**Greetings**”作为提示参数。
- 双击 **dataProvider** 参数的文本字段以打开“值”对话框。
- 单击加号，然后用“**Hello World!**”替换标签值
- 重复上一步骤，添加 **Have a nice day!** 和 **Top of the Morning!** 标签值
- 单击“确定”以关闭“值”对话框。

9. 保存该文件。

10. 如果尚未打开，请通过按 **F9** 或从“窗口”菜单选择“动作”以打开“动作”面板。单击主时间轴的第 **1** 帧，然后在“动作”面板中输入下面的代码：

```
import flash.events.Event;
import fl.events.ComponentEvent;
import fl.events.ColorPickerEvent;
import fl.controls.RadioButtonGroup;

var rbGrp:RadioButtonGroup = RadioButtonGroup.getGroup("fontRbGrp");
rbGrp.addEventListener(MouseEvent.CLICK, rbHandler);
txtCp.addEventListener(ColorPickerEvent.CHANGE, cpHandler);
msgCb.addEventListener(Event.CHANGE, cbHandler);
```

前三行导入应用程序使用的事件类。用户与组件之一进行交互时，会发生事件。接下来的五行为应用程序希望侦听的事件注册事件处理函数。用户单击 **RadioButton** 时发生 **click** 事件。用户在 **ColorPicker** 中选择其它颜色时发生 **change** 事件。用户从 **ComboBox** 的下拉列表选择其它问候时发生 **change** 事件。

第四行导入 **RadioButtonGroup** 类以便应用程序可以为一组 **RadioButton** 分配事件侦听器，而不是分别为每个按钮分配侦听器。

11. 将下面一行代码添加到“动作”面板以创建 **tf TextFormat** 对象，应用程序使用此对象更改 **TextArea** 中文本的 **size** 和 **color** 样式属性。

```
var tf:TextFormat = new TextFormat();
```

12. 添加下列代码以创建 `rbHandler` 事件处理函数。在用户单击其中一个 **RadioButton** 组件时，此函数处理 `click` 事件。

```
function rbHandler(event:MouseEvent):void {
    switch(event.target.selection.name) {
        case "smallRb":
            tf.size = 14;
            break;
        case "largerRb":
            tf.size = 18;
            break;
        case "largestRb":
            tf.size = 24;
            break;
    }
    aTa.setStyle("textFormat", tf);
}
```

此函数使用 `switch` 语句检查 `event` 对象的 `target` 属性，以确定哪个 **RadioButton** 触发了事件。`currentTarget` 属性包含触发事件的对象名称。根据用户单击的是哪个 **RadioButton**，应用程序将 **TextArea** 中文本的大小更改为 14、18 或 24 磅。

13. 添加下列代码以实现 `cpHandler()` 函数，此函数处理 **ColorPicker** 中的值的更改：

```
function cpHandler(event:ColorPickerEvent):void {
    tf.color = event.target.selectedColor;
    aTa.setStyle("textFormat", tf);
}
```

此函数将 `tf` **TextFormat** 对象的 `color` 属性设置为 **ColorPicker** 中选定的颜色，然后调用 `setStyle()` 将此颜色应用到 `aTa` **TextArea** 实例中的文本。

14. 添加下列代码以实现 `cbHandler()` 函数，此函数处理 **ComboBox** 中选择的更改：

```
function cbHandler(event:Event):void {
    aTa.text = event.target.selectedItem.label;
}
```

此函数只是将 **TextArea** 中的文本替换为 **ComboBox** 中选择的文本 (`event.target.selectedItem.label`)。

15. 选择“控制”>“测试影片”或按 **Ctrl+Enter** 编译代码，然后测试 **Greetings** 应用程序。

下面的部分向您演示如何使用外部的 **ActionScript** 类，以及其“库”中只有必需组件的 **FLA** 文件构建相同的应用程序。

使用外部类文件创建 Greetings2 应用程序：

1. 选择“文件”>“新建”。
2. 在“新建文档”对话框中，选择“Flash 文件 (ActionScript 3.0)”，然后单击“确定”。
打开一个新的 Flash 窗口。
3. 选择“文件”>“保存”，将 Flash 文件命名为“**Greetings2.fla**”，然后单击“保存”按钮。
4. 将下列各个组件从“组件”面板拖到“库”中：

- ColorPicker
- ComboBox
- RadioButton
- TextArea

因为编译的 SWF 文件会使用所有资源，所以您需要将资源都添加到“库”中。将组件拖到“库”面板的底部。在您将这些组件添加到“库”中时，会自动添加其它资源（List、TextInput 和 UI ScrollBox）。

5. 在“属性”窗口中，为“文档类”键入 **Greetings2**。

如果 Flash 显示一个“无法找到该文档类的定义”的警告，请忽略。您将按下面步骤定义 Greetings2 类。此类定义应用程序的主要功能。

6. 保存 Greetings2.fla 文件。
7. 选择“文件”>“新建”。
8. 在“新建文档”对话框中，选择“ActionScript 文件”，然后单击“确定”。
打开一个新的脚本窗口。

9. 在脚本窗口中添加下列代码：

```
package {
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.MouseEvent;
    import flash.text.TextFormat;
    import fl.events.ComponentEvent;
    import fl.events.ColorPickerEvent;
    import fl.controls.ColorPicker;
    import fl.controls.ComboBox;
    import fl.controls.RadioButtonGroup;
    import fl.controls.RadioButton;
    import fl.controls.TextArea;
    public class Greetings2 extends Sprite {
        private var aTa:TextArea;
        private var msgCb:ComboBox;
        private var smallRb:RadioButton;
        private var largerRb:RadioButton;
```

```

private var largestRb:RadioButton;
private var rbGrp:RadioButtonGroup;
private var txtCp:ColorPicker;
private var tf:TextFormat = new TextFormat();
public function Greetings2() {

```

脚本定义一个名为 **Greetings2** 的 **ActionScript 3.0** 类。脚本进行以下操作：

- 导入我们将要在文件中使用的类。通常情况下，您可以在代码中引用其它类时添加这些导入语句，但为了简便起见，此示例将所有这些语句在一个步骤中导入。
- 声明变量以表示我们将要添加到代码中的组件对象的不同类型。另一个变量创建 **tf** **TextFormat** 对象。
- 为类定义构造函数 **Greetings2()**。我们将这些行添加到此函数中，并按下列步骤向类添加其它方法。

10. 选择“文件” > “保存”，将文件命名为 **“Greetings2.as”**，然后单击“保存”按钮。

11. 向 **Greeting2()** 函数添加下列代码行：

```

createUI();
setUpHandlers();
}

```

此函数现在应该如下所示：

```

public function Greetings2() {
createUI();
setUpHandlers();
}

```

12. 在 **Greeting2()** 方法的右括号后添加下列代码行：

```

private function createUI() {
bldTxtArea();
bldColorPicker();
bldComboBox();
bldRadioButtons();
}
private function bldTxtArea() {
aTa = new TextArea();
aTa.setSize(230, 44);
aTa.text = "Hello World!";
aTa.move(165, 57);
addChild(aTa);
}
private function bldColorPicker() {
txtCp = new ColorPicker();
txtCp.move(96, 72);
addChild(txtCp);
}
private function bldComboBox() {
msgCb = new ComboBox();
msgCb.width = 130;
}

```

```

msgCb.move(265, 120);
msgCb.prompt = "Greetings";
msgCb.addItem({data:"Hello.", label:"English"});
msgCb.addItem({data:"Bonjour.", label:"Français"});
msgCb.addItem({data:"¡Hola!", label:"Español"});
addChild(msgCb);
}
private function bldRadioButtons() {
    rbGrp = new RadioButtonGroup("fontRbGrp");
    smallRb = new RadioButton();
    smallRb.setSize(100, 22);
    smallRb.move(155, 120);
    smallRb.group = rbGrp; //"fontRbGrp";
    smallRb.label = "Small";
    smallRb.name = "smallRb";
    addChild(smallRb);
    largerRb = new RadioButton();
    largerRb.setSize(100, 22);
    largerRb.move(155, 148);
    largerRb.group = rbGrp;
    largerRb.label = "Larger";
    largerRb.name = "largerRb";
    addChild(largerRb);
    largestRb = new RadioButton();
    largestRb.setSize(100, 22);
    largestRb.move(155, 175);
    largestRb.group = rbGrp;
    largestRb.label = "Largest";
    largestRb.name = "largestRb";
    addChild(largestRb);
}

```

这些行执行下列操作：

- 实例化应用程序中使用的组件。
- 设置每个组件的大小、位置和属性。
- 使用 `addChild()` 方法将各个组件添加到舞台上。

13. 在 `bldRadioButtons()` 方法的右括号后，添加 `setUpHandlers()` 方法的下列代码：

```

private function setUpHandlers():void {
    rbGrp.addEventListener(MouseEvent.CLICK, rbHandler);
    txtCp.addEventListener(ColorPickerEvent.CHANGE, cpHandler);
    msgCb.addEventListener(Event.CHANGE, cbHandler);
}
private function rbHandler(event:MouseEvent):void {
    switch(event.target.selection.name) {
        case "smallRb":
            tf.size = 14;
            break;
        case "largerRb":
            tf.size = 18;
    }
}

```

```

        break;
    case "largestRb":
        tf.size = 24;
        break;
    }
    aTa.setStyle("textFormat", tf);
}
private function cpHandler(event:ColorPickerEvent):void {
    tf.color = event.target.selectedColor;
    aTa.setStyle("textFormat", tf);
}
private function cbHandler(event:Event):void {
    aTa.text = event.target.selectedItem.data;
}
}
}

```

这些函数定义组件的事件侦听器。

14. 选择“文件” > “保存”以保存文件。
15. 选择“控制” > “测试影片”或按 **Ctrl+Enter** 编译代码，然后测试 **Greetings2** 应用程序。

运行后续示例

在开发及运行 **Greetings** 应用程序后，您应当已具备运行本书中介绍的其它代码示例所需的基本知识。每个示例中相关的 **ActionScript 3.0** 代码都会高亮显示并加以讨论，而您应当能够将本书中的每个示例剪切并粘贴到 **FLA** 文件中，进行编译，然后加以运行。

使用组件

在本章中，您将学习如何在文档中使用组件。本章介绍以下主题：

组件体系结构	31
使用组件文件	34
调试组件应用程序	37
设置参数和属性	38
库	40
调整组件大小	42
实时预览	43
处理事件	43
使用显示列表	45
使用 FocusManager	48
使用基于 List 的组件	49
使用 DataProvider	50
使用 CellRenderer	59
使组件具有辅助功能	66

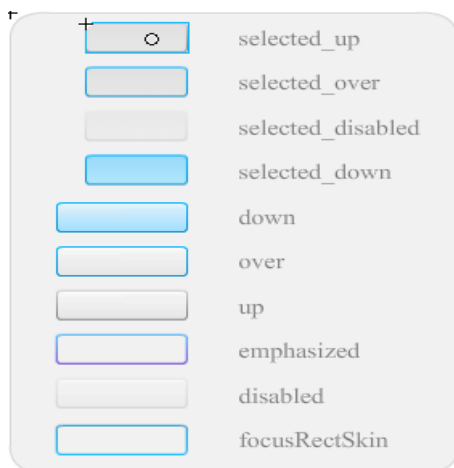
组件体系结构

Adobe Flash Player 9.0.28.0 版及更高版本支持 ActionScript 3.0 组件。这些组件与在 Flash CS3 之前构建的组件不兼容。有关使用 ActionScript 2.0 组件的信息，请参阅《使用 ActionScript 2.0 组件》和《ActionScript 2.0 组件语言参考》。

ActionScript 3.0 用户界面 (UI) 组件是作为基于 FLA 的组件实现的，但 Flash CS3 同时支持基于 SWC 和 FLA 的组件。例如，FLVPlayback 和 FLVPlaybackCaptioning 组件是基于 SWC 的组件。您可以将其中一种类型的组件置于 Components 文件夹中以使其显示在“组件”面板中。这两种类型的组件的构建方式不同，因此在这里将它们分开介绍。

ActionScript 3.0 基于 FLA 的组件

ActionScript 3.0 用户界面组件是具有内置外观的基于 FLA (.fla) 的文件，您可以通过在舞台上双击组件访问此类文件以对其进行编辑。这种组件的外观及其它资源位于时间轴的第 2 帧上。双击这种组件时，Flash 将自动跳到第 2 帧并打开该组件外观的调色板。下图显示了针对 Button 组件显示的外观调色板。



Button 组件的外观

有关组件外观和自定义组件的详细信息，请参阅第 129 页的第 4 章“自定义 UI 组件”和第 193 页的“自定义 FLVPlayback 组件”。

为了加快应用程序的编译速度并避免与 ActionScript 3.0 设置冲突，Flash CS3 基于 FLA 的 UI 组件还包含一个含有该组件的已编译 ActionScript 代码的 SWC。ComponentShim SWC 放置在舞台上每个用户界面组件的第 2 帧上，以便使预编译的定义可用。若要使某一组件可用于 ActionScript，该组件必须位于舞台上或位于库中，并且在其“链接属性”中已选中“在第一帧导出”选项。若要使用 ActionScript 创建组件，还必须使用 import 语句导入类以访问该类。有关 import 语句的信息，请参阅《ActionScript 3.0 语言和组件参考》。

基于 SWC 的组件

基于 SWC 的组件也有一个 FLA 文件和一个 ActionScript 类文件，但它们已编译并导出为 SWC。SWC 文件是一个由预编译的 Flash 元件和 ActionScript 代码组成的包，使用它可避免重新编译不会更改的元件和代码。

FLVPlayback 和 FLVPlaybackCaptioning 组件是基于 SWC 的组件。它们具有外部外观，而不是内置外观。FLVPlayback 组件具有默认外观，您可以通过以下方式更改其默认外观：从预设计外观的集合中选择一种外观，基于“组件”面板中的 UI 控件（BackButton、BufferingBar 等）自定义控件或者创建自定义外观。有关详细信息，请参阅第 193 页的“自定义 FLVPlayback 组件”。

在 Flash 中，您可以按如下方式将影片剪辑转换为编译剪辑。

编译影片剪辑：

- 右击 (Windows) 或按住 Control 单击 (Macintosh) “库”面板中的影片剪辑，然后选择“转换为编译剪辑”。

编译剪辑的行为方式与从中编译它的影片剪辑相似，但与普通影片剪辑相比，编译剪辑的显示速度和发布速度要快得多。编译剪辑无法进行编辑，但其属性可以显示在“属性”检查器和“组件”检查器中。

SWC 组件包含编译剪辑，此组件的预编译 ActionScript 定义以及描述此组件的其它文件。如果您创建自己的组件，则可以将其导出为 SWC 文件以便分发。

导出 SWC 文件：

- 选择“库”面板中的影片剪辑并右击 (Windows) 或按住 Control 单击 (Macintosh)，然后选择“导出 SWC 文件”。



Flash CS3 SWC 文件的格式与 Flex SWC 格式兼容，因此可以在这两个产品之间交换 SWC 文件，但交换后文件可能会发生修改。

有关创建基于 SWC 的组件的信息，请参阅

www.adobe.com/go/learn_fl_creating_components_cn。

ActionScript 3.0 组件 API

每个 ActionScript 3.0 组件都是基于一个 ActionScript 3.0 类构建的，该类位于一个包文件夹中，其名称格式为 `fl.packageName.className`。例如，Button 组件是 Button 类的实例，其包名称为 `fl.controls.Button`。将组件类导入应用程序中时，必须引用包名称。可以用以下语句导入 Button 类：

```
import fl.controls.Button;
```

有关组件类文件的位置的详细信息，请参阅第 34 页的“使用组件文件”。

组件的类定义了一些方法、属性、事件和样式，使用它们可以在应用程序中与该组件进行交互。ActionScript 3.0 UI 组件是 Sprite 和 UIComponent 类的子类，继承了它们的属性、方法和事件。Sprite 类是基本的显示列表构造块，与 MovieClip 类似，但不具有时间轴。UIComponent 类是所有可视组件（交互式和非交互式）的基类。每个组件的继承路径及其属性、方法、事件和样式都在《ActionScript 3.0 语言和组件参考》中进行了介绍。

所有 ActionScript 3.0 组件都使用 ActionScript 3.0 事件处理模型。有关事件处理的详细信息，请参阅第 43 页的“处理事件”和《ActionScript 3.0 编程》。

使用组件文件

本节说明组件文件的存储位置、在何处查找 ActionScript 源文件以及如何在“组件”面板中添加和删除组件。

组件文件的存储位置

Flash 组件存储在应用程序级别的 Configuration 文件夹中。



有关这些文件夹的信息，请参阅《使用 Flash》中的“随 Flash 安装的配置文件夹”。

组件安装在以下位置：

- Windows 2000 或 Windows XP:
C:\Program Files\Adobe\Adobe Flash CS3\language\Configuration\Components
- Mac OS X:
Macintosh HD:Applications:Adobe Flash CS3:Configuration:Components

在 Components 文件夹中，用户界面 (UI) 组件位于 User Interface.fla 文件中，FLVPlayback (FLVPlaybackAS3.swc) 和 FLVPlaybackCaptioning 组件位于 Video 文件夹中。

您还可以将组件存储在以下基于用户的位置：

- Windows 2000 或 Windows XP:
C:\Documents and Settings*username*\Local Settings\Application Data\Adobe\Adobe Flash CS3\en\Configuration\Components
- Mac OS X:
Macintosh HD:Users:<username>:Library:Application Support:Adobe Flash CS3:Configuration:Components

组件源文件的存储位置

对于 Windows 2000 或 Windows XP，组件的 ActionScript (.as) 类文件（即“源文件”）安装在以下应用程序文件夹中：

- 用户界面组件
C:\Program Files\Adobe\Adobe Flash CS3\en\Configuration\Component Source\ActionScript 3.0\User Interface\fl
- FLVPlayback
C:\Program Files\Adobe\Adobe Flash CS3\en\Configuration\Component Source\ActionScript 3.0\FLVPlayback\fl\video
- FLVPlaybackCaptioning
C:\Program Files\Adobe\Adobe Flash CS3\en\Configuration\Component Source\ActionScript 3.0\FLVPlaybackCaptioning\fl\video

对于 Mac OS X，组件源文件位于以下位置：

- 用户界面组件
Macintosh HD:Applications:Adobe Flash CS3:Configuration:Component Source>ActionScript 3.0>User Interface:fl
- FLVPlayback
Macintosh HD:Applications:Adobe Flash CS3:Configuration:Component Source>ActionScript 3.0:FLVPlayback:fl:video
- FLVPlaybackCaptioning
Macintosh HD:Applications:Adobe Flash CS3:Configuration:Component Source>ActionScript 3.0:FLVPlaybackCaptioning:fl:video

组件源文件和类路径

因为 **ActionScript 3.0** 组件的代码是在内部编译的，所以不应在类路径变量中指定 **ActionScript** 类文件的位置。如果您确实在类路径中包含了它们的位置，则会增加编译应用程序所需的时间。但是，如果 **Flash** 在类路径设置中找到组件类文件，该类文件将始终优先于组件的内部编译代码。

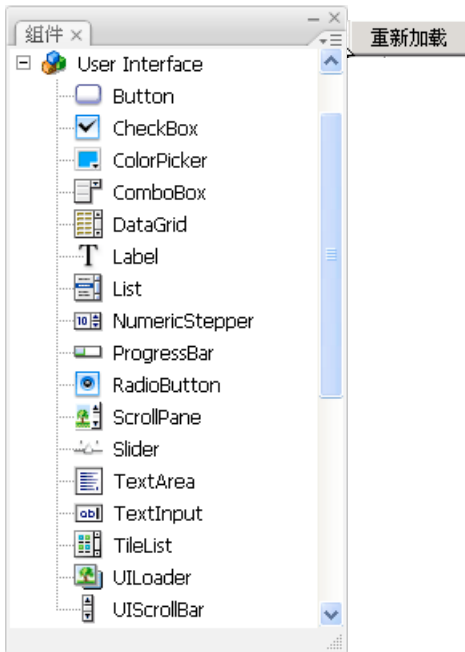
在调试包含组件的应用程序时，您可能需要将组件源文件的位置添加到类路径设置。有关详细信息，请参阅第 37 页的“[调试组件应用程序](#)”。

修改组件文件

如果要更新、添加或删除基于 **SWC** 的组件，或者将新的基于 **FLA** 的组件添加到 **Flash** 中，则必须将它们重新加载到“组件”面板以使其可用。您可以通过重新启动 **Flash** 或从“组件”面板菜单中选择“重新加载”来重新加载组件。这将导致 **Flash** 拾取已添加到 **Components** 文件夹的任何组件。

在 **Flash** 运行时将组件重新加载到“组件”面板中：

- 从“组件”面板菜单中选择“重新加载”。



“组件”面板菜单中的“重新加载”菜单项

从“组件”面板中删除组件:

- 从 Components 文件夹中删除 FLA、SWC 或 MXP 文件，并重新启动 Flash 或从“组件”面板菜单中选择“重新加载”。MXP 文件是已从 Adobe Exchange 下载的组件文件。

您可以在 Flash 运行时删除和替换基于 SWC 的组件，重新加载后将反映这些更改；但是如果您更改或删除基于 FLA 的组件，则这些更改直到您终止并重新启动 Flash 后才能反映出来。不过，您可以添加基于 FLA 的组件并用“重新加载”命令加载它们。

提示

Adobe 建议您首先对要更改的任何 Flash 组件文件（.fla 或 .as）做一个备份。然后就可以在必要时进行还原。

调试组件应用程序

ActionScript 3.0 组件包含它们的所有源代码，从而减少了应用程序的编译时间。但是，Flash 调试器无法检查编译剪辑内的代码。因此，如果您要将应用程序的调试深入到组件的源代码，必须将组件源文件添加到类路径设置。

组件包文件夹的位置是相对于组件类型的源文件位置而言的。若要引用所有 UI 组件的所有 ActionScript 3.0 源文件，请将以下位置添加到用户界面包的类路径：

```
$(AppConfig)/Component Source/ActionScript 3.0/User Interface
```

提醒

这将覆盖所有 UI 组件的内部编译代码并增加应用程序的编译时间。如果您因任何原因更改了组件的源文件，则该组件可能会因此而表现出不同的行为。

若要设置类路径，请从“编辑”菜单中选择“首选参数”，然后从“类别”列表中选择“ActionScript”并单击“ActionScript 3.0 设置”按钮。若要添加新条目，请单击显示当前设置的窗口上方的加号。

\$(AppConfig) 变量引用位于 Flash CS3 安装位置的 Flash CS3 Configuration 文件夹。通常，该路径如下：

对于 Windows 2000 或 Windows XP

```
C:\Program Files\Adobe\Adobe Flash CS3\language\Configuration\
```

对于 Mac OS X

```
Macintosh HD:Applications:Adobe Flash CS3:Configuration:
```

提醒

如果必须更改组件源文件，Adobe 强烈建议您将原始源文件复制到其它位置，并将该位置添加到类路径。

有关组件源文件位置的详细信息，请参阅第 35 页的“组件源文件的存储位置”。

设置参数和属性

每个组件都带有参数，通过设置这些参数可以更改组件的外观和行为。参数是组件的类的属性，显示在“属性”检查器和“组件”检查器中。最常用的属性显示为创作参数；其它参数必须使用 **ActionScript** 来设置。可以在创作时设置的所有参数都可以使用 **ActionScript** 来设置。使用 **ActionScript** 设置参数将覆盖在创作时设置的任何值。

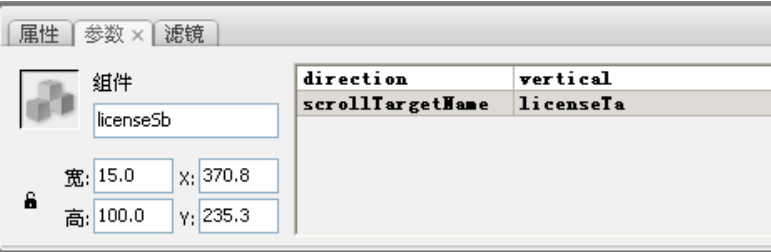
大多数 **ActionScript 3.0** 用户界面组件都从 **UIComponent** 类和基类继承属性和方法。例如，**Button** 和 **CheckBox** 类从 **UIComponent** 类和 **BaseButton** 类继承属性。您既可以访问组件继承的属性，也可以访问它自己的类属性。例如，**ProgressBar** 组件从 **UIComponent** 继承了 **ProgressBar.enabled** 属性，但它自己也有 **ProgressBar.percentComplete** 属性。您可以同时访问两个属性来与 **ProgressBar** 组件的实例进行交互。有关组件的属性的详细信息，请参阅它在 [《ActionScript 3.0 语言和组件参考》](#) 中的相应类条目。

您可以使用“属性”检查器或“组件”检查器设置组件实例的参数。

若要在“属性”检查器中输入组件的实例名称：

1. 选择“窗口” > “属性” > “属性”。
2. 在舞台上选择组件的一个实例。
3. 在“影片剪辑”下拉列表下方的“< 实例名称 >”框中输入组件实例的名称。或者，单击“参数”选项卡并在“组件”字样下的框中输入名称。为您要设置的任何参数输入值。

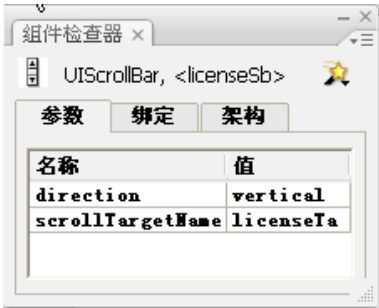
最好向实例名称添加用于指示组件类型的后缀：这可使 **ActionScript** 代码更加易读。在此示例中，实例名称为“**licenseSb**”，因为组件是用于在“**licenseTa**”文本区域中滚动许可协议的滚动条。



组件实例名称字段

在“组件”检查器中输入组件实例的参数：

1. 选择“窗口”>“组件检查器”。
2. 在舞台上选择组件的一个实例。
3. 单击“参数”选项卡并为列出的任意参数输入值。



“组件”检查器中的组件参数

在 ActionScript 中设置组件属性

在 ActionScript 中，可以使用点 (.) 运算符（点语法）访问属于舞台上的对象或实例的属性或方法。点语法表达式以实例的名称开头，后面跟着一个点，最后以要指定的元素结尾。例如，以下 ActionScript 代码设置 CheckBox 实例 aCh 的 width 属性，使其宽度为 50 像素：

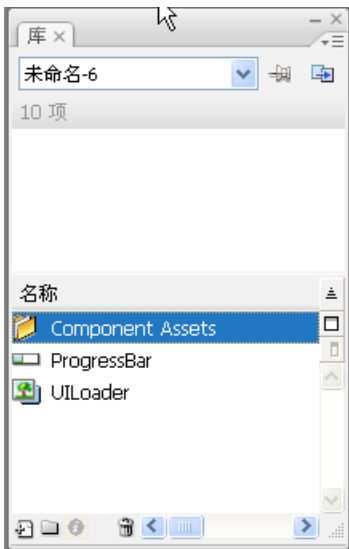
```
aCh.width = 50;
```

下面的 if 语句检查用户是否已选中该复选框：

```
if (aCh.selected == true) {  
    displayImg(redCar);  
}
```

库

首次将组件添加到文档时，Flash 会将其作为影片剪辑导入到“库”面板中。还可以将组件从“组件”面板直接拖到“库”面板中，然后将其实例添加到舞台上。在任何情况下，您都必须将组件添加到库中，才能访问其类元素。

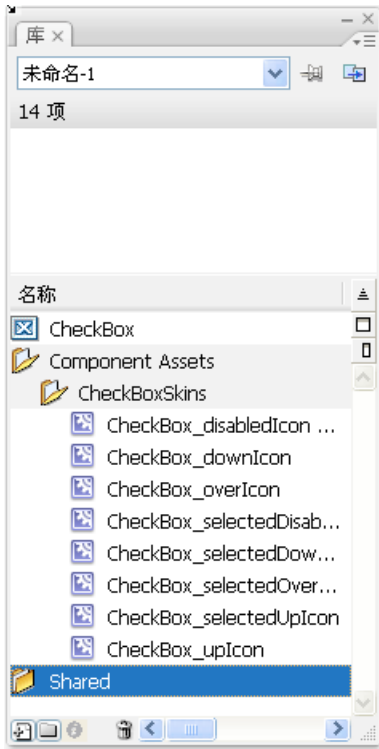


“库”面板中的 *ProgressBar* 组件

如果将组件添加到库中并使用 **ActionScript** 创建其实例，则必须首先使用 `import` 语句导入该组件的类。在 `import` 语句中，必须同时指定组件的包名称和组件的类名称。例如，下面的语句导入 **Button** 类：

```
import fl.controls.Button;
```


将组件置于库中时，Flash 还将导入其资源的文件夹，该文件夹包含该组件在不同状态下的外观。组件的“外观”由一个元件集合构成，其中的元件在应用程序中构成了该组件的图形显示。单个外观是指示组件特定状态的图形表示形式（或影片剪辑）。例如，在 **CheckBox** 的 **Component Assets** 文件夹中，**CheckBox_disabledIcon** 外观提供了组件处于禁用状态时的图形表示形式。**CheckBox_selectedDownIcon** 外观提供了在单击 **CheckBox** 并按住鼠标按键时显示的图像。



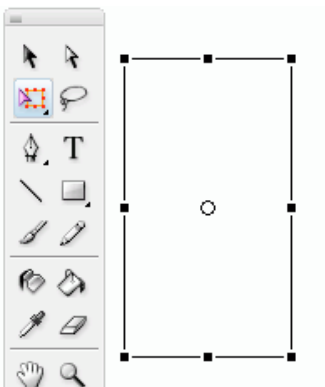
“库”面板中的组件资源

如果愿意，可以使用 **Component Assets** 文件夹中的内容更改组件的外观。有关详细信息，请参阅第 129 页的第 4 章“自定义 UI 组件”。

组件入库后，就可以将它的更多实例添加到文档中，方法是将其图标从“组件”面板或“库”面板中拖到舞台上。

调整组件大小

可以使用“任意变形”工具或 `setSize()` 方法来调整组件实例的大小。

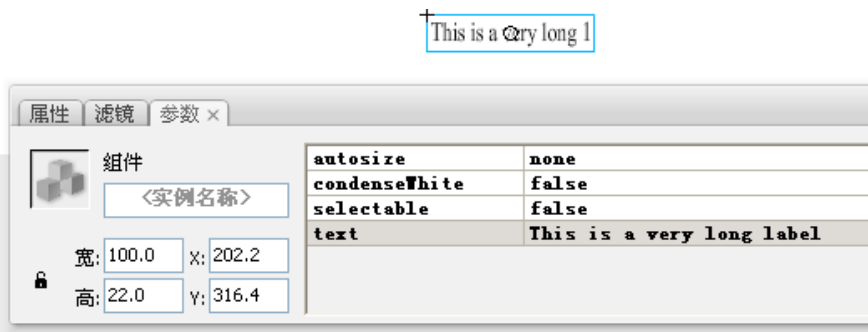


使用“任意变形”工具调整舞台上的 *List* 组件的大小

可以从任意组件实例中调用 `setSize()` 方法（请参阅 `UIComponent.setSize()`）来调整其大小。下面的代码将一个 *List* 组件实例的大小调整为宽 200 像素、高 300 像素：

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

组件不会自动调整大小以适合其标签。如果添加到文档中的组件实例不够大，而无法显示其标签，就会将标签文本剪切掉。您必须调整组件大小以适合其标签。



Label 组件中被裁剪的文本

有关调整组件大小的详细信息，请参阅 [《ActionScript 3.0 语言和组件参考》](#) 中各个组件条目。

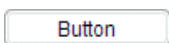
实时预览

使用默认启用的“实时预览”功能，可以在舞台上查看组件将在发布的 Flash 内容中出现的近似大小和外观。

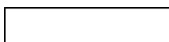
打开或关闭“实时预览”：

- 选择“控制”>“启用实时预览”。如果该选项旁边出现一个复选标记，表明已经启用了这项功能。

实时预览反映了不同组件的不同参数。有关实时预览中反映的组件参数的信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的各个组件条目。



启用“实时预览”的 *Button* 组件



禁用“实时预览”的 *Button* 组件

“实时预览”中的组件不可操作。若要测试功能，必须使用“控制”>“测试影片”命令。

处理事件

每一个组件在用户与它交互时都会广播事件。例如，当用户单击一个 **Button** 时，它会调度 `MouseEvent.CLICK` 事件；当用户选择 **List** 中的一个项目时，**List** 会调度 `Event.CHANGE` 事件。当组件发生重要事情时也会引发事件，例如，当 **UILoader** 实例完成内容加载时，会生成一个 `Event.COMPLETE` 事件。若要处理事件，您需要编写在该事件被触发时需要执行的 **ActionScript** 代码。

组件的事件包括该组件继承的所有类的事件。这意味着，所有 **ActionScript 3.0** 用户界面组件都从 **UIComponent** 类继承事件，因为它是 **ActionScript 3.0** 用户界面组件的基类。若要查看某一组件广播的事件列表，请参阅《[ActionScript 3.0 语言和组件参考](#)》中该组件的类条目中的“事件”部分。

有关 **ActionScript 3.0** 中事件处理的完整说明，请参阅《[ActionScript 3.0 编程](#)》。

关于事件侦听器

以下要点适用于 **ActionScript 3.0** 组件的事件处理：

- 所有事件均由组件类的实例广播。组件实例为“广播器”。
- 通过调用组件实例的 `addEventListener()` 方法，可以注册事件的“侦听器”。例如，下面这行代码向 **Button** 实例 `aButton` 添加了一个 `MouseEvent.CLICK` 事件的侦听器：
`aButton.addEventListener(MouseEvent.CLICK, clickHandler);`
`addEventListener()` 方法的第二个参数注册在该事件发生时要调用的函数的名称，即 `clickHandler`。此函数也称作“回调函数”。
- 您可以向一个组件实例注册多个侦听器。
`aButton.addEventListener(MouseEvent.CLICK, clickHandler1);`
`aButton.addEventListener(MouseEvent.CLICK, clickHandler2);`
- 也可以向多个组件实例注册一个侦听器。
`aButton.addEventListener(MouseEvent.CLICK, clickHandler1);`
`bButton.addEventListener(MouseEvent.CLICK, clickHandler1);`
- 会将一个事件对象传递给该事件处理函数，该对象包含有关该事件类型和广播该事件的实例的信息。有关详细信息，请参阅第 44 页的“关于事件对象”。
- 在应用程序终止或您使用 `removeEventListener()` 显式删除侦听器之前，侦听器会一直保持活动状态。例如，下面这行代码删除 `aButton` 上 `MouseEvent.CLICK` 事件的侦听器：
`aButton.removeEventListener(MouseEvent.CLICK, clickHandler);`

关于事件对象

事件对象继承自 **Event** 对象类，它的一些属性包含了有关所发生事件的信息，其中包括提供事件基本信息的 `target` 和 `type` 属性：

属性	说明
<code>type</code>	表示事件类型的字符串。
<code>target</code>	对广播事件的组件实例的引用。

如果事件具有其它属性，则会在《**ActionScript 3.0 语言和组件参考**》中该事件的类描述中列出这些属性。

事件对象是自动生成的，当事件发生时会将它传递给事件处理函数。

您可以在该函数内使用事件对象来访问所广播的事件的名称，或者访问广播该事件的组件的实例名称。通过该实例名称，可以访问其它组件属性。例如，下面的代码使用 `evtObj` 事件对象的 `target` 属性来访问 `aButton` 的 `label` 属性并将它显示在“输出”面板中：

```
import fl.controls.Button;
import flash.events.MouseEvent;

var aButton:Button = new Button();
aButton.label = "Submit";
addChild(aButton);
aButton.addEventListener(MouseEvent.CLICK, clickHandler);

function clickHandler(evtObj:MouseEvent){
    trace("The " + evtObj.target.label + " button was clicked");
}
```

使用显示列表

所有 **ActionScript 3.0** 组件均继承自 **DisplayObject** 类，因此，它们均有权访问该类的方法和属性以与显示列表交互。“显示列表”是应用程序中所显示对象和可视元素的层次结构。此层次结构包含以下元素：

- 舞台，它是顶层容器
- 显示对象，包括形状、影片剪辑、文本字段以及其它
- 显示对象容器，即可以包含子显示对象的特殊类型的显示对象。

显示列表中对象的顺序决定了这些对象在父容器中的深度。对象的深度是指它在舞台上或在其显示容器中从上到下或从前到后来看的位置。对象重叠时，深度的顺序很明显；但即使对象不重叠，深度顺序也存在。显示列表中的每个对象在舞台上都有对应的深度。如果要将对象放置在其它对象的前面或移动到其它对象的后面来更改该对象的深度，则需要更改该对象在显示列表中的位置。对象在显示列表中的默认顺序是将它们放置在舞台上的顺序。位于显示列表中位置 **0** 处的对象是在深度顺序中处于最底部的对象。

向显示列表添加组件

通过调用 `DisplayObjectContainer` 容器的 `addChild()` 或 `addChildAt()` 方法，可以向该容器添加对象。对于舞台，在创作过程中还可以通过创建对象来向其显示列表添加对象；对于组件，则可以通过将组件从“组件”面板中拖到舞台上来自向其显示列表添加对象。若要使用 `ActionScript` 向容器添加对象，首先要通过使用 `new` 运算符调用对象的构造函数来创建该对象的一个实例，然后再调用 `addChild()` 或 `addChildAt()` 方法将它放置到舞台上或显示列表中。`addChild()` 方法将该对象放置到显示列表中的下一位置，而 `addChildAt()` 则指定该对象将要添加到的位置。如果您指定的位置已经被占用，则位于该位置以及该位置之上的对象均会向上移动 1 个位置。`DisplayObjectContainer` 对象的 `numChildren` 属性指定了它包含的显示对象的数目。可以通过调用 `getChildAt()` 方法并指定位置来检索显示列表中的对象，如果您知道对象的名称，也可以通过调用 `getChildByName()` 方法来检索对象。



使用 `ActionScript` 添加组件时，如果要通过名称在显示列表中访问该组件，您必须为该组件的 `name` 属性分配一个名称。

下面的示例列出了显示列表中三个组件的名称和位置。首先，将一个 `NumericStepper`、一个 `Button` 和一个 `ComboBox` 拖到舞台上，使它们相互重叠，并为它们分别指定实例名称“`aNs`”、“`aButton`”和“`aCb`”。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
var i:int = 0;
while(i < numChildren) {
    trace(getChildAt(i).name + " is at position: " + i++);
}
```

在“输出”面板中应看到以下行：

```
aNs is at position: 0
aButton is at position: 1
aCb is at position: 2
```

移动显示列表中的组件

通过调用 `addChildAt()` 方法并提供对象名称和您要将该对象放置到的位置作为方法的参数，可以更改该对象在显示列表中的位置和显示深度。例如，将下面的代码添加到上一示例中可将 `NumericStepper` 放在顶部，重复执行此循环可显示这些组件在显示列表中的新位置：

```
this.addChildAt(aNs, numChildren - 1);
i = 0;
while(i < numChildren) {
    trace(getChildAt(i).name + " is at position: " + i++);
}
```

在“输出”面板中应看到以下内容：

```
aNs is at position: 0  
aButton is at position: 1  
aCb is at position: 2  
aButton is at position: 0  
aCb is at position: 1  
aNs is at position: 2
```

NumericStepper 还应显示在屏幕中其它组件的前面。

请注意，`numChildren` 是显示列表中对象的数目（从 1 到 “n”），而该列表中的第一个位置为 0。因此，如果该列表中有三个对象，则第三个对象的索引位置为 2。这意味着您可以使用 `numChildren - 1` 引用显示列表中的最后一个位置（对于显示深度而言，则是引用顶层对象）。

从显示列表中删除组件

可以使用 `removeChild()` 和 `removeChildAt()` 方法将组件从显示对象容器及其显示列表中删除。下面的示例将三个 **Button** 组件在舞台上依次放到另一个的前面，并为每个组件都添加一个事件侦听器。单击每个 **Button** 时，事件处理函数会将其从显示列表和舞台上删除。

从显示列表中删除组件：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将一个 **Button** 从“组件”面板拖到“库”面板上。
3. 打开“动作”面板，在主时间轴中选择第 1 帧，然后添加以下代码：

```
import fl.controls.Button;  
  
var i:int = 0;  
while(i++ < 3) {  
    makeButton(i);  
}  
function removeButton(event:MouseEvent):void {  
    removeChildAt(numChildren - 1);  
}  
function makeButton(num) {  
    var aButton:Button = new Button();  
    aButton.name = "Button" + num;  
    aButton.label = aButton.name;  
    aButton.move(200, 200);  
    addChild(aButton);  
    aButton.addEventListener(MouseEvent.CLICK, removeButton);  
}
```

有关显示列表的完整说明，请参阅《ActionScript 3.0 编程》中的第 12 章“显示编程”。

使用 FocusManager

当用户按 **Tab** 键在 **Flash** 应用程序中导航时或在应用程序中单击时，**FocusManager** 类会确定接收输入焦点的组件。除非您正在创建组件，否则您无需向应用程序添加 **FocusManager** 实例，也不必编写任何代码来激活 **FocusManager**。

如果 **RadioButton** 对象接收焦点，**FocusManager** 将检查该对象和具有相同 `groupName` 值的所有对象，然后将焦点设置在 `selected` 属性设置为 `true` 的对象上。

每个模式窗口组件都包含 **FocusManager** 的一个实例，因此，该窗口上的控件也就成为它们自己的 **Tab** 集。这样可以防止用户按 **Tab** 切换到其它窗口中的组件。

FocusManager 使用容器中元素的深度级别（“**z**”顺序）作为默认导航方案或“**Tab** 键循环”。用户通常使用 **Tab** 键来导航 **Tab** 键循环，焦点将从具有焦点的第一个组件移动到最后一个，然后再回到第一个。深度级别主要按组件拖到舞台上的顺序设置；但是，也可以使用“修改”>“排列”>“移至顶层”或“移至底层”命令来确定最终的“**z**”顺序。有关深度级别的详细信息，请参阅第 45 页的“使用显示列表”。

可以调用 `setFocus()` 方法来使应用程序中的某一组件实例获得焦点。例如，下面的示例为当前容器 (`this`) 创建一个 **FocusManager** 实例，并且使 **Button** 实例 `aButton` 获得焦点。

```
var fm:FocusManager = new FocusManager(this);
fm.setFocus(aButton);
```

可以通过调用 `getFocus()` 方法来确定哪个组件具有焦点；可以通过调用 `getNextFocusManagerComponent()` 方法来确定 **Tab** 键循环中哪个组件接下来将获得焦点。在下面的示例中，一个 **CheckBox**、一个 **RadioButton** 和一个 **Button** 在舞台上，每个组件都有 `MouseEvent.CLICK` 和 `FocusEvent.MOUSE_FOCUS_CHANGE` 事件的侦听器。`MouseEvent.CLICK` 事件发生时，因为用户单击了组件，所以 `showFocus()` 函数调用 `getNextFocusManagerComponent()` 方法，以确定 **Tab** 键循环中哪个组件接下来将获得焦点。然后，它调用 `setFocus()` 方法来使该组件获得焦点。

`FocusEvent.MOUSE_FOCUS_CHANGE` 事件发生时，`fc()` 函数显示发生了此事件的组件的名称。当用户单击不同于 **Tab** 键循环中下一个组件的组件时，触发此事件。

```
// This example assumes a CheckBox (aCh), a RadioButton (aRb) and a Button
// (aButton) have been placed on the Stage.
```

```
import fl.managers.FocusManager;
import flash.display.InteractiveObject;
```

```
var fm:FocusManager = new FocusManager(this);
```

```
aCh.addEventListener(MouseEvent.CLICK, showFocus);
aRb.addEventListener(MouseEvent.CLICK, showFocus);
aButton.addEventListener(MouseEvent.CLICK, showFocus);
aCh.addEventListener(FocusEvent.MOUSE_FOCUS_CHANGE, fc);
aRb.addEventListener(FocusEvent.MOUSE_FOCUS_CHANGE, fc);
aButton.addEventListener(FocusEvent.MOUSE_FOCUS_CHANGE, fc);
```



```

function showFocus(event:MouseEvent):void {
    var nextComponent:InteractiveObject = fm.getNextFocusManagerComponent();
    trace("Next component in tab loop is: " + nextComponent.name);
    fm.setFocus(nextComponent);
}

function fc(fe:FocusEvent):void {
    trace("Focus Change: " + fe.target.name);
}

```

若要创建在用户按下 **Enter** (Windows) 或 **Return** (Macintosh) 时接收焦点的 **Button**，请将 `FocusManager.defaultButton` 属性设置为要作为默认 **Button** 的 **Button** 实例，如下面的代码所示：

```

import fl.managers.FocusManager;

var fm:FocusManager = new FocusManager(this);
fm.defaultButton = okButton;

```

FocusManager 类会覆盖默认的 **Flash Player** 焦点矩形，并绘制一个圆角的自定义焦点矩形。

有关在 **Flash** 应用程序中创建焦点方案的详细信息，请参阅 [《ActionScript 3.0 语言和组件参考》](#) 中的 [FocusManager 类](#)。若要创建自定义的焦点管理器，必须创建实现 *IFocusManager* 接口的类。有关详细信息，请参阅 [《ActionScript 3.0 语言和组件参考》](#) 中的 [IFocusManager](#)。

使用基于 List 的组件

List、**DataGrid** 和 **TileList** 组件均继承自 **SelectableList** 基类。因此，这些组件被视为基于 **List** 的组件。**ComboBox** 由一个文本框和一个 **List** 组成，因此它也是基于 **List** 的组件。

List 由若干行组成。**DataGrid** 和 **TileList** 由可分成多个列的若干行组成。行和列的相交部分即为单元格。对于由仅包含一列的若干行组成的 **List**，它的每一行都是一个单元格。单元格有以下两个重要方面：

- 单元格中保存的数据值称为项目。“项目”是用于将信息单元存储在 **List** 中的 **ActionScript** 对象。可以将 **List** 看作一个数组，此数组中的每个索引空间就是一个项目。在 **List** 中，项目是一个对象，该对象通常有一个显示出来的 `label` 属性和一个用于存储数据的 `data` 属性。“数据提供者”是 **List** 中项目的数据模型。只需将数据提供者赋给基于 **List** 的组件的 `dataProvider` 属性，即可填充该组件。

- 单元格可以保存不同类型的数据：从文本到图像、影片剪辑或您可以创建的任何类。出于此原因，单元格的绘制或渲染方式必须与其内容相适应。因此，基于 **List** 的组件有一个用于渲染其单元格的“单元格渲染器”。对于 **DataGrid**，每一列都是一个 **DataGridColumn** 对象，此对象也有一个 `cellRenderer` 属性，所以每一列都可以用适合其内容的方式进行渲染。

所有基于 **List** 的组件都有 `cellRenderer` 和 `dataProvider` 属性，您可以设置这些属性以加载和渲染这些组件的单元格。有关使用这些属性和使用基于 **List** 的组件的详细信息，请参阅第 50 页的“使用 **DataProvider**”和第 59 页的“使用 **CellRenderer**”。

使用 DataProvider

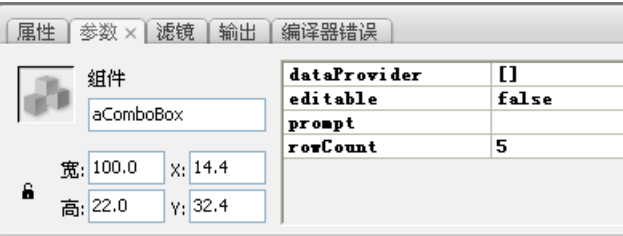
DataProvider 是可以用来向 **ComboBox**、**DataGrid**、**List** 和 **TileList** 组件提供数据的数据源。上述每个组件类都有一个 `dataProvider` 属性，您可以将 **DataProvider** 对象赋给该属性以用数据填充该组件的单元格。通常情况下，数据提供者是一个数据集，如 **Array** 或 **XML** 对象。

创建 DataProvider

对于 **ComboBox**、**List** 和 **TileList** 组件，您可以在创作环境中使用 `dataProvider` 参数来创建 **DataProvider**。**DataGrid** 组件在“属性”检查器中没有提供 `dataProvider` 参数，原因是它可能有多个列，所以它的数据提供者比较复杂。您还可以使用 **ActionScript** 为这些组件以及 **DataGrid** 创建 **DataProvider**。

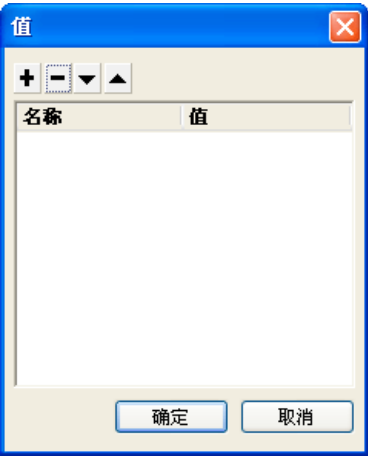
使用 dataProvider 参数

通过单击“属性”检查器或“组件”检查器的“参数”选项卡中的 `dataProvider` 参数，可以为 **ComboBox**、**List** 和 **TileList** 组件创建简单的数据提供者。下图显示了“属性”检查器中的此参数。



“属性”检查器中的 `dataProvider` 参数

如果双击值单元格（它最初显示一个空的 `Array`），将打开“值”对话框，可在此对话框中输入多个标签和数据值来创建数据提供者。



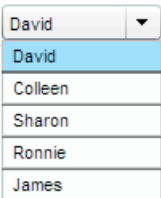
dataProvider 的“值”对话框

单击加号可向 `dataProvider` 添加项目。单击减号可删除项目。单击向上箭头可在列表中将所选项目上移，单击向下箭头可在列表中将所选项目下移。下图显示的“值”对话框创建一个由孩子的姓名及生日组成的列表。



带有数据的“值”对话框

您创建的 **Array** 由若干对标签和值字段组成。标签字段为 `label` 和 `data`，值字段为孩子的姓名及生日。标签字段标识 **List** 中显示的内容，在本例中即为孩子的姓名。产生的 **ComboBox** 如下所示：



*由 **DataProvider** 填充的 **ComboBox***

添加完数据后，单击“确定”以关闭该对话框。现在，已用您创建的项目填充了 `dataProvider` 参数中的 **Array**。

<code>allowMultipleSelection</code>	<code>false</code>
<code>dataProvider</code>	<code>[{label:David,data:11/19/1995},{label:Colleen,data:4/20/1993},{label:Sharon,data:9/6/1997},</code>
<code>horizontalLineScrollSize</code>	<code>1</code>
<code>horizontalPageScrollSize</code>	<code>0</code>
<code>horizontalScrollPolicy</code>	<code>auto</code>
<code>verticalLineScrollSize</code>	<code>1</code>

包含数据的 `dataProvider` 参数

通过使用 **ActionScript** 访问组件的 `dataProvider` 属性，可以访问您创建的标签和数据值。

使用 **ActionScript**

通过在 **Array** 或 **XML** 对象中创建数据并将该对象作为 `value` 参数提供给 **DataProvider** 构造函数，可以创建 **DataProvider**。

提醒

在 **ActionScript 3.0** 中，无法直接将 **Array** 或 **XML** 对象赋给 `dataProvider` 属性，原因是此属性定义为 **DataProvider** 对象，因而只能接收 **DataProvider** 类型的对象。

下面的示例用若干孩子的姓名及生日填充由仅包含一列的若干行组成的 **List** 组件。此示例在 `items` **Array** 中定义此列表，并在创建 **DataProvider** 实例 (`new DataProvider(items)`) 时将它作为参数提供，从而将它赋给此 **List** 组件的 `dataProvider` 属性。

```
import fl.controls.List;
import fl.data.DataProvider;

var aList:List = new List();
var items:Array = [
    {label:"David", data:"11/19/1995"},
    {label:"Colleen", data:"4/20/1993"},
    {label:"Sharon", data:"9/06/1997"},
];
```

```
{label:"Ronnie", data:"7/6/1993"},
{label:"James", data:"2/15/1994"},
];
aList.dataProvider = new DataProvider(items);
addChild(aList);
aList.move(150,150);
```

Array 由若干对标签和值字段组成。标签字段为 `label` 和 `data`，值字段为孩子的姓名及生日。标签字段标识 **List** 中显示的内容，在本例中即为孩子的姓名。产生的 **List** 如下所示：

David
Colleen
Sharon
Ronnie
James

由 *DataProvider* 填充的 *List*

当用户选择列表中的某一项目（通过单击该项目，此时会引发 `change` 事件）时可以使用数据字段的值。下面的示例向上一示例添加一个 **TextArea** (`aTa`) 和一个事件处理函数 (`changeHandler`)，以便在用户选择此 **List** 中的某一姓名时显示此孩子的生日。

```
import fl.controls.List;
import fl.controls.TextArea;
import flash.events.Event;
import fl.data.DataProvider;

var aList:List = new List();
var aTa:TextArea = new TextArea();
var items:Array = [
{label:"David", data:"1/19/1995"},
{label:"Colleen", data:"4/20/1993"},
{label:"Sharon", data:"9/06/1994"},
{label:"Ronnie", data:"7/6/1993"},
{label:"James", data:"2/15/1994"},
];
aList.dataProvider = new DataProvider(items);

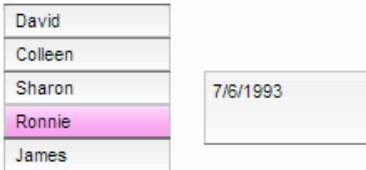
addChild(aList);
addChild(aTa);

aList.move(150,150);
aTa.move(150, 260);

aList.addEventListener(Event.CHANGE, changeHandler);

function changeHandler(event:Event):void {
    aTa.text = event.target.selectedItem.data;
};
```

现在，如果用户选择此 **List** 中某个孩子的姓名，此孩子的生日就会显示在此 **TextArea** 中，如下图所示。changeHandler() 函数将此 **TextArea** (aTa.text) 的 text 属性设置为所选项目中的数据字段 (event.target.selectedItem.data) 的值，从而实现此过程。event.target 属性是触发此事件的对象，在本例中即为此 **List**。



显示 **List** 的 **DataProvider** 中的数据字段

可以将非文本数据包含在 **DataProvider** 中。下面的示例将 **MovieClip** 包含在为 **TileList** 提供数据的 **DataProvider** 中。它在创建 **MovieClip**（一个带颜色的框）之后，通过调用 addItem() 添加每个项目来构建 **DataProvider**。

```
import fl.data.DataProvider;
import flash.display.DisplayObject;

var aBox:MovieClip = new MovieClip();
var i:uint = 0;
var colors:Array = new Array(0x000000, 0xFF0000, 0x0000CC, 0x00CC00,
    0xFFFF00);
var colorNames:Array = new Array("Midnight", "Cranberry", "Sky", "Forest",
    "July");
var dp:DataProvider = new DataProvider();
for(i=0; i < colors.length; i++) {
    drawBox(aBox, colors[i]); // draw box w next color in array
    dp.addItem( {label:colorNames[i], source:aBox} );
}
aTl.dataProvider = dp;
aTl.columnWidth = 110;
aTl.rowHeight = 130;
aTl.setSize(280,150);
aTl.move(150, 150);
aTl.setStyle("contentPadding", 5);

function drawBox(box:MovieClip,color:uint):void {
    box.graphics.beginFill(color, 1.0);
    box.graphics.drawRect(0, 0, 100, 100);
    box.graphics.endFill();
}
```

也可以使用 XML 数据（而非数组）来填充 **DataProvider** 对象。例如，下面的代码将数据存储在一个名为 `employeesXML` 的 XML 对象中，然后将该对象传递给 `DataProvider()` 构造函数的 **value** 参数：

```
import fl.controls.DataGrid;
import fl.data.DataProvider;

var aDg:DataGrid = new DataGrid();
addChild(aDg);

var employeesXML:XML =
    <employees>
        <employee Name="Edna" ID="22" />
        <employee Name="Stu" ID="23" />
    </employees>;

var myDP:DataProvider = new DataProvider(employeesXML);

aDg.columns = ["Name", "ID"];
aDg.dataProvider = myDP;
```

可以将数据作为 XML 数据的属性 (**attribute**) 提供，如上一段代码所示；也可以将数据作为 XML 数据的属性 (**property**) 提供，如下面的代码所示：

```
var employeesXML:XML =
    <employees>
        <employee>
            <Name>Edna</Name>
            <ID>22</ID>
        </employee>
        <employee>
            <Name>Stu</Name>
            <ID>23</ID>
        </employee>
    </employees>;
```

DataProvider 还有一组用来访问和操作它的方法和属性。可以使用 **DataProvider API** 在 **DataProvider** 中添加项目、删除项目、替换项目、对项目排序以及合并项目。

操作 DataProvider

使用 `addItem()` 和 `addItemAt()` 方法可以将项目添加到 **DataProvider** 中。下面的示例添加用户在可编辑的 **ComboBox** 的文本字段中输入的项目。在此示例中，假定已将一个 **ComboBox** 拖到舞台上并已为其指定实例名称 `aCb`。

```
import fl.data.DataProvider;
import fl.events.ComponentEvent;

var items:Array = [
    {label:"Roger"},
```

```

{label:"Carolyn"},
{label:"Darrell"},
{label:"Rebecca"},
{label:"Natalie"},
{label:"Mitchell"},
];
aCb.dataProvider = new DataProvider(items);

aCb.addEventListener(ComponentEvent.ENTER, newItemHandler);

function newItemHandler(event:ComponentEvent):void {
    var newRow:int = event.target.length + 1;
    event.target.addItemAt({label:event.target.selectedLabel},
        event.target.length);
}

```

还可以通过组件的 **DataProvider** 替换和删除组件中的项目。下面的示例实现了两个单独的 **List** 组件 listA 和 listB，并提供了一个标签为 “Sync” 的 **Button**。用户单击此 **Button** 时，此示例将使用 replaceItemAt() 方法用 listA 中的项目替换 listB 中的项目。如果 listA 比 listB 长，此示例将调用 addItem() 方法向 listB 添加额外的项目。如果 listB 比 listA 长，此示例将调用 removeItemAt() 方法删除 listB 中多余的项目。

// Requires the List and Button components to be in the library

```

import fl.controls.List;
import fl.controls.Button;
import flash.events.Event;
import fl.data.DataProvider;

var listA:List = new List();
var listB:List = new List();
var syncButton:Button = new Button();
syncButton.label = "Sync";

var itemsA:Array = [
{label:"David"},
{label:"Colleen"},
{label:"Sharon"},
{label:"Ronnie"},
{label:"James"},
];
var itemsB:Array = [
{label:"Roger"},
{label:"Carolyn"},
{label:"Darrell"},
{label:"Rebecca"},
{label:"Natalie"},
{label:"Mitchell"},
];
listA.dataProvider = new DataProvider(itemsA);

```



```

listB.dataProvider = new DataProvider(itemsB);

addChild(listA);
addChild(listB);
addChild(syncButton);

listA.move(100, 100);
listB.move(250, 100);
syncButton.move(175, 220);

syncButton.addEventListener(MouseEvent.CLICK, syncHandler);

function syncHandler(event:MouseEvent):void {
    var i:uint = 0;
    if(listA.length > listB.length) { //if listA is longer, add items to B
        while(i < listB.length) {
            listB.dataProvider.replaceItemAt(listA.dataProvider.getItemAt(i), i);
            ++i;
        }
        while(i < listA.length) {
            listB.dataProvider.addItem(listA.dataProvider.getItemAt(i++));
        }
    } else if(listA.length == listB.length) { //if listA and listB are equal
length
        while(i < listB.length) {
            listB.dataProvider.replaceItemAt(listA.dataProvider.getItemAt(i), i);
            ++i;
        }
    } else { //if listB is longer, remove extra items from B
        while(i < listA.length) {
            listB.dataProvider.replaceItemAt(listA.dataProvider.getItemAt(i), i);
            ++i;
        }
        while(i < listB.length) {
            listB.dataProvider.removeItemAt(i++);
        }
    }
}
}

```

还可以使用 `merge()`、`sort()` 和 `sortOn()` 方法合并 **DataProvider** 以及对其进行排序。下面的示例用两个垒球队的部分花名册填充两个 **DataGrid** 实例（`aDg` 和 `bDg`）。它添加一个标签为 **Merge** 的 **Button**，当用户单击此 **Button** 时，事件处理函数 (`mrgHandler`) 将合并 `bDg` 和 `aDg` 的花名册，并按最终 **DataGrid** 的 “Name” 列对其进行排序。

```

import fl.data.DataProvider;
import fl.controls.DataGrid;
import fl.controls.Button;

var aDg:DataGrid = new DataGrid();
var bDg:DataGrid = new DataGrid();

```

```

var mrgButton:Button = new Button();
addChild(aDg);
addChild(bDg);
addChild(mrgButton);
bldRosterGrid(aDg);
bldRosterGrid(bDg);
var aRoster:Array = new Array();
var bRoster:Array = new Array();
aRoster = [
    {Name:"Wilma Carter", Bats:"R", Throws:"R", Year:"So", Home: "Redlands,
    CA"},
    {Name:"Sue Pennypacker", Bats:"L", Throws:"R", Year:"Fr", Home: "Athens,
    GA"},
    {Name:"Jill Smithfield", Bats:"R", Throws:"L", Year:"Sr", Home:
    "Spokane, WA"},
    {Name:"Shirley Goth", Bats:"R", Throws:"R", Year:"Sr", Home: "Carson,
    NV"}
];
bRoster = [
    {Name:"Angelina Davis", Bats:"R", Throws:"R", Year:"So", Home: "Odessa,
    TX"},
    {Name:"Maria Santiago", Bats:"L", Throws:"L", Year:"Sr", Home: "Tacoma,
    WA"},
    {Name:"Debbie Ferguson", Bats:"R", Throws:"R", Year: "Jr", Home: "Bend,
    OR"}
];
aDg.dataProvider = new DataProvider(aRoster);
bDg.dataProvider = new DataProvider(bRoster);
aDg.move(50,50);
aDg.rowCount = aDg.length;
bDg.move(50,200);
bDg.rowCount = bDg.length;
mrgButton.label = "Merge";
mrgButton.move(200, 315);
mrgButton.addEventListener(MouseEvent.CLICK, mrgHandler);

function bldRosterGrid(dg:DataGrid){
    dg.setSize(400, 300);
    dg.columns = ["Name", "Bats", "Throws", "Year", "Home"];
    dg.columns[0].width = 120;
    dg.columns[1].width = 50;
    dg.columns[2].width = 50;
    dg.columns[3].width = 40;
    dg.columns[4].width = 120;
};

function mrgHandler(event:MouseEvent):void {
    aDg.dataProvider.merge(bDg.dataProvider);
    aDg.dataProvider.sortOn("Name");
}

```

有关详细信息，请参阅 [《ActionScript 3.0 语言和组件参考》](#) 中的 `DataProvider` 类。

使用 CellRenderer

CellRenderer 是一个类，基于 **List** 的组件（如 **List**、**DataGrid**、**TileList** 和 **ComboBox**）使用此类来操作和显示其各行的自定义单元格内容。自定义的单元格可以包含文本、预先构建的组件（如 **CheckBox**）或您可以创建的任何显示对象类。若要使用自定义 **CellRenderer** 来渲染数据，您可以扩展 **CellRenderer** 类或实现 **ICellRenderer** 接口来创建您自己的自定义 **CellRenderer** 类。

List、**DataGrid**、**TileList** 和 **ComboBox** 类为 **SelectableList** 类的子类。**SelectableList** 类包含 **cellRenderer** 样式。该样式用于定义组件用来渲染单元格的显示对象。

通过调用 **List** 对象的 **setRendererStyle()** 方法，可以调整 **CellRenderer** 使用的样式格式设置（请参阅下一节“[设置单元格格式](#)”）。您也可以定义一个自定义类，将其用作 **CellRenderer**（请参阅第 60 页的“[定义自定义 CellRenderer 类](#)”）。

设置单元格格式

CellRenderer 类包含许多用于控制单元格格式的样式。

使用以下样式可以定义用于单元格不同状态（禁用、按下、指针经过和弹起）的外观：

- **disabledSkin** 和 **selectedDisabledSkin**
- **downSkin** 和 **selectedDownSkin**
- **overSkin** 和 **selectedOverSkin**
- **upSkin** 和 **selectedUpSkin**

以下样式用于设置文本的格式：

- **disabledTextFormat**
- **textFormat**
- **textPadding**

可以通过调用 **List** 对象的 **setRendererStyle()** 方法或 **CellRenderer** 对象的 **setStyle()** 方法来设置这些样式。可以通过调用 **List** 对象的 **getRendererStyle()** 方法或 **CellRenderer** 对象的 **getStyle()** 方法来获取这些样式。还可以通过 **List** 对象的 **rendererStyles** 属性或 **CellRenderer** 对象的 **getStyleDefinition()** 方法访问定义所有渲染器样式（作为对象的命名属性）的对象。

可以调用 **clearRendererStyle()** 方法将样式重置为其默认值。

若要获取或设置列表中各行的高度值，请使用 **List** 对象的 **rowHeight** 属性。

定义自定义 CellRenderer 类

例如，下面的代码中包括两个类。`ListSample` 类用于实例化 `List` 组件，并且该类使用另一个类 `CustomRenderer` 来定义用于此 `List` 组件的单元格渲染器。`CustomRenderer` 类扩展了 `CellRenderer` 类。

使用扩展 `CellRenderer` 类的类来定义自定义 `CellRenderer`：

1. 选择“文件” > “新建”。
2. 在显示的“新建文档”对话框中，选择“Flash 文件 (ActionScript 3.0)”，然后单击“确定”。
3. 选择“窗口” > “组件”以显示“组件”面板。
4. 在“组件”面板中，将 `List` 组件拖到舞台上。
5. 如果 Flash 未显示“属性”检查器，请选择“窗口” > “属性” > “属性”。
6. 选择 `List` 组件后，在“属性”检查器中设置属性：
 - 实例名称：`myList`
 - W（宽度）：200
 - H（高度）：300
 - X: 20
 - Y: 20
7. 在时间轴中选择图层 1 的第 1 帧，然后选择“窗口” > “动作”。
8. 在“动作”面板中键入以下脚本：

```
myList.setStyle("cellRenderer", CustomCellRenderer);
myList.addItem({label:"Burger -- $5.95"});
myList.addItem({label:"Fries -- $1.95"});
```
9. 选择“文件” > “保存”。为文件指定一个名称，然后单击“确定”按钮。
10. 选择“文件” > “新建”。
11. 在显示的“新建文档”对话框中，选择“ActionScript 文件”，然后单击“确定”按钮。
12. 在脚本窗口中，输入以下代码来定义 `CustomCellRenderer` 类：

```
package {
    import fl.controls.listClasses.CellRenderer;
    import flash.text.TextFormat;
    import flash.filters.BevelFilter;
    public class CustomCellRenderer2 extends CellRenderer {
        public function CustomCellRenderer2() {
            var format:TextFormat = new TextFormat("Verdana", 12);
            setStyle("textFormat", format);
            this.filters = [new BevelFilter()];
        }
    }
}
```

13. 选择“文件”>“保存”。将文件命名为 `CustomCellRenderer.as`，并将其放在 FLA 文件所在的目录中，然后单击“确定”按钮。

14. 选择“控制”>“测试影片”。

还可以使用继承 `DisplayObject` 类并实现 `ICellRenderer` 接口的任何类来定义

`CellRenderer`。例如，下面的代码定义了两个类。`ListSample2` 类将 `List` 对象添加到显示列表中，并将其 `CellRenderer` 定义为使用 `CustomRenderer` 类。`CustomRenderer` 类扩展了 `CheckBox` 类（`CheckBox` 类扩展了 `DisplayObject` 类）并实现了 `ICellRenderer` 接口。请注意，`CustomRenderer` 类为 `ICellRenderer` 接口中定义的 `data` 和 `listData` 属性定义了 `getter` 和 `setter` 方法。`ICellRenderer` 接口中定义的其它属性和方法（`selected` 属性和 `setSize()` 方法）已在 `CheckBox` 类中定义：

使用实现 `ICellRenderer` 接口的类来定义自定义 `CellRenderer`：

1. 选择“文件”>“新建”。
2. 在显示的“新建文档”对话框中，选择“Flash 文件 (ActionScript 3.0)”，然后单击“确定”。
3. 选择“窗口”>“组件”以显示“组件”面板。
4. 在“组件”面板中，将 `List` 组件拖到舞台上。
5. 如果 Flash 未显示“属性”检查器，请选择“窗口”>“属性”>“属性”。
6. 选择 `List` 组件后，在“属性”检查器中设置属性：
 - 实例名称：myList
 - W（宽度）：100
 - H（高度）：300
 - X：20
 - Y：20
7. 在时间轴中选择图层 1 的第 1 帧，然后选择“窗口”>“动作”。
8. 在“动作”面板中键入以下脚本：

```
myList.setStyle("cellRenderer", CustomCellRenderer);  
myList.addItem({name:"Burger", price:"$5.95"});  
myList.addItem({name:"Fries", price:"$1.95"});
```
9. 选择“文件”>“保存”。为文件指定一个名称，然后单击“确定”按钮。
10. 选择“文件”>“新建”。
11. 在显示的“新建文档”对话框中，选择“ActionScript 文件”，然后单击“确定”按钮。

12. 在脚本窗口中，输入以下代码来定义 `CustomCellRenderer` 类：

```
package
{
    import fl.controls.CheckBox;
    import fl.controls.listClasses.ICellRenderer;
    import fl.controls.listClasses.ListData;
    public class CustomCellRenderer extends CheckBox implements ICellRenderer
    {
        private var _listData:ListData;
        private var _data:Object;
        public function CustomCellRenderer() {
        }
        public function set data(d:Object):void {
            _data = d;
            label = d.label;
        }
        public function get data():Object {
            return _data;
        }
        public function set listData(ld:ListData):void {
            _listData = ld;
        }
        public function get listData():ListData {
            return _listData;
        }
    }
}
```

13. 选择“文件” > “保存”。将文件命名为 `CustomCellRenderer.as`，并将其放在 FLA 文件所在的目录中，然后单击“确定”按钮。

14. 选择“控制” > “测试影片”。

还可以使用库中的元件定义 `CellRenderer`。必须为 `ActionScript` 导出此元件，并且此库元件的类名称必须有一个实现 `ICellRenderer` 接口或扩展 `CellRenderer` 类（或其子类之一）的关联类文件。

下面的示例使用库元件定义自定义 `CellRenderer`。

使用库元件定义 `CellRenderer`：

1. 选择“文件” > “新建”。
2. 在显示的“新建文档”对话框中，选择“Flash 文件 (ActionScript 3.0)”，然后单击“确定”。
3. 选择“窗口” > “组件”以显示“组件”面板。
4. 在“组件”面板中，将 `List` 组件拖到舞台上。
5. 如果 `Flash` 未显示“属性”检查器，请选择“窗口” > “属性” > “属性”。

6. 选择 List 组件后，在“属性”检查器中设置属性：
 - 实例名称：myList
 - W（宽度）：100
 - H（高度）：400
 - X：20
 - Y：20
7. 单击“参数”面板，然后双击 **dataProvider** 行的第二列。
8. 在显示的“值”对话框中，单击加号两次以添加两个数据元素（标签分别设置为 label0 和 label1），然后单击“确定”按钮。
9. 使用“文本”工具在舞台上绘制一个文本字段。
10. 选择该文本字段后，在“属性”检查器中设置属性：
 - 文本类型：动态文本
 - 实例名称：textField
 - W（宽度）：100
 - 字体大小：24
 - X：0
 - Y：0
11. 选择该文本字段后，选择“修改”>“转换为元件”。
12. 在“转换为元件”对话框中进行以下设置，然后单击“确定”。
 - 名称：MyCellRenderer
 - 类型：MovieClip
 - 为 ActionScript 导出：选中
 - 在第一帧导出：选中
 - 类：MyCellRenderer
 - 基类：flash.display.SimpleButton

如果 Flash 显示“ActionScript 类警告”，请单击警告框中的“确定”按钮。
13. 从舞台上删除新影片剪辑元件的实例。
14. 在时间轴中选择图层 1 的第 1 帧，然后选择“窗口”>“动作”。
15. 在“动作”面板中键入以下脚本：

```
myList.setStyle("cellRenderer", MyCellRenderer);
```
16. 选择“文件”>“保存”。为文件指定一个名称，然后单击“确定”按钮。
17. 选择“文件”>“新建”。
18. 在显示的“新建文档”对话框中，选择“ActionScript 文件”，然后单击“确定”按钮。

19. 在脚本窗口中，输入以下代码来定义 **MyCellRenderer** 类：

```
package {
    import flash.display.MovieClip;
    import flash.filters.GlowFilter;
    import flash.text.TextField;
    import fl.controls.listClasses.ICellRenderer;
    import fl.controls.listClasses.ListData;
    import flash.utils.setInterval;
    public class MyCellRenderer extends MovieClip implements ICellRenderer
    {
        private var _listData:ListData;
        private var _data:Object;
        private var _selected:Boolean;
        private var glowFilter:GlowFilter;
        public function MyCellRenderer() {
            glowFilter = new GlowFilter(0xFFFF00);
            setInterval(toggleFilter, 200);
        }
        public function set data(d:Object):void {
            _data = d;
            textField.text = d.label;
        }
        public function get data():Object {
            return _data;
        }
        public function set listData(ld:ListData):void {
            _listData = ld;
        }
        public function get listData():ListData {
            return _listData;
        }
        public function set selected(s:Boolean):void {
            _selected = s;
        }
        public function get selected():Boolean {
            return _selected;
        }
        public function setSize(width:Number, height:Number):void {
        }
        public function setStyle(style:String, value:Object):void {
        }
        private function toggleFilter():void {
            if (textField.filters.length == 0) {
                textField.filters = [glowFilter];
            } else {
                textField.filters = [];
            }
        }
    }
}
```


20.选择“文件”>“保存”。将文件命名为 **MyCellRenderer.as**，并将其放在 **FLA** 文件所在的目录中，然后单击“确定”按钮。

21. 选择“控制”>“测试影片”。

CellRenderer 属性

data 属性是一个包含为 **CellRenderer** 设置的所有属性的对象。例如，在下面的类（该类定义了一个扩展 **Checkbox** 类的自定义 **CellRenderer**）中，注意 **data** 属性的 **setter** 函数将 **data.label** 的值传递给从 **CheckBox** 类继承的 **label** 属性：

```
public class CustomRenderer extends CheckBox implements ICellRenderer {
    private var _listData:ListData;
    private var _data:Object;
    public function CustomRenderer() {
    }
    public function set data(d:Object):void {
        _data = d;
        label = d.label;
    }
    public function get data():Object {
        return _data;
    }
    public function set listData(ld:ListData):void {
        _listData = ld;
    }
    public function get listData():ListData {
        return _listData;
    }
}
```

selected 属性用于定义列表中的某一单元格是否已被选中。

为 DataGrid 对象的列应用 CellRenderer

DataGrid 对象可以有多个列，并且可以为每个列指定不同的单元格渲染器。**DataGrid** 的每个列均由一个 **DataGridColumn** 对象表示，并且 **DataGridColumn** 类包含 **cellRenderer** 属性，可以通过此属性为该列定义 **CellRenderer**。

为可编辑单元格定义 CellRenderer

`DataGridCellEditor` 类定义了用于 `DataGrid` 对象中可编辑单元格的渲染器。当 `DataGrid` 对象的 `editable` 属性设置为 `true` 并且用户单击要编辑的单元格时，它将成为该单元格的渲染器。若要为可编辑单元格定义 `CellRenderer`，请为 `DataGrid` 对象的 `columns` 数组的每个元素设置 `itemEditor` 属性。

将图像、SWF 文件或影片剪辑用作 CellRenderer

`CellRenderer` 的子类 `ImageCell` 类定义用于渲染单元格（主要内容为图像、SWF 文件或影片剪辑的单元格）的对象。`ImageCell` 类包含以下用于定义单元格外观的样式：

- `imagePadding` — 用于分隔单元格边缘和图像边缘的填充（以像素为单位）
- `selectedSkin` — 用于指示所选状态的外观
- `textOverlayAlpha` — 单元格标签后面叠加层的不透明度
- `textPadding` — 用于分隔单元格边缘和文本边缘的填充（以像素为单位）

`ImageCell` 类是 `TileList` 类的默认 `CellRenderer`。

使组件具有辅助功能

可以通过屏幕阅读器使有视觉障碍的用户获知 **Flash** 应用程序中的可视内容，屏幕阅读器用于提供屏幕内容的语音描述。有关如何使 **Flash** 应用程序可由屏幕阅读器访问的信息，请参阅 [《使用 Flash》](#) 中的第 18 章“创建具有辅助功能的内容”。

若要使 **ActionScript 3.0** 组件可由屏幕阅读器访问，还必须导入该组件的辅助功能类并调用该类的 `enableAccessibility()` 方法。可以使以下 **ActionScript 3.0** 组件可由屏幕阅读器访问：

组件	辅助功能类
Button	ButtonAccImpl
CheckBox	CheckBoxAccImpl
ComboBox	ComboBoxAccImpl
List	ListAccImpl
RadioButton	RadioButtonAccImpl
TileList	TileListAccImpl

组件的辅助功能类在 `fl.accessibility` 软件包中。例如，若要使 `CheckBox` 可由屏幕阅读器访问，应在应用程序中添加以下语句：

```
import fl.accessibility.CheckBoxAccImpl;
```

```
CheckBoxAccImpl.enableAccessibility();
```

无论为一个组件创建了多少个实例，只需要对它启用辅助功能一次。



启用辅助功能后，由于在编译期间包含了所需的类，因而会略微增大文件的大小。

大多数组件还可以通过键盘来导航。有关启用辅助组件和使用键盘导航的详细信息，请参阅“使用 UI 组件”的“用户交互”章节和《[ActionScript 3.0 语言和组件参考](#)》中的[辅助功能类](#)。

使用 UI 组件

本章将介绍如何使用 Flash 随附的以下 **ActionScript 3.0** 用户界面 (UI) 组件：

使用 Button 组件	69
使用 CheckBox 组件	73
使用 ColorPicker 组件	76
使用 ComboBox 组件	78
使用 DataGrid 组件	82
使用 Label 组件	88
使用 List 组件	90
使用 NumericStepper	95
使用 ProgressBar	98
使用 RadioButton	104
使用 ScrollPane 组件	107
使用 Slider	110
使用 TextArea	113
使用 TextInput	116
使用 TileList	120
使用 UILoader	123
使用 UIScrollBar	125

使用 **Button** 组件

Button 组件是一个可调整大小的矩形按钮，用户可以通过鼠标或空格键按下该按钮以在应用程序中启动操作。可以给 **Button** 添加一个自定义图标。也可以将 **Button** 的行为从按下改为切换。在单击切换 **Button** 后，它将保持按下状态，直到再次单击时才会返回到弹起状态。

Button 是许多表单和 **Web** 应用程序的基础部分。每当您需要让用户启动一个事件时，都可以使用按钮实现。例如，大多数表单都有“提交”按钮。您也可以给演示文稿添加“上一个”和“下一个”按钮。


与 Button 进行用户交互

可以在应用程序中启用或者禁用按钮。在禁用状态下，按钮不接收鼠标或键盘输入。如果单击或者切换到某个按钮，处于启用状态的就会接收焦点。当 **Button** 实例具有焦点时，可以使用以下按键来控制它：

键	说明
Shift+Tab	将焦点移到上一个对象。
空格键	按下或释放按钮并触发 click 事件。
Tab	将焦点移到下一个对象。
Enter/Return	如果按钮设置为 FocusManager 的默认 Button，则将焦点移到下一个对象。

有关控制焦点的详细信息，请参阅 [《ActionScript 3.0 语言和组件参考》](#) 中的 [IFocusManager](#) 接口和 [FocusManager](#) 类，以及第 48 页的“使用 FocusManager”。

每个 **Button** 实例的实时预览反映在创作过程中对“属性”检查器或“组件”检查器中的参数所做的更改。

如果图标比按钮大，它将会延伸到按钮的边框外。

若要将应用程序中的某个按钮指定为默认的普通按钮（当用户按 **Enter** 时接收单击事件的按钮），请设置 `FocusManager.defaultButton`。例如，以下代码将默认按钮设置为名为 `submitButton` 的 **Button** 实例。

```
FocusManager.defaultButton = submitButton;
```

在将 **Button** 组件添加到应用程序时，可以添加以下数行 **ActionScript** 代码，使屏幕阅读器能够访问它：

```
import fl.accessibility.ButtonAccImpl;

ButtonAccImpl.enableAccessibility();
```

无论为一个组件创建了多少个实例，只需要对它启用辅助功能一次。

Button 参数

可以在“属性”检查器（“窗口” > “属性” > “属性”）或“组件”检查器（“窗口” > “组件检查器”）中为每个 **Button** 实例设置以下创作参数：`emphasized`、`label`、`labelPlacement`、`selected` 和 `toggle`。其中每个参数都有对应的同名 **ActionScript** 属性。为这些参数赋值时，将设置应用程序中属性的初始状态。在 **ActionScript** 中设置的属性会覆盖在对应参数中设置的值。有关这些参数可能值的信息，请参阅 [《ActionScript 3.0 语言和组件参考》](#) 中的 **Button** 类。

创建具有 Button 组件的应用程序

以下过程解释了如何在创作时将 **Button** 组件添加到应用程序。在此示例中，单击 **Button** 时会更改 **ColorPicker** 组件的状态。

创建具有 Button 组件的应用程序：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将一个 **Button** 组件从“组件”面板拖到舞台上，并在“属性”检查器中为该组件输入以下值：
 - 输入实例名称 **aButton**。
 - 为 **label** 参数输入 **Show**。
3. 在舞台上添加 **ColorPicker**，并为它指定实例名称 **aCp**。
4. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 **ActionScript** 代码：

```
aCp.visible = false;

aButton.addEventListener(MouseEvent.CLICK, clickHandler);

function clickHandler(event:MouseEvent):void {

    switch(event.currentTarget.label) {
        case "Show":
            aCp.visible = true;
            aButton.label = "Disable";
            break;
        case "Disable":
            aCp.enabled = false;
            aButton.label = "Enable";
            break;
        case "Enable":
            aCp.enabled = true;
            aButton.label = "Hide";
            break;
        case "Hide":
            aCp.visible = false;
            aButton.label = "Show";
            break;
    }
}
```

第二行代码将函数 `clickHandler()` 注册为 `MouseEvent.CLICK` 事件的事件处理函数。用户单击 **Button** 时，事件将发生，从而使 `clickHandler()` 函数根据 **Button** 的值执行以下操作之一：

- **Show** 使 **ColorPicker** 可见，并将 **Button** 的标签更改为 **Disable**。
- **Disable** 禁用 **ColorPicker**，并将 **Button** 的标签更改为 **Enable**。
- **Enable** 启用 **ColorPicker**，并将 **Button** 的标签更改为 **Hide**。
- **Hide** 使 **ColorPicker** 不可见，并将 **Button** 的标签更改为 **Show**。

5. 选择“控制” > “测试影片”，运行应用程序。

以下过程使用 **ActionScript** 创建一个切换 **Button**，并且当单击该 **Button** 时在“输出”面板中显示事件类型。该示例通过调用类的构造函数创建一个 **Button** 实例，并通过调用 `addChild()` 方法将该实例添加到舞台上。

使用 **ActionScript** 代码创建 **Button**：

1. 创建一个新的 **Flash** 文件 (**ActionScript 3.0**) 文档。

2. 将 **Button** 组件从“组件”面板拖到当前文档的“库”面板中。

此操作将组件添加到库中，但不会在应用程序中显示它。

3. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下代码创建一个 **Button** 实例：

```
import fl.controls.Button;

var aButton:Button = new Button();
addChild(aButton);
aButton.label = "Click me";
aButton.toggle = true;
aButton.move(50, 50);
```

`move()` 方法将按钮放在舞台的 50 (x 坐标)，50 (y 坐标) 位置。

4. 现在，添加以下 **ActionScript** 来创建一个事件侦听器和一个事件处理函数：

```
aButton.addEventListener(MouseEvent.CLICK, clickHandler);

function clickHandler(event:MouseEvent):void {
    trace("Event type: " + event.type);
}
```

5. 选择“控制” > “测试影片”。

单击按钮时，**Flash** 会在“输出”面板中显示消息“事件类型：单击”。

使用 CheckBox 组件

CheckBox 组件是一个可以选中或取消选中的方框。当它被选中后，框中会出现一个复选标记。可以为 **CheckBox** 添加一个文本标签，并可以将它放在 **CheckBox** 的左侧、右侧、上面或下面。

可以使用 **CheckBox** 收集一组不相互排斥的 `true` 或 `false` 值。例如，如果应用程序需要收集有关您希望购买何种类型轿车的信息，那么可以使用 **CheckBox** 来供您选择轿车特征。

与 CheckBox 进行用户交互

可以在应用程序中启用或者禁用 **CheckBox**。如果 **CheckBox** 已启用，且用户单击它或者它的标签，则 **CheckBox** 会获得输入焦点并显示为按下状态。如果用户在按下鼠标按键时将鼠标指针移到 **CheckBox** 或其标签的边界区域之外，则组件的外观会返回到其原始状态，并保持输入焦点。在组件上释放鼠标之前，**CheckBox** 的状态不会更改。另外，**CheckBox** 有选中和取消选中两种禁用状态，这两种状态分别使用 `selectedDisabledSkin` 和 `disabledSkin` 设置，不允许鼠标或键盘的交互操作。

如果 **CheckBox** 被禁用，则无论用户进行什么交互操作，它都会显示其禁用外观。在禁用状态下，**CheckBox** 不接收鼠标或键盘输入。

如果用户单击 **CheckBox** 实例或者用 **Tab** 按键切换到它时，**CheckBox** 实例将接收焦点。当 **CheckBox** 实例具有焦点时，可以使用以下按键来控制它：

键	说明
Shift+Tab	将焦点移到上一个元素。
空格键	选中或者取消选中组件，并触发 <code>change</code> 事件。
Tab	将焦点移到下一个元素。

有关控制焦点的详细信息，请参阅第 48 页的“使用 **FocusManager**”和《**ActionScript 3.0 语言和组件参考**》中的 **FocusManager** 类。

每个 **CheckBox** 实例的实时预览反映在创作过程中对“属性”检查器或“组件”检查器中的参数所做的更改。

在将 **CheckBox** 组件添加到应用程序时，可以添加以下数行 **ActionScript** 代码，使屏幕阅读器能够访问它：

```
import fl.accessibility.CheckBoxAccImpl;

CheckBoxAccImpl.enableAccessibility();
```

无论组件有多少实例，都只对组件启用一次辅助功能。

CheckBox 参数

可以在“属性”检查器或“组件”检查器中为每个 [CheckBox](#) 组件实例设置以下创作参数：`label`、`labelPlacement` 和 `selected`。其中每个参数都有对应的同名 `ActionScript` 属性。有关这些参数可能值的信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 `CheckBox` 类。

创建具有 CheckBox 组件的应用程序

以下过程解释了如何在创作时将 `CheckBox` 组件添加到应用程序，本示例摘自一个贷款申请表。该申请表询问申请人是否拥有自己的住房，并为申请人提供了一个 `CheckBox` 来回答“是”。如果是，则申请表将为申请人显示两个单选按钮，以表示房屋的相对价值。

创建具有 CheckBox 组件的应用程序：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将一个 `CheckBox` 组件从“组件”面板拖到舞台上。
3. 在“属性”检查器中，执行以下操作：
 - 输入 **homeCh** 作为实例名称。
 - 输入 **140** 作为宽度 (W) 值。
 - 输入 **“Own your home?”** 作为 `label` 参数。
4. 将两个 `RadioButton` 组件从“组件”面板拖到舞台上，并将它们分别放在 `CheckBox` 的下面和右侧。在“属性”检查器中为它们输入以下值：
 - 输入 **underRb** 和 **overRb** 作为实例名称。
 - 输入 **120** 作为两个单选按钮的 W（宽）参数。
 - 输入 **“Under \$500,000?”** 作为 `underRb` 的 `label` 参数。
 - 输入 **“Over \$500,000?”** 作为 `overRb` 的 `label` 参数。
 - 输入 **valueGrp** 作为两个单选按钮的 `groupName` 参数。
5. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 `ActionScript` 代码：

```
homeCh.addEventListener(MouseEvent.CLICK, clickHandler);
underRb.enabled = false;
overRb.enabled = false;

function clickHandler(event:MouseEvent):void {
    underRb.enabled = event.target.selected;
    overRb.enabled = event.target.selected;
}
```

此代码为 `click` 事件创建一个事件处理函数，如果选中 `homeCh` 复选框，则该函数启用 `underRb` 和 `overRb` 单选按钮，如果未选中 `homeCh`，则禁用这两个单选按钮。有关详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 `MouseEvent` 类。

6. 选择“控制” > “测试影片”。

以下示例重复了上述应用，但采用 **ActionScript** 来创建 **CheckBox** 和 **RadioButton** 组件。

使用 **ActionScript** 创建 **CheckBox**:

1. 创建一个新的 **Flash** 文件 (**ActionScript 3.0**) 文档。
2. 将 **CheckBox** 组件和 **RadioButton** 组件从“组件”面板拖到当前文档的“库”面板中。
如果“库”面板没有打开，请按 **Ctrl+L** 或选择“窗口” > “库”以打开“库”面板。
此操作使组件可用于应用程序，但不会将它们放在舞台上。
3. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下代码创建并放置组件实例：

```
import fl.controls.CheckBox;
import fl.controls.RadioButton;

var homeCh:CheckBox = new CheckBox();
var underRb:RadioButton = new RadioButton();
var overRb:RadioButton = new RadioButton();
addChild(homeCh);
addChild(underRb);
addChild(overRb);
underRb.groupName = "valueGrp";
overRb.groupName = "valueGrp";
homeCh.move(200, 100);
homeCh.width = 120;
homeCh.label = "Own your home?";
underRb.move(220, 130);
underRb.enabled = false;
underRb.width = 120;
underRb.label = "Under $500,000?";
overRb.move(220, 150);
overRb.enabled = false;
overRb.width = 120;
overRb.label = "Over $500,000?";
```

此代码使用 **CheckBox()** 和 **RadioButton()** 构造函数创建组件，并使用 **addChild()** 方法将组件放在舞台上。然后，使用 **move()** 方法确定组件在舞台上的位置。

4. 现在，添加以下 **ActionScript** 来创建一个事件侦听器和一个事件处理函数：

```
homeCh.addEventListener(MouseEvent.CLICK, clickHandler);

function clickHandler(event:MouseEvent):void {
    underRb.enabled = event.target.selected;
    overRb.enabled = event.target.selected;
}
```

此代码为 **click** 事件创建一个事件处理函数，如果选中 **homeCh** **CheckBox**，则该函数启用 **underRb** 和 **overRb** 单选按钮，如果未选中 **homeCh**，则禁用这两个单选按钮。有关详细信息，请参阅《**ActionScript 3.0 语言和组件参考**》中的 **MouseEvent** 类。

5. 选择“控制” > “测试影片”。

使用 ColorPicker 组件

ColorPicker 组件允许用户从样本列表中选择颜色。**ColorPicker** 的默认模式是在方形按钮中显示单一颜色。用户单击按钮时，样本面板中将出现可用的颜色列表，同时出现一个文本字段，显示当前所选颜色的十六进制值。

可以通过将 `colors` 属性设置为要显示的颜色值，来设置出现在 **ColorPicker** 中的颜色。

与 ColorPicker 组件进行用户交互

ColorPicker 允许用户选择颜色，并将该颜色应用于应用程序中的另一个对象。例如，如果要允许用户个性化应用程序的某些元素，如背景颜色或文本颜色，那么可以包括一个 **ColorPicker**，并应用用户所选择的颜色。

用户通过单击面板中的颜色样本或在文本字段中输入颜色的十六进制值来选择颜色。用户选择颜色之后，您可以使用 **ColorPicker** 的 `selectedColor` 属性，将该颜色应用于文本或应用程序中的另一个对象。

如果用户将鼠标指针移到 **ColorPicker** 实例上或通过 **Tab** 切换到该实例，则该实例将获得焦点。当 **ColorPicker** 实例的样本面板打开时，可以使用以下按键来控制它：

键	说明
Home	将选区移到样本面板中的第一种颜色。
向上键	在样本面板中将选区向上移动一行。
向下键	在样本面板中将选区向下移动一行。
向右键	将选区从样本面板中的一种颜色移到该颜色的右侧。
向左键	将选区从样本面板中的一种颜色移到该颜色的左侧。
End	将选区移到样本面板中的最后一种颜色。

ColorPicker 参数

您可以在“属性”检查器或“组件”检查器中，为每个 **ColorPicker** 实例设置以下创作参数：`selectedColor` 和 `showTextField`。其中每个参数都有对应的同名 **ActionScript** 属性。有关这些参数可能值的信息，请参阅 [《ActionScript 3.0 语言和组件参考》](#) 中的 **ColorPicker** 类。

创建具有 ColorPicker 组件的应用程序

以下示例在创作时将 **ColorPicker** 组件添加到应用程序。在此示例中，每当您在 **ColorPicker** 中更改颜色时，`changeHandler()` 函数将调用 `drawBox()` 函数，用您在 **ColorPicker** 中所选择的颜色绘制一个新的框。

创建具有 ColorPicker 组件的应用程序：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将一个 **ColorPicker** 组件从“组件”面板拖到舞台的中央，然后为其指定实例名 **aCp**。
3. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 ActionScript 代码：

```
import fl.events.ColorPickerEvent;

var aBox:MovieClip = new MovieClip();
drawBox(aBox, 0xFF0000); // draw a red box
addChild(aBox);

aCp.addEventListener(ColorPickerEvent.CHANGE, changeHandler);

function changeHandler(event:ColorPickerEvent):void {
    drawBox(aBox, event.target.selectedColor);
}

function drawBox(box:MovieClip, color:uint):void {
    box.graphics.beginFill(color, 1);
    box.graphics.drawRect(100, 150, 100, 100);
    box.graphics.endFill();
}
```

4. 选择“控制” > “测试影片”。
5. 单击 **ColorPicker**，并选择要用于框的颜色。

以下示例使用 `ColorPicker()` 构造函数和 `addChild()` 在舞台上创建一个 **ColorPicker**。该示例将 `colors` 属性设置为红色 (0xFF0000)、绿色 (0x00FF00) 和蓝色 (0x0000FF) 的颜色值，以指定 **ColorPicker** 将显示的颜色。还会创建一个 **TextArea** 组件，每当您在 **ColorPicker** 中选择一种不同的颜色时，该示例将更改 **TextArea** 中文本的颜色，以与之匹配。

使用 ActionScript 创建 ColorPicker 组件：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将一个 **ColorPicker** 组件从“组件”面板拖到“库”面板中。
3. 将一个 **TextArea** 组件从“组件”面板拖到“库”面板中。

4. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 **ActionScript** 代码：

```
import fl.controls.ColorPicker;
import fl.controls.TextArea;
import fl.events.ColorPickerEvent;

var aCp:ColorPicker = new ColorPicker();
var aTa:TextArea = new TextArea();
var aTf:TextFormat = new TextFormat();

aCp.move(100, 100);
aCp.addEventListener(ColorPickerEvent.CHANGE, changeHandler);

aTa.text = "Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Vivamus quis nisl vel tortor nonummy vulputate. Quisque sit amet eros
sed purus euismod tempor. Morbi tempor. Class aptent taciti sociosqu ad
litora torquent per conubia nostra, per inceptos hymenaeos. Curabitur
diam. Suspendisse at purus in ipsum volutpat viverra. Nulla
pellentesque libero id libero.";
aTa.setSize(200, 200);
aTa.move(200,100);

addChild(aCp);
addChild(aTa);

function changeHandler(event:ColorPickerEvent):void {
    if(TextFormat(aTa.getStyle("textFormat"))){
        aTf = TextFormat(aTa.getStyle("textFormat"));
    }
    aTf.color = event.target.selectedColor;
    aTa.setStyle("textFormat", aTf);
}
```

5. 选择“控制” > “测试影片”。

使用 ComboBox 组件

ComboBox 组件允许用户从下拉列表中进行单项选择。**ComboBox** 可以是静态的，也可以是可编辑的。可编辑的 **ComboBox** 允许用户在列表顶端的文本字段中直接输入文本。如果下拉列表超出文档底部，该列表将会向上打开，而不是向下。**ComboBox** 由三个子组件构成：**BaseButton**、**TextInput** 和 **List** 组件。

在可编辑的 **ComboBox** 中，只有按钮是点击区域，文本框不是。对于静态 **ComboBox**，按钮和文本框一起组成点击区域。点击区域通过打开或关闭下拉列表来做出响应。

当用户在列表中做出选择（无论通过鼠标还是键盘）时，所选项的标签将复制到 **ComboBox** 顶端的文本字段中。

与 ComboBox 组件进行用户交互

在任何需要从列表中选择一项的表单或应用程序中，都可以使用 **ComboBox** 组件。例如，您可以在客户地址表单中提供一个州 / 省的下拉列表。对于比较复杂的情况，可以使用可编辑的 **ComboBox**。例如，在提供驾驶方向的应用程序中，您可以使用一个可编辑的 **ComboBox** 以允许用户输入出发地址和目标地址。下拉列表可以包含用户以前输入过的地址。

如果 **ComboBox** 可编辑，即 `editable` 属性为 `true`，则以下按键将从文本输入框中移除焦点，并保留先前的值。但 **Enter** 键除外，如果用户输入文本，该键将首先应用新的值。

键	说明
Shift+Tab	将焦点移到上一项。如果选择了新的项，则调度 <code>change</code> 事件。
Tab	将焦点移到下一项。如果选择了新的项，则调度 <code>change</code> 事件。
向下键	将选区向下移动一项。
End	将选区移到列表底端。
Esc	关闭下拉列表，并将焦点返回到 ComboBox 。
Enter	关闭下拉列表，并将焦点返回到 ComboBox 。当 ComboBox 可编辑，且用户输入文本时， <code>Enter</code> 会将值设置为输入的文本。
Home	将选区移到列表顶端。
Page Up	将选区向上移动一页。
Page Down	将选区向下移动一页。

在将 **ComboBox** 组件添加到应用程序时，可以添加以下数行 **ActionScript** 代码，使屏幕阅读器能够访问它：

```
import fl.accessibility.ComboBoxAccImpl;

ComboBoxAccImpl.enableAccessibility();
```

无论组件有多少实例，都只对组件启用一次辅助功能。

ComboBox 参数

可以在“属性”检查器或“组件”检查器中为每个 **ComboBox** 实例设置以下参数：

`dataProvider`、`editable`、`prompt` 和 `rowCount`。其中每个参数都有对应的同名 **ActionScript** 属性。有关这些参数可能值的信息，请参阅 [《ActionScript 3.0 语言和组件参考》](#) 中的 **ComboBox** 类。有关使用 `dataProvider` 参数的信息，请参阅第 50 页的“使用 `dataProvider` 参数”。

创建具有 ComboBox 组件的应用程序

以下过程解释了如何在创作时将 **ComboBox** 组件添加到应用程序。该 **ComboBox** 是可编辑的，如果您在文本字段中键入 **Add**，则示例会将该项添加到下拉列表中。

创建具有 ComboBox 组件的应用程序：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将一个 **ComboBox** 组件拖到舞台上，然后为其指定实例名称 **aCb**。在“参数”选项卡上，将 `editable` 参数设置为 `true`。
3. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下代码：

```
import fl.data.DataProvider;
import fl.events.ComponentEvent;

var items:Array = [
    {label:"screen1", data:"screenData1"},
    {label:"screen2", data:"screenData2"},
    {label:"screen3", data:"screenData3"},
    {label:"screen4", data:"screenData4"},
    {label:"screen5", data:"screenData5"},
];
aCb.dataProvider = new DataProvider(items);

aCb.addEventListener(ComponentEvent.ENTER, onAddItem);

function onAddItem(event:ComponentEvent):void {
    var newRow:int = 0;
    if (event.target.text == "Add") {
        newRow = event.target.length + 1;
        event.target.addItemAt({label:"screen" + newRow, data:"screenData" +
            newRow},
            event.target.length);
    }
}
```

4. 选择“控制” > “测试影片”。

以下示例用 **ActionScript** 创建一个 **ComboBox** 组件，并用加利福尼亚州旧金山地区的大学列表进行填充。该示例根据提示文本的宽来设置 **ComboBox** 的 `width` 属性，并将 `dropdownWidth` 属性设置得稍宽一些，以容纳最长的大学名称。

该示例在 **Array** 实例中创建一个大学列表，使用 `label` 属性来存储学校名称，并使用 `data` 属性来存储每个学校网站的 **URL**。然后，通过设置 **ComboBox** 的 `dataProvider` 属性将该 **Array** 分配给 **ComboBox**。

当用户从列表中选择大学时，将触发 **Event.CHANGE** 事件，并调用 `changeHandler()` 函数，该函数将 `data` 属性加载到 **URL** 请求中，以访问学校的网站。

请注意，最后一行将 **ComboBox** 实例的 `selectedIndex` 属性设置为 `-1`，以便在列表关闭时重新显示提示。否则，将显示所选择的学校名称而不是提示。

使用 ActionScript 创建 ComboBox 组件：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将一个 **ComboBox** 组件从“组件”面板拖到“库”面板中。
3. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 ActionScript 代码：

```
import fl.controls.ComboBox;
import fl.data.DataProvider;
import flash.net.navigateToURL;

var sfUniversities:Array = new Array(
    {label:"University of California, Berkeley",
     data:"http://www.berkeley.edu/"},
    {label:"University of San Francisco",
     data:"http://www.usfca.edu/"},
    {label:"San Francisco State University",
     data:"http://www.sfsu.edu/"},
    {label:"California State University, East Bay",
     data:"http://www.csu Hayward.edu/"},
    {label:"Stanford University", data:"http://www.stanford.edu/"},
    {label:"University of Santa Clara", data:"http://www.scu.edu/"},
    {label:"San Jose State University", data:"http://www.sjsu.edu/"}
);

var aCb:ComboBox = new ComboBox();
aCb.dropdownWidth = 210;
aCb.width = 200;
aCb.move(150, 50);
aCb.prompt = "San Francisco Area Universities";
aCb.dataProvider = new DataProvider(sfUniversities);
aCb.addEventListener(Event.CHANGE, changeHandler);

addChild(aCb);

function changeHandler(event:Event):void {
    var request:URLRequest = new URLRequest();
    request.url = ComboBox(event.target).selectedItem.data;
    navigateToURL(request);
    aCb.selectedIndex = -1;
}
```

4. 选择 “控制” > “测试影片”。

您可以在 **Flash** 创作环境中实现并运行此示例，但如果您试图通过单击 **ComboBox** 中的项来访问大学网站，则会显示警告消息。若要在 **Internet** 上访问功能齐全的 **ComboBox**，请访问以下网址：

<http://www.helpexamples.com/peter/bayAreaColleges/bayAreaColleges.html>

使用 DataGrid 组件

DataGrid 组件允许用户将数据显示在行和列构成的网格中，数据来自数组或来自 **DataProvider** 可以解析为数组的外部 XML 文件。 **DataGrid** 组件包括垂直和水平滚动、事件支持（包括对可编辑单元格的支持）和排序功能。

您可以调整和自定义诸如网格中的字体、颜色以及列边框等特征。对于网格中的任何列，都可以使用自定义影片剪辑作为单元格渲染器。（单元格渲染器显示单元格的内容。）可以禁用滚动条，并使用 **DataGrid** 方法来创建页面视图样式的显示。有关自定义的详细信息，请参阅《**ActionScript 3.0 语言和组件参考**》中的 **DataGridColumn** 类。

与 DataGrid 组件进行用户交互

可以使用鼠标和键盘与 **DataGrid** 组件进行交互。

如果 `sortableColumns` 属性和列的 `sortable` 属性均为 `true`，则单击列标题时会根据列的值对数据进行排序。可以通过将某个列的 `sortable` 属性设置为 `false` 来禁用该列的排序功能。

如果 `resizableColumns` 属性为 `true`，则可以通过在标题行中拖动列分隔符来调整列的大小。

在可编辑单元格内单击会将焦点赋予该单元格，单击不可编辑的单元格不会影响焦点。如果某个单元格的 `DataGrid.editable` 和 `DataGridColumn.editable` 属性均为 `true`，则该单元格是可编辑的。

有关详细信息，请参阅《**ActionScript 3.0 语言和组件参考**》中的 **DataGrid** 和 **DataGridColumn** 类。

当 **DataGrid** 实例从单击或 **Tab** 键切换中获得焦点时，您可以使用以下按键来控制它：

键	说明
向下键	如果正在编辑单元格，则插入点将移到单元格文本的末尾。如果单元格不可编辑，则向下箭头处理选区的方式与 List 组件相同。
向上键	如果正在编辑单元格，则插入点将移到单元格文本的开头。如果单元格不可编辑，则向上箭头处理选区的方式与 List 组件相同。
Shift+ 向上箭头 / 向下箭头	如果 DataGrid 不可编辑，且 <code>allowMultipleSelection</code> 为 <code>true</code> ，则选择连续多行。用反向箭头反转方向可取消选择所选的行，直至经过最开始的一行，从这一行开始沿反方向选择行。

键	说明
Shift+ 单击	如果 allowMultipleSelection 为 true，则选择所选行与当前尖号位置（突出显示的单元格）之间的所有行。
Ctrl+ 单击	如果 allowMultipleSelection 为 true，则选择更多行，这些行不一定是连续的。
向右键	如果正在编辑单元格，则插入点将向右移动一个字符。如果单元格不可编辑，则向右箭头不执行任何操作。
向左键	如果正在编辑单元格，则插入点将向左移动一个字符。如果单元格不可编辑，则向左箭头不执行任何操作。
Home	选择 DataGrid 中的第一行。
End	选择 DataGrid 中的最后一行。
Page Up	选择 DataGrid 页中的第一行。页由 DataGrid 无需滚动即可显示的多行组成。
Page Down	选择 DataGrid 页中的最后一行。页由 DataGrid 无需滚动即可显示的多行组成。
Return/Enter/ Shift+Enter	如果单元格可编辑，则会提交更改，并且插入点将移到同一列中单元格的下一行（向上或向下，视 Shift 切换而定）。
Shift+Tab/Tab	如果 DataGrid 可编辑，则将焦点移到上一项 / 下一项，直至列的末尾，然后移到上一行 / 下一行，直至第一个或最后一个单元格。如果所选的是第一个单元格，则 Shift+Tab 将焦点移到上一个控件。如果所选的是最后一个单元格，则 Tab 将焦点移到下一个控件。 如果 DataGrid 不可编辑，则将焦点移到上一个 / 下一个控件。

可以使用 **DataGrid** 组件作为许多种数据驱动应用程序的基础。您不但可以轻松地显示数据的格式化表格视图，而且可以使用单元格渲染器功能建立更为复杂和可编辑的用户界面片段。以下是 **DataGrid** 组件的实际用途：

- Webmail 客户端
- 搜索结果页
- 电子表格应用程序，如借贷计算器和纳税申请表应用程序

在设计具有 **DataGrid** 组件的应用程序时，了解 **List** 组件的设计很有帮助，因为 **DataGrid** 类扩展自 **SelectableList** 类。有关 **SelectableList** 类和 **List** 组件的详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 [SelectableList](#) 和 [List](#) 类。

在将 **DataGrid** 组件添加到应用程序时，可以添加以下数行 **ActionScript** 代码，使屏幕阅读器能够访问它：

```
import fl.accessibility.DataGridAccImpl;
DataGridAccImpl.enableAccessibility();
```

无论组件有多少实例，都只对组件启用一次辅助功能。有关详细信息，请参阅《使用 **Flash**》中的第 18 章“创建具有辅助功能的内容”。

DataGrid 参数

可以在“属性”检查器或“组件”检查器中为每个 **DataGrid** 组件实例设置以下创作参数：

allowMultipleSelection、editable、headerHeight、horizontalLineScrollSize、horizontalPageScrollSize、horizontalScrollPolicy、resizableColumns、rowHeight、showHeaders、verticalLineScrollSize、verticalPageScrollSize 和 verticalScrollPolicy。其中每个参数都有对应的同名 **ActionScript** 属性。有关这些参数可能值的信息，请参阅《**ActionScript 3.0 语言和组件参考**》中的 **DataGrid** 类。

创建具有 DataGrid 组件的应用程序

若要创建具有 **DataGrid** 组件的应用程序，首先必须确定数据的来源。通常，数据来自数组，您可以通过设置 **dataProvider** 属性将数组拉入网格。也可以使用 **DataGrid** 和 **DataGridColumn** 类的方法将数据添加到网格中。

以下示例创建一个 **DataGrid**，以显示全球球队的花名册。该示例在一个 **Array** (**aRoster**) 中定义花名册，然后将其分配给 **DataGrid** 的 **dataProvider** 属性。

使用本地数据提供者和 DataGrid 组件：

1. 在 **Flash** 中，选择“文件”>“新建”，然后选择“Flash 文件 (ActionScript 3.0)”。
2. 将一个 **DataGrid** 组件从“组件”面板拖到舞台上。
3. 在“属性”检查器中，输入实例名称“**aDg**”。
4. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 **ActionScript** 代码：

```
import fl.data.DataProvider;

bldRosterGrid(aDg);
var aRoster:Array = new Array();
aRoster = [
    {Name:"Wilma Carter", Bats:"R", Throws:"R", Year:"So", Home:
    "Redlands, CA"},
    {Name:"Sue Pennypacker", Bats:"L", Throws:"R", Year:"Fr", Home:
    "Athens, GA"},
    {Name:"Jill Smithfield", Bats:"R", Throws:"L", Year:"Sr", Home:
    "Spokane, WA"}];
```

```

        {Name:"Shirley Goth", Bats:"R", Throws:"R", Year:"Sr", Home: "Carson,
NV"},
        {Name:"Jennifer Dunbar", Bats:"R", Throws:"R", Year:"Fr", Home:
"Seaside, CA"},
        {Name:"Patty Crawford", Bats:"L", Throws:"L", Year:"Jr", Home:
"Whittier, CA"},
        {Name:"Angelina Davis", Bats:"R", Throws:"R", Year:"So", Home:
"Odessa, TX"},
        {Name:"Maria Santiago", Bats:"L", Throws:"L", Year:"Sr", Home:
"Tacoma, WA"},
        {Name:"Debbie Ferguson", Bats:"R", Throws:"R", Year: "Jr", Home:
"Bend, OR"},
        {Name:"Karen Bronson", Bats:"R", Throws:"R", Year: "Sr", Home:
"Billings, MO"},
        {Name:"Sylvia Munson", Bats:"R", Throws:"R", Year: "Jr", Home:
"Pasadena, CA"},
        {Name:"Carla Gomez", Bats:"R", Throws:"L", Year: "Sr", Home: "Corona,
CA"},
        {Name:"Betty Kay", Bats:"R", Throws:"R", Year: "Fr", Home:
"Palo Alto, CA"},
];
aDg.dataProvider = new DataProvider(aRoster);
aDg.rowCount = aDg.length;

function bldRosterGrid(dg:DataGrid){
    dg.setSize(400, 300);
    dg.columns = ["Name", "Bats", "Throws", "Year", "Home"];
    dg.columns[0].width = 120;
    dg.columns[1].width = 50;
    dg.columns[2].width = 50;
    dg.columns[3].width = 40;
    dg.columns[4].width = 120;
    dg.move(50,50);
};

```

bldRosterGrid() 函数设置 **DataGrid** 的大小以及列的顺序与大小。

5. 选择“控制” > “测试影片”。

请注意，您可以单击任意列标题，以根据该列的值按降序对 **DataGrid** 的内容进行排序。

以下示例使用 addColumn() 方法将 **DataGridColumn** 实例添加到 **DataGrid** 中。列表示队员的姓名及其分数。该示例还设置了 sortOptions 属性，以指定每一列的排序选项：Array.CASEINSENSITIVE 用于 **Name** 列，Array.NUMERIC 用于 **Score** 列。此外，该示例将 **DataGrid** 的长设置为行数，宽设置为 200，从而适当地调整 **DataGrid** 的大小。

为应用程序中的 **DataGrid** 组件指定列和添加排序：

1. 在 Flash 中，选择“文件”>“新建”，然后选择“Flash 文件 (ActionScript 3.0)”。
2. 将一个 **DataGrid** 组件从“组件”面板拖到舞台上。
3. 在“属性”检查器中，输入实例名称“**aDg**”。
4. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 **ActionScript** 代码：

```
import fl.controls.dataGridClasses.DataGridColumn;
import fl.events.DataGridEvent;
import fl.data.DataProvider;
// Create columns to enable sorting of data.
var nameDGC:DataGridColumn = new DataGridColumn("name");
nameDGC.sortOptions = Array.CASEINSENSITIVE;
var scoreDGC:DataGridColumn = new DataGridColumn("score");
scoreDGC.sortOptions = Array.NUMERIC;
aDg.addColumn(nameDGC);
aDg.addColumn(scoreDGC);
var aDP_array:Array = new Array({name:"clark", score:3135},
    {name:"Bruce", score:403}, {name:"Peter", score:25});
aDg.dataProvider = new DataProvider(aDP_array);
aDg.rowCount = aDg.length;
aDg.width = 200;
```

5. 选择“控制”>“测试影片”。

以下示例使用 **ActionScript** 创建一个 **DataGrid** 组件，并用队员姓名和分数的数组进行填充。

使用 ActionScript 创建 DataGrid 组件实例：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将一个 **DataGrid** 组件从“组件”面板拖到当前文档的“库”面板中。
此操作将组件添加到库中，但不会在应用程序中显示它。
3. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 **ActionScript** 代码：

```
import fl.controls.DataGrid;
import fl.data.DataProvider;

var aDg:DataGrid = new DataGrid();
addChild(aDg);
aDg.columns = [ "Name", "Score" ];
aDg.setSize(140, 100);
aDg.move(10, 40);
```

此代码创建 **DataGrid** 实例，然后调整网格大小和定位网格。

4. 创建一个数组，向数组中添加数据，并将该数组标识为 **DataGrid** 的数据提供者：

```
var aDP_array:Array = new Array();
aDP_array.push({Name:"Clark", Score:3135});
aDP_array.push({Name:"Bruce", Score:403});
aDP_array.push({Name:"Peter", Score:25});
aDg.dataProvider = new DataProvider(aDP_array);
aDg.rowCount = aDg.length;
```

5. 选择 “控制” > “测试影片”。

以下示例使用 **DataGridColumn** 类创建 **DataGrid** 的列。该示例通过将 XML 对象作为 **DataProvider()** 构造函数的 **value** 参数传递，来填充 **DataGrid**。

用 XML 文件加载 DataGrid:

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 在 “组件” 面板中，双击 **DataGrid** 组件以将其添加到舞台上。
3. 在 “属性” 检查器中，输入实例名称 “**aDg**”。
4. 打开 “动作” 面板，在主时间轴中选择第 1 帧，然后输入以下 ActionScript 代码：

```
import fl.controls.dataGridClasses.DataGridColumn;
import fl.data.DataProvider;

var teamXML:XML = <team>
    <player name="Player A" avg="0.293" />
    <player name="Player B" avg="0.214" />
    <player name="Player C" avg="0.317" />
</team>;

var nameCol:DataGridColumn = new DataGridColumn("name");
nameCol.headerText = "Name";
nameCol.width = 120;
var avgCol:DataGridColumn = new DataGridColumn("avg");
avgCol.headerText = "Average";
avgCol.width = 60;

var myDP:DataProvider = new DataProvider(teamXML);

aDg.columns = [nameCol, avgCol];
aDg.width = 200;
aDg.dataProvider = myDP;
aDg.rowCount = aDg.length;
```

5. 选择 “控制” > “测试影片”。

使用 Label 组件

Label 组件显示单行文本，通常用于标识网页上的其它某些元素或活动。可以指定标签采用 HTML 格式，以利用其文本格式的标签。还可以控制标签的对齐方式和大小。**Label** 组件没有边框、不能具有焦点，并且不广播任何事件。

每个 **Label** 实例的实时预览反映了创作期间在“属性”检查器或“组件”检查器中对参数所做的更改。标签没有边框，因此，查看其实时预览的唯一方法是设置其 **text** 参数。

与 Label 组件进行用户交互

Label 组件用于为表单中的另一个组件创建文本标签，例如，**TextInput** 字段左侧的“姓名：”标签接受用户的姓名。最好用 **Label** 组件替代纯文本字段，以便采用样式来保持统一的外观。

如果要旋转 **Label** 组件，那么必须嵌入字体，否则在测试影片时这些字体不会显示。

Label 参数

可以在“属性”检查器或“组件”检查器中为每个 **Label** 组件实例设置以下创作参数：
autoSize、**condenseWhite**、**selectable**、**text** 和 **wordWrap**。其中每个参数都有对应的同名 **ActionScript** 属性。有关这些参数可能值的信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 **Label** 类。

创建具有 Label 组件的应用程序

以下过程解释了如何在创作时将 **Label** 组件添加到应用程序。在此示例中，标签仅显示文本“Expiration Date”。

创建具有 **Label** 组件的应用程序：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将一个 **Label** 组件从“组件”面板拖到舞台上，并在“属性”检查器中为该组件输入以下值：
 - 输入 **aLabel** 作为实例名称。
 - 输入 **80** 作为 W 值。
 - 输入 **100** 作为 X 值。
 - 输入 **100** 作为 Y 值。
 - 输入 **Expiration Date** 作为 **text** 参数。

3. 将一个 **TextArea** 组件拖到舞台上，并在“属性”检查器中为该组件输入以下值：

- 输入 **aTa** 作为实例名称。
- 输入 **22** 作为 **H** 值。
- 输入 **200** 作为 **X** 值。
- 输入 **100** 作为 **Y** 值。

4. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 **ActionScript** 代码：

```
var today:Date = new Date();
var expDate:Date = addDays(today, 14);
aTa.text = expDate.toString();

function addDays(date:Date, days:Number):Date {
    return addHours(date, days*24);
}

function addHours(date:Date, hrs:Number):Date {
    return addMinutes(date, hrs*60);
}

function addMinutes(date:Date, mins:Number):Date {
    return addSeconds(date, mins*60);
}

function addSeconds(date:Date, secs:Number):Date {
    var mSecs:Number = secs * 1000;
    var sum:Number = mSecs + date.getTime();
    return new Date(sum);
}
```

5. 选择“控制” > “测试影片”。

以下示例使用 **ActionScript** 创建一个 **Label** 组件。该示例使用 **Label** 来标识 **ColorPicker** 组件的功能，并使用 **htmlText** 属性将格式应用于 **Label** 的文本。

使用 **ActionScript** 创建 **Label** 组件实例：

1. 创建一个新的 **Flash** 文件 (**ActionScript 3.0**) 文档。
2. 将一个 **Label** 组件从“组件”面板拖到当前文档的“库”面板中。
3. 将一个 **ColorPicker** 组件从“组件”面板拖到当前文档的“库”面板中。
4. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 **ActionScript** 代码：

```
import fl.controls.Label;
import fl.controls.ColorPicker;

var aLabel:Label = new Label();
var aCp:ColorPicker = new ColorPicker();
```

```
addChild(aLabel);
addChild(aCp);

aLabel.htmlText = '<font face="Arial" color="#FF0000" size="14">Fill:</font>';
aLabel.x = 200;
aLabel.y = 150;
aLabel.width = 25;
aLabel.height = 22;

aCp.x = 230;
aCp.y = 150;
```

5. 选择“控制”>“测试影片”。

使用 List 组件

List 组件是一个可滚动的单选或多选列表框。列表还可显示图形及其它组件。在单击标签或数据参数字段时，会出现“值”对话框，您可以使用该对话框来添加显示在列表中的项。也可以使用 `List.addItem()` 和 `List.addItemAt()` 方法将项添加到列表中。

List 组件使用基于零的索引，其中索引为 **0** 的项即显示在顶端的项。当使用 **List** 类的方法和属性添加、删除或替换列表项时，可能需要指定该列表项的索引。

与 List 组件进行用户交互

您可以建立一个列表，以便用户可以选择一项或多项。例如，用户访问电子商务网站时需要选择想要购买的商品。一共有 **30** 种商品，用户在列表中上下滚动，并通过单击选择某一种。

您也可以设计一个列表，使用自定义影片剪辑作为行，这样就可以向用户显示更多信息。例如，在电子邮件应用程序中，每个信箱可能就是一个 **List** 组件，而每行可能会有指明优先级和状态的图标。

当 **List** 实例通过单击或 **Tab** 切换获得焦点时，可以使用以下按键来控制它：

键	说明
字母数字键	跳转到标签中以 <code>Key.getAscii()</code> 作为首字符的下一项。
Ctrl	允许选择和取消选择多个不相邻的项目的切换键。
向下键	将选区向下移动一项。
Home	将选区移到列表顶端。
Page Down	将选区向下移动一页。
Page Up	将选区向上移动一页。
Shift	允许进行连续选择。
向上键	将选择向上移动一项。

提醒

Page Up 键和 Page Down 键所使用的页大小比可以显示的项数少一项。例如，在一个十行的下拉列表中向下翻页，将会依次显示第 0-9 项、第 9-18 项、第 18-27 项等等，每页都会有一个重叠项。
另请注意，滚动单位为像素，而不是行。

有关控制焦点的详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 [IFocusManager](#) 接口和 [FocusManager](#) 类，以及第 48 页的“使用 [FocusManager](#)”。

每个 **List** 实例在舞台上的实时预览反映在创作过程中对属性检查器或组件检查器中的参数所做的更改。

在将 **List** 组件添加到应用程序时，可以添加以下几行 **ActionScript** 代码，以使其可由屏幕阅读器访问：

```
import fl.accessibility.ListAccImpl;

ListAccImpl.enableAccessibility();
```

无论组件有多少实例，都只对组件启用一次辅助功能。有关详细信息，请参阅《[使用 Flash](#)》中的第 18 章“创建具有辅助功能的内容”。

List 参数

可以在“属性”检查器或“组件”检查器中为每个 **List** 组件实例设置以下参数：

`allowMultipleSelection`、`dataProvider`、`horizontalLineScrollSize`、`horizontalPageScrollSize`、`horizontalScrollPolicy`、`multipleSelection`、`verticalLineScrollSize`、`verticalPageScrollSize` 和 `verticalScrollPolicy`。其中每个参数都有对应的同名 **ActionScript** 属性。有关这些参数的可能值的信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 **List** 类。有关使用 `dataProvider` 参数的信息，请参阅第 50 页的“使用 `dataProvider` 参数”。

创建具有 List 的应用程序

下面的示例说明了如何在创作时将 **List** 组件添加到应用程序。此示例中的 **List** 由标识汽车型号的标签和包含价格的数据字段构成。

将单个 **List** 组件添加到应用程序：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将一个 **List** 组件从“组件”面板拖到舞台上。
3. 在“属性”检查器中，执行以下操作：
 - 输入实例名称 **aList**。
 - 将值 **200** 分配给 **W**（宽）。
4. 使用“文本”工具在 aList 下方创建一个文本字段，并为它指定实例名称 **aTf**。
5. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 ActionScript 代码：

```
import fl.controls.List;
import flash.text.TextField;

aTf.type = TextFieldType.DYNAMIC;
aTf.border = false;

// Create these items in the Property inspector when data and label
// parameters are available.
aList.addItem({label:"1956 Chevy (Cherry Red)", data:35000});
aList.addItem({label:"1966 Mustang (Classic)", data:27000});
aList.addItem({label:"1976 Volvo (Xcllnt Cond)", data:17000});
aList.allowMultipleSelection = true;

aList.addEventListener(Event.CHANGE, showData);

function showData(event:Event) {
    aTf.text = "This car is priced at: $" + event.target.selectedItem.data;
}
```

此代码使用 addItem() 方法，用三个项来填充 aList，为每一项分配一个 label 值（该值将出现在列表中）和一个 data 值。当您在列表中选择某一项时，事件侦听器将调用 showData() 函数，该函数会显示所选项的 data 值。

6. 选择“控制”>“测试影片”来编译并运行此应用程序。

下面的示例还创建一个由汽车型号及其价格组成的 **List**。不过，该示例使用数据提供者，而不是 addItem() 方法来填充此 **List**。

使用数据提供者填充 List 实例:

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将一个 List 组件从“组件”面板拖到舞台上。
3. 在“属性”检查器中, 执行以下操作:
 - 输入实例名称 **aList**。
 - 将值 **200** 分配给 **W** (宽)。
4. 使用“文本”工具在 aList 下方创建一个文本字段, 并为它指定实例名称 **aTf**。
5. 打开“动作”面板, 在主时间轴中选择第 1 帧, 然后输入以下 ActionScript 代码:

```
import fl.controls.List;
import fl.data.DataProvider;
import flash.text.TextField;

aTf.type = TextFieldType.DYNAMIC;
aTf.border = false;

var cars:Array = [
    {label:"1956 Chevy (Cherry Red)", data:35000},
    {label:"1966 Mustang (Classic)", data:27000},
    {label:"1976 Volvo (Xcllnt Cond)", data:17000},
];
aList.dataProvider = new DataProvider(cars);
aList.allowMultipleSelection = true;

aList.addEventListener(Event.CHANGE, showData);

function showData(event:Event) {
    aTf.text = "This car is priced at: $" + event.target.selectedItem.data;
}
```

6. 选择“控制” > “测试影片”, 以查看此 List 及其项。

下面的示例创建一个由颜色名称组成的 List, 当您选择某种颜色时, 它将该颜色应用于影片剪辑。

使用 List 组件控制 MovieClip 实例:

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将一个 List 组件从“组件”面板拖到舞台上, 并在“属性”检查器中为该组件指定以下值:
 - 输入 **aList** 作为实例名称。
 - 输入 **60** 作为 H 值。
 - 输入 **100** 作为 X 值。
 - 输入 **150** 作为 Y 值。

3. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 **ActionScript** 代码：

```
aList.addItem({label:"Blue", data:0x0000CC});
aList.addItem({label:"Green", data:0x00CC00});
aList.addItem({label:"Yellow", data:0xFFFF00});
aList.addItem({label:"Orange", data:0xFF6600});
aList.addItem({label:"Black", data:0x000000});

var aBox:MovieClip = new MovieClip();
addChild(aBox);

aList.addEventListener(Event.CHANGE, changeHandler);
function changeHandler(event:Event) {
    drawBox(aBox, event.target.selectedItem.data);
};

function drawBox(box:MovieClip,color:uint):void {
    box.graphics.beginFill(color, 1.0);
    box.graphics.drawRect(225, 150, 100, 100);
    box.graphics.endFill();
}
```

4. 选择“控制” > “测试影片”，运行应用程序。
5. 单击此 **List** 中的颜色，可以看到颜色会显示在影片剪辑中。

下面的示例使用 **ActionScript** 创建一个简单列表，并使用 `addItem()` 方法填充该列表。

使用 **ActionScript** 创建 **List** 组件实例：

1. 创建一个新的 **Flash** 文件 (**ActionScript 3.0**) 文档。
2. 将一个 **List** 组件从“组件”面板拖到“库”面板中。
3. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 **ActionScript** 代码：

```
import fl.controls.List;

var aList:List = new List();
aList.addItem({label:"One", data:1});
aList.addItem({label:"Two", data:2});
aList.addItem({label:"Three", data:3});
aList.addItem({label:"Four", data:4});
aList.addItem({label:"Five", data:5});
aList.setSize(60, 40);
aList.move(200,200);
addChild(aList);
aList.addEventListener(Event.CHANGE, changeHandler);
function changeHandler(event:Event):void {
    trace(event.target.selectedItem.data);
}
```

4. 选择“控制” > “测试影片”，运行应用程序。

使用 NumericStepper

NumericStepper 组件允许用户逐个通过一组经过排序的数字。该组件由显示在小向上箭头和向下箭头按钮旁边的文本框中的数字组成。用户按下按钮时，数字将按 `stepSize` 参数中指定的单位递增或递减，直到用户释放按钮或达到最大或最小值为止。**NumericStepper** 组件的文本框中的文本也是可编辑的。

每个 **NumericStepper** 实例的实时预览反映了“属性”检查器或“组件”检查器中 `value` 参数的设置。但是，在实时预览中，无法通过鼠标或键盘与 **NumericStepper** 的箭头按钮进行交互。

用户与 NumericStepper 的交互

NumericStepper 组件可用于任何您想让用户选择数值的场合。例如，可以在表单中使用 **NumericStepper** 组件来设置信用卡到期日期的月、日和年。还可以使用 **NumericStepper** 组件来允许用户改变字体大小。

NumericStepper 组件只处理数值数据。此外，要显示两个以上的数值位置（如数字 5246 或 1.34），在创作时必须调整步进器的大小。

可以在应用程序中启用或禁用 **NumericStepper**。在禁用状态下，**NumericStepper** 不接收鼠标或键盘输入。当 **NumericStepper** 处于启用状态时，如果单击它或通过 **Tab** 切换到它，则它将获得焦点，且其内部焦点会设置为文本框。当 **NumericStepper** 实例具有焦点时，可以使用以下键来控制它：

键	说明
向下键	值一次变化一个单位。
向左键	在文本框中将插入点向左移动。
向右键	在文本框中将插入点向右移动。
Shift+Tab	将焦点移到上一个对象。
Tab	将焦点移到下一个对象。
向上键	值一次变化一个单位。

有关控制焦点的详细信息，请参阅 [《ActionScript 3.0 语言和组件参考》](#) 中的 **FocusManager** 类以及第 48 页的“使用 **FocusManager**”。

NumericStepper 参数

可以在“属性”检查器或“组件”检查器中为每个 [NumericStepper](#) 实例设置以下参数：`maximum`、`minimum`、`stepSize` 和 `value`。其中每个参数都有对应的同名 [ActionScript](#) 属性。有关这些参数的可能值的信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 [NumericStepper](#) 类。

创建具有 NumericStepper 的应用程序

下面的过程解释了如何在创作时将 [NumericStepper](#) 组件添加到应用程序。该示例将一个 [NumericStepper](#) 组件和一个 [Label](#) 组件放置到舞台上，然后在 [NumericStepper](#) 实例上创建一个 `Event.CHANGE` 事件的侦听器。当数字步进器中的值更改时，该示例会在 [Label](#) 实例的 `text` 属性中显示新值。

创建具有 NumericStepper 组件的应用程序：

1. 将一个 [NumericStepper](#) 组件从“组件”面板拖到舞台上。
2. 在“属性”检查器中，输入实例名称“**aNs**”。
3. 将一个 [Label](#) 组件从“组件”面板拖到舞台上。
4. 在“属性”检查器中，输入实例名称“**aLabel**”。
5. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 [ActionScript](#) 代码：

```
import flash.events.Event;

aLabel.text = "value = " + aNs.value;

aNs.addEventListener(Event.CHANGE, changeHandler);
function changeHandler(event:Event):void {
    aLabel.text = "value = " + event.target.value;
};
```

此示例将标签的 `text` 属性设置为 [NumericStepper](#) 的值。只要 [NumericStepper](#) 实例中的值发生变化，`changeHandler()` 函数就会更新标签的 `text` 属性。

6. 选择“控制” > “测试影片”。

下面的示例使用 [ActionScript](#) 代码创建三个 [NumericStepper](#)，分别用于输入用户出生日期的月、日和年。该示例还为每个 [NumericStepper](#) 添加了用作提示和标识符的 [Label](#)。

使用 ActionScript 创建 NumericStepper:

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将一个 Label 拖到 “库” 面板中。
3. 将一个 NumericStepper 组件拖到 “库” 面板中。
4. 打开 “动作” 面板，在主时间轴中选择第 1 帧，然后输入以下 ActionScript 代码：

```
import fl.controls.Label;
import fl.controls.NumericStepper;

var dobPrompt:Label = new Label();
var moPrompt:Label = new Label();
var dayPrompt:Label = new Label();
var yrPrompt:Label = new Label();

var moNs:NumericStepper = new NumericStepper();
var dayNs:NumericStepper = new NumericStepper();
var yrNs:NumericStepper = new NumericStepper();

addChild(dobPrompt);
addChild(moPrompt);
addChild(dayPrompt);
addChild(yrPrompt);
addChild(moNs);
addChild(dayNs);
addChild(yrNs);

dobPrompt.setSize(65, 22);
dobPrompt.text = "Date of birth:"
dobPrompt.move(80, 150);

moNs.move(150, 150);
moNs.setSize(40, 22);
moNs.minimum = 1;
moNs.maximum = 12;
moNs.stepSize = 1;
moNs.value = 1;

moPrompt.setSize(25, 22);
moPrompt.text = "Mo.";
moPrompt.move(195, 150);

dayNs.move(225, 150);
dayNs.setSize(40, 22);
dayNs.minimum = 1;
dayNs.maximum = 31;
dayNs.stepSize = 1;
dayNs.value = 1;
```

```

dayPrompt.setSize(25, 22);
dayPrompt.text = "Day";
dayPrompt.move(270, 150);

yrNs.move(300, 150);
yrNs.setSize(55, 22);
yrNs.minimum = 1900;
yrNs.maximum = 2006;
yrNs.stepSize = 1;
yrNs.value = 1980;

yrPrompt.setSize(30, 22);
yrPrompt.text = "Year";
yrPrompt.move(360, 150);

```

5. 选择“控制” > “测试影片”，运行应用程序。

使用 ProgressBar

ProgressBar 组件用于显示内容的加载进度，当内容较大且可能延迟应用程序的执行时，显示进度可令用户安心。**ProgressBar** 对于显示图像和部分应用程序的加载进度非常有用。加载进程可能是确定的，也可能是不确定的。当要加载的内容量为已知时，使用“确定的”进度栏。确定的进度栏是一段时间内任务进度的线性表示。当要加载的内容量为未知时，使用“不确定的”进度栏。还可以添加 **Label** 组件，以将加载进度显示为百分比。

ProgressBar 组件使用 9 切片缩放，并具有条形外观、轨道外观和不确定外观。

用户与 ProgressBar 的交互

可以采用三种模式来使用 **ProgressBar** 组件。最常用的模式是事件模式和轮询模式。这两种模式指定一个发出 `progress` 和 `complete` 事件（事件模式和轮询模式）或公开 `bytesLoaded` 和 `bytesTotal` 属性（轮询模式）的加载进程。还可以在手动模式下使用 **ProgressBar** 组件，方法是：设置 `maximum`、`minimum` 和 `value` 属性，并调用 `ProgressBar.setProgress()` 方法。可以设置不确定的属性，以指示 **ProgressBar** 是用条纹图案填充并且源的大小未知 (`true`)，还是实心填充并且源的大小已知 (`false`)。

ProgressBar 的模式是通过设置其 `mode` 属性设置的，具体方法为：在“属性”检查器或“组件”检查器中设置 `mode` 参数，或使用 **ActionScript**。

使用 **ProgressBar** 显示处理状态（如正在分析 100,000 项）时，如果它处于单个帧循环中，则屏幕将不会重绘，因而看不到进度栏的更新。

ProgressBar 参数

可以在“属性”检查器或“组件”检查器中为每个 [ProgressBar](#) 实例设置以下参数：

`direction`、`mode` 和 `source`。其中每个参数都有对应的同名 `ActionScript` 属性。

可以通过编写 `ActionScript` 代码使用 `ProgressBar` 组件的属性、方法和事件来控制它的这些选项和其它选项。有关详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 `ProgressBar` 类。

创建具有 ProgressBar 的应用程序

下面的过程展示了如何在创作时将 `ProgressBar` 组件添加到应用程序。在此示例中，此 `ProgressBar` 使用事件模式。在事件模式下，加载的内容发出 `progress` 和 `complete` 事件，`ProgressBar` 调度这两个事件来显示进度。当发生 `progress` 事件时，该示例将更新标签以显示已加载内容的百分比。当发生 `complete` 事件时，该示例将显示“Loading complete”以及 `bytesTotal` 属性的值，该属性值为文件的大小。

在事件模式下创建具有 `ProgressBar` 组件的应用程序：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将 `ProgressBar` 组件从“组件”面板拖到舞台上。
 - 在“属性”检查器中，输入实例名称“**aPb**”。
 - 在“参数”部分中，输入 **200** 作为 X 值。
 - 输入 **260** 作为 Y 值。
 - 选择 `event` 作为 `mode` 参数。
3. 将 `Button` 组件从“组件”面板拖到舞台上。
 - 在“属性”检查器中，输入“**loadButton**”作为实例名称。
 - 输入 **220** 作为 X 参数。
 - 输入 **290** 作为 Y 参数。
 - 输入 **Load Sound** 作为 `label` 参数。
4. 将一个 `Label` 组件拖到舞台上，然后为其指定实例名称 **progLabel**。
 - 输入 **150** 作为 W 值。
 - 输入 **200** 作为 X 参数。
 - 输入 **230** 作为 Y 参数。
 - 在“参数”部分中，清除 `text` 参数的值。

5. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 **ActionScript** 代码，此代码用于加载一个 **mp3** 音频文件：

```
import fl.controls.ProgressBar;
import flash.events.ProgressEvent;
import flash.events.IOErrorEvent;

var aSound:Sound = new Sound();
aPb.source = aSound;
var url:String = "http://www.helpexamples.com/flash/sound/song1.mp3";
var request:URLRequest = new URLRequest(url);

aPb.addEventListener(ProgressEvent.PROGRESS, progressHandler);
aPb.addEventListener(Event.COMPLETE, completeHandler);
aSound.addEventListener(IOErrorEvent.IO_ERROR, ioErrorHandler);
loadButton.addEventListener(MouseEvent.CLICK, clickHandler);

function progressHandler(event:ProgressEvent):void {
    progLabel.text = ("Sound loading ... " + aPb.percentComplete);
}

function completeHandler(event:Event):void {
    trace("Loading complete");
    trace("Size of file: " + aSound.bytesTotal);
    aSound.close();
    loadButton.enabled = false;
}

function clickHandler(event:MouseEvent) {
    aSound.load(request);
}

function ioErrorHandler(event:IOErrorEvent):void {
    trace("Load failed due to: " + event.text);
}
```

6. 选择“控制” > “测试影片”。

下面的示例将 **ProgressBar** 设置为轮询模式。在轮询模式下，进度的确定方式为：侦听正在加载的内容上的 **progress** 事件并使用其 **bytesLoaded** 和 **bytesTotal** 属性计算进度。此示例加载一个 **Sound** 对象，侦听其 **progress** 事件，并使用其 **bytesLoaded** 和 **bytesTotal** 属性计算加载百分比。它将加载百分比同时显示在标签和“输出”面板中。

在轮询模式下创建具有 **ProgressBar** 组件的应用程序：

1. 创建一个新的 **Flash** 文件 (ActionScript 3.0) 文档。
2. 将一个 **ProgressBar** 组件从“组件”面板拖到舞台上，并在“属性”检查器中输入以下值：
 - 输入 **aPb** 作为实例名称。
 - 输入 **185** 作为 X 值。
 - 输入 **225** 作为 Y 值。
3. 将一个 **Label** 组件拖到舞台上，并在“属性”检查器中输入以下值：
 - 输入 **progLabel** 作为实例名称。
 - 输入 **180** 作为 X 值。
 - 输入 **180** 作为 Y 值。
 - 在“参数”部分中，清除 **text** 参数的值。

4. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 **ActionScript** 代码，此代码创建一个 **Sound** 对象 (aSound) 并调用 **loadSound()** 将声音加载到此 **Sound** 对象中：

```
import fl.controls.ProgressBarMode;
import flash.events.ProgressEvent;
import flash.media.Sound;

var aSound:Sound = new Sound();
var url:String = "http://www.helpexamples.com/flash/sound/song1.mp3";
var request:URLRequest = new URLRequest(url);

aPb.mode = ProgressBarMode.POLLED;
aPb.source = aSound;
aSound.addEventListener(ProgressEvent.PROGRESS, loadListener);

aSound.load(request);

function loadListener(event:ProgressEvent) {
    var percentLoaded:int = event.target.bytesLoaded /
        event.target.bytesTotal * 100;
    progLabel.text = "Percent loaded: " + percentLoaded + "%";
    trace("Percent loaded: " + percentLoaded + "%");
}
```

5. 选择“控制” > “测试影片”，运行应用程序。

下面的示例将 **ProgressBar** 设置为手动模式。在手动模式下，必须通过调用 **setProgress()** 方法来手动设置进度，并为其提供当前值和最大值来确定进度。在手动模式中不需要设置 **source** 属性。该示例使用一个最大值为 **250** 的 **NumericStepper** 组件来逐渐增加 **ProgressBar** 中的进度。当 **NumericStepper** 中的值发生变化，从而触发 **CHANGE** 事件时，事件处理函数 (**nsChangeHandler**) 将调用 **setProgress()** 方法来推进进度栏。它还显示已完成进度相对于最大值的百分比。

在手动模式下创建具有 ProgressBar 组件的应用程序：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将 ProgressBar 组件从“组件”面板拖到舞台上，并在“属性”检查器中为其指定以下值：
 - 输入 **aPb** 作为实例名称。
 - 输入 **180** 作为 X 值。
 - 输入 **175** 作为 Y 值。
3. 将一个 NumericStepper 组件拖到舞台上，并在“属性”检查器中输入以下值：
 - 输入 **aNs** 作为实例名称。
 - 输入 **220** 作为 X 值。
 - 输入 **215** 作为 Y 值。
 - 在“参数”部分中，输入 **250** 作为 maximum 参数，输入 **0** 作为 minimum 值，输入 **1** 作为 stepSize 参数，输入 **0** 作为 value 参数。
4. 将一个 Label 组件拖到舞台上，并在“属性”检查器中输入以下值：
 - 输入 **progLabel** 作为实例名称。
 - 输入 **150** 作为 W 值。
 - 输入 **180** 作为 X 值。
 - 输入 **120** 作为 Y 值。
 - 在“参数”选项卡中，清除 text 参数的值 Label。
5. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下代码：

```
import fl.controls.ProgressBarDirection;
import fl.controls.ProgressBarMode;
import flash.events.Event;

aPb.direction = ProgressBarDirection.RIGHT;
aPb.mode = ProgressBarMode.MANUAL;
aPb.minimum = aNs.minimum;
aPb.maximum = aNs.maximum;
aPb.indeterminate = false;

aNs.addEventListener(Event.CHANGE, nsChangeHandler);

function nsChangeHandler(event:Event):void {
    aPb.value = aNs.value;
    aPb.setProgress(aPb.value, aPb.maximum);
    progLabel.text = "Percent of progress = " + int(aPb.percentComplete) +
"%";
}
```

6. 选择“控制” > “测试影片”，运行应用程序。
7. 单击 **NumericStepper** 上的向上箭头，以推进进度栏。

下面的示例使用 **ActionScript** 创建一个 **ProgressBar**。除此之外，它还重复上一示例的功能，即在手动模式下创建 **ProgressBar**。

使用 **ActionScript** 创建 **ProgressBar**:

1. 创建一个新的 **Flash** 文件 (**ActionScript 3.0**) 文档。
2. 将 **ProgressBar** 组件拖到“库”面板中。
3. 将 **NumericStepper** 组件拖到“库”面板中。
4. 将 **Label** 组件拖到“库”面板中。
5. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下代码：

```
import fl.controls.ProgressBar;
import fl.controls.NumericStepper;
import fl.controls.Label;
import fl.controls.ProgressBarDirection;
import fl.controls.ProgressBarMode;
import flash.events.Event;

var aPb:ProgressBar = new ProgressBar();
var aNs:NumericStepper = new NumericStepper();
var progLabel:Label = new Label();

addChild(aPb);
addChild(aNs);
addChild(progLabel);

aPb.move(180,175);
aPb.direction = ProgressBarDirection.RIGHT;
aPb.mode = ProgressBarMode.MANUAL;

progLabel.setSize(150, 22);
progLabel.move(180, 150);
progLabel.text = "";

aNs.move(220, 215);
aNs.maximum = 250;
aNs.minimum = 0;
aNs.stepSize = 1;
aNs.value = 0;

aNs.addEventListener(Event.CHANGE, nsChangeHandler);
```

```
function nsChangeHandler(event:Event):void {
    aPb.setProgress(aNs.value, aNs.maximum);
    progLabel.text = "Percent of progress = " + int(aPb.percentComplete) +
        "%";
}
```

- 6. 选择 “控制” > “测试影片”，运行应用程序。
- 7. 单击 `NumericStepper` 上的向上箭头，以推进进度栏。

使用 RadioButton

使用 `RadioButton` 组件可以强制用户只能选择一组选项中的一项。该组件必须用于至少有两个 `RadioButton` 实例的组中。在任意给定时刻，都只能有一个组成员被选中。选择组中的一个单选按钮将取消选择该组中当前选中的单选按钮。可以设置 `groupName` 参数来指示单选按钮属于哪个组。

单选按钮是 Web 上许多表单应用程序的基础部分。只要您需要让用户从一组选项选择一个，您就可以使用单选按钮。例如，在表单上询问客户要使用哪种信用卡时，就可以使用单选按钮。

用户与 RadioButton 的交互

可以启用或禁用单选按钮。禁用的单选按钮不接收鼠标或键盘输入。当用户单击或使用 `Tab` 切换到 `RadioButton` 组件组时，只有选中的单选按钮才会接收焦点。然后用户就可以使用以下按键来控制它：

键	说明
向上箭头 / 向左箭头	将选择移到单选按钮组内的上一个单选按钮。
向下箭头 / 向右箭头	将选择移到单选按钮组内的下一个单选按钮。
Tab	将焦点从单选按钮组移到下一个组件。

有关控制焦点的详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 `IFocusManager` 接口和 `FocusManager` 类，以及第 48 页的“使用 `FocusManager`”。

每个 `RadioButton` 实例在舞台上的实时预览反映在创作过程中对“属性”检查器或“组件”检查器中的参数所做的更改。但是，实时预览不能体现选择的互斥性。如果将同组的两个单选按钮的 `selected` 参数都设置为 `true`，则它们都会显示为选中状态，但是在运行时只有最后创建的实例才显示为选中状态。有关详细信息，请参阅第 105 页的“[RadioButton 参数](#)”。

在将 **RadioButton** 组件添加到应用程序时，可以添加以下几行代码，以使其可由屏幕阅读器访问：

```
import fl.accessibility.RadioButtonAccImpl;  
RadioButtonAccImpl.enableAccessibility();
```

无论组件有多少实例，都只对组件启用一次辅助功能。有关详细信息，请参阅《使用 Flash》中的第 18 章“创建具有辅助功能的内容”。

RadioButton 参数

可以在“属性”检查器或“组件”检查器中为每个 **RadioButton** 组件实例设置以下创作参数：groupName、label、LabelPlacement、selected 和 value。其中每个参数都有对应的同名 **ActionScript** 属性。有关这些参数的可能值的信息，请参阅《**ActionScript 3.0 语言和组件参考**》中的 **RadioButton** 类。

可以编写 **ActionScript** 代码，以使用 **RadioButton** 类的方法、属性和事件来设置 **RadioButton** 实例的其它选项。

创建具有 RadioButton 的应用程序

以下过程解释了如何在创作时将 **RadioButton** 组件添加到应用程序。在此示例中，使用 **RadioButton** 表示是非问题。**RadioButton** 的数据显示在 **TextArea** 中。

创建具有 RadioButton 组件的应用程序：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将两个 **RadioButton** 组件从“组件”面板拖到舞台上。
3. 选择第一个单选按钮。在“属性”检查器中，为其指定实例名称 **yesRb** 和组名称 **rbGroup**。
4. 选择第二个单选按钮。在“属性”检查器中，为其指定实例名称 **noRb** 和组名称 **rbGroup**。
5. 将一个 **TextArea** 组件从“组件”面板拖到舞台上，并为其指定实例名称 **aTa**。
6. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 **ActionScript** 代码：

```
yesRb.label = "Yes";  
yesRb.value = "For";  
noRb.label = "No";  
noRb.value = "Against";  
  
yesRb.move(50, 100);  
noRb.move(100, 100);  
aTa.move(50, 30);  
noRb.addEventListener(MouseEvent.CLICK, clickHandler);
```

```
yesRb.addEventListener(MouseEvent.CLICK, clickHandler);

function clickHandler(event:MouseEvent):void {
    aTa.text = event.target.value;
}
```

7. 选择“控制” > “测试影片”，运行应用程序。

下面的示例使用 **ActionScript** 创建三个 **RadioButton**，分别表示红色、蓝色和绿色，并绘制一个灰色的框。每个 **RadioButton** 的 `value` 属性指定与此按钮关联的颜色的十六进制值。当用户单击其中一个 **RadioButton** 时，`clickHandler()` 函数将调用 `drawBox()`，同时传递此 **RadioButton** 的 `value` 属性中的颜色值来为此框着色。

使用 ActionScript 创建 RadioButton:

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将一个 **RadioButton** 组件拖到“库”面板中。
3. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 **ActionScript** 代码：

```
import fl.controls.RadioButton;
import fl.controls.RadioButtonGroup;

var redRb:RadioButton = new RadioButton();
var blueRb:RadioButton = new RadioButton();
var greenRb:RadioButton = new RadioButton();
var rbGrp:RadioButtonGroup = new RadioButtonGroup("colorGrp");

var aBox:MovieClip = new MovieClip();
drawBox(aBox, 0xCCCCCC);

addChild(redRb);
addChild(blueRb);
addChild(greenRb);
addChild(aBox);

redRb.label = "Red";
redRb.value = 0xFF0000;
blueRb.label = "Blue";
blueRb.value = 0x0000FF;
greenRb.label = "Green";
greenRb.value = 0x00FF00;
redRb.group = blueRb.group = greenRb.group = rbGrp;
redRb.move(100, 260);
blueRb.move(150, 260);
greenRb.move(200, 260);

rbGrp.addEventListener(MouseEvent.CLICK, clickHandler);
```

```

function clickHandler(event:MouseEvent):void {
    drawBox(aBox, event.target.selection.value);
}

function drawBox(box:MovieClip,color:uint):void {
    box.graphics.beginFill(color, 1.0);
    box.graphics.drawRect(125, 150, 100, 100);
    box.graphics.endFill();
}

```

4. 选择“控制” > “测试影片”，运行应用程序。

有关详细信息，请参阅 [《ActionScript 3.0 语言和组件参考》](#) 中的 `RadioButton` 类。

使用 ScrollPane 组件

如果某些内容对于它们要加载到其中的区域而言过大，则可以使用 `ScrollPane` 组件来显示这些内容。例如，如果您有一幅大图像，而在应用程序中只有很小的空间来显示它，则可以将它加载到 `ScrollPane` 中。`ScrollPane` 可以接受影片剪辑、JPEG、PNG、GIF 和 SWF 文件。

像 `ScrollPane` 和 `UILoader` 这样的组件具有 `complete` 事件，使用此事件可以确定内容何时完成加载。如果要对 `ScrollPane` 或 `UILoader` 组件的内容设置属性，可侦听 `complete` 事件，并在事件处理函数中设置属性。例如，下面的代码创建一个 `Event.COMPLETE` 事件的侦听器和一个将 `ScrollPane` 内容的 `alpha` 属性设置为 `.5` 的事件处理函数：

```

function spComplete(event:Event):void{
    aSp.content.alpha = .5;
}
aSp.addEventListener(Event.COMPLETE, spComplete);

```

如果在将内容加载到 `ScrollPane` 时指定一个位置，则必须将该位置（X 和 Y 坐标）指定为 `0,0`。例如，以下代码可以正确加载 `ScrollPane`，原因是框在位置 `0,0` 处绘制：

```

var box:MovieClip = new MovieClip();
box.graphics.beginFill(0xFF0000, 1);
box.graphics.drawRect(0, 0, 150, 300);
box.graphics.endFill();
aSp.source = box;//load ScrollPane

```

有关详细信息，请参阅 [《ActionScript 3.0 语言和组件参考》](#) 中的 `ScrollPane` 类。

用户与 ScrollPane 的交互

可以启用或禁用 ScrollPane。禁用的 ScrollPane 不接收鼠标或键盘输入。当 ScrollPane 具有焦点时，用户可以使用以下按键来控制它：

键	说明
向下键	将内容向上移动一垂直滚动行。
向上键	将内容向下移动一垂直滚动行。
End	将内容移到 ScrollPane 的底部。
向左键	将内容向右移动一水平滚动行。
向右键	将内容向左移动一水平滚动行。
Home	将内容移到 ScrollPane 的顶部。
End	将内容移到 ScrollPane 的底部。
Page Down	将内容向上移动一垂直滚动页。
Page Up	将内容向下移动一垂直滚动页。

用户可以使用鼠标在 ScrollPane 的内容以及垂直和水平滚动条上与 ScrollPane 进行交互。如果 scrollDrag 属性设置为 true，则用户可以使用鼠标拖动内容。如果内容上出现手形指针，则表明用户可以拖动此内容。与其它大多数控件不同，此时动作发生于按下鼠标按键时，并将持续到松开鼠标按键时。如果内容中包含有效的 Tab 停靠位，则必须将 scrollDrag 设置为 false。否则每次在内容上单击鼠标时都会调用滚动拖动。

ScrollPane 参数

可以在“属性”检查器或“组件”检查器中为每个 ScrollPane 实例设置以下参数：horizontalLineScrollSize、horizontalPageScrollSize、horizontalScrollPolicy、scrollDrag、source、verticalLineScrollSize、verticalPageScrollSize 和 verticalScrollPolicy。其中每个参数都有对应的同名 **ActionScript** 属性。有关这些参数的可能值的信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 ScrollPane 类。

可以通过编写 **ActionScript** 使用 ScrollPane 组件的属性、方法和事件来控制它的这些选项和其它选项。

创建具有 ScrollPane 的应用程序

以下过程解释了如何在创作时将 **ScrollPane** 组件添加到应用程序。在此示例中，**ScrollPane** 从 `source` 属性指定的路径加载一张图片。

创建具有 ScrollPane 组件的应用程序：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将 **ScrollPane** 组件从“组件”面板中拖到舞台上，并为其指定实例名称“**aSp**”。
3. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 ActionScript 代码：

```
import fl.events.ScrollEvent;

aSp.setSize(300, 200);

function scrollListener(event:ScrollEvent):void {
    trace("horizontalScPosition: " + aSp.horizontalScrollPosition +
        ", verticalScrollPosition = " + aSp.verticalScrollPosition);
};
aSp.addEventListener(ScrollEvent.SCROLL, scrollListener);

function completeListener(event:Event):void {
    trace(event.target.source + " has completed loading.");
};
// Add listener.
aSp.addEventListener(Event.COMPLETE, completeListener);

aSp.source = "http://www.helpexamples.com/flash/images/image1.jpg";
```

4. 选择“控制” > “测试影片”，运行应用程序。

该示例创建一个 **ScrollPane**、设置其大小并使用 `source` 属性向其中加载一个图像。它还创建两个侦听器。第一个侦听器在用户垂直或水平滚动时侦听 `scroll` 事件，并显示此图像的位置。第二个侦听器侦听 `complete` 事件，并在“输出”面板中显示说明图像已加载完成的消息。

下面的示例使用 ActionScript 创建一个 **ScrollPane**，并在其中放置一个宽 150 像素、高 300 像素的影片剪辑（红色的框）。

使用 ActionScript 创建 ScrollPane 实例：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将 **ScrollPane** 组件从“组件”面板拖到“库”面板中。
3. 将 **DataGrid** 组件从“组件”面板拖到“库”面板中。

4. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 **ActionScript** 代码：

```
import fl.containers.ScrollPane;
import fl.controls.ScrollPolicy;
import fl.controls.DataGrid;
import fl.data.DataProvider;

var aSp:ScrollPane = new ScrollPane();
var aBox:MovieClip = new MovieClip();
drawBox(aBox, 0xFF0000); //draw a red box

aSp.source = aBox;
aSp.setSize(150, 200);
aSp.move(100, 100);

addChild(aSp);

function drawBox(box:MovieClip,color:uint):void {
    box.graphics.beginFill(color, 1);
    box.graphics.drawRect(0, 0, 150, 300);
    box.graphics.endFill();
}
```

5. 选择“控制” > “测试影片”，运行应用程序。

使用 Slider

Slider 组件允许用户通过滑动与值范围相对应的轨道端点之间的图形“滑块”来选择值。例如，您可以使用滑块来让用户选择诸如数字或百分比之类的值。还可以使用 **ActionScript** 使滑块的值影响另一个对象的行为。例如，可以将滑块与图片关联，根据滑块的相对位置或值来缩小或放大图片。

Slider 的当前值由滑块在轨道端点之间或在此 **Slider** 的最小值与最大值之间的相对位置决定。

Slider 在其最小值与最大值之间的值范围可以是连续的，但您也可以通过设置 `snapInterval` 参数指定在最小值与最大值之间值与值之间的间隔。**Slider** 可以按照指定的间隔沿轨道显示刻度线，这些刻度线与分配给滑块的值无关。

默认情况下滑块为水平方向，但是，可以通过将 `direction` 参数的值设置为 **vertical**，将滑块指定为垂直方向。滑块轨道从一端延伸到另一端，刻度线从左至右紧贴在轨道上。

用户与 Slider 组件的交互

当 Slider 实例具有焦点时，可以使用以下按键来控制它：

键	说明
向右键	增大水平滑块的关联值。
向上键	增大垂直滑块的关联值。
向左键	减小水平滑块的关联值。
向下键	减小垂直滑块的关联值。
Shift+Tab	将焦点移到上一个对象。
Tab	将焦点移到下一个对象。

有关控制焦点的详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 [IFocusManager](#) 接口和 [FocusManager](#) 类，以及第 48 页的“使用 FocusManager”。

每个 Slider 实例的实时预览反映了创作期间在“属性”检查器或“组件”检查器中对参数所做的更改。

Slider 参数

可以在“属性”检查器或“组件”检查器中为每个 Slider 组件实例设置以下创作参数：`direction`、`liveDragging`、`maximum`、`minimum`、`snapInterval`、`tickInterval` 和 `value`。其中每个参数都有对应的同名 **ActionScript** 属性。有关这些参数的可能值的信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 Slider 类。

创建具有 Slider 的应用程序

下面的示例创建一个 Slider 实例，以允许用户表达他或她对某一假设事件的满意程度。用户可向右或向左移动滑块，以指示较高或较低的满意度。

创建具有 Slider 组件的应用程序：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将一个 Label 组件从“组件”面板拖到舞台中央。
 - 为其指定实例名称 **valueLabel**。
 - 将值 **0 percent** 赋给 `text` 参数。

3. 将一个 Slider 组件从“组件”面板拖到 value_lbl 下方中央位置。
 - 为其指定实例名称 **aSlider**。
 - 将其宽 (W:) 指定为 **200**。
 - 将其高 (H:) 指定为 **10**。
 - 将值 **100** 赋给 maximum 参数。
 - 将值 **10** 同时赋给 snapInterval 和 tickInterval 参数。
4. 将另一个 Label 实例从“库”面板中拖到 aSlider 下方中央位置。
 - 为其指定实例名称 **promptLabel**。
 - 将其宽 (W:) 指定为 **250**。
 - 将其高 (H:) 指定为 **22**。
 - 输入 **Please indicate your level of satisfaction** 作为 text 参数的值。
5. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 ActionScript 代码：

```
import fl.controls.Slider;
import fl.events.SliderEvent;
import fl.controls.Label;

aSlider.addEventListener(SliderEvent.CHANGE, changeHandler);

function changeHandler(event:SliderEvent):void {
    valueLabel.text = event.value + "percent";
}
```

6. 选择“控制” > “测试影片”。

在此示例中，当您将滑块从一个间隔移到另一个间隔时，SliderEvent.CHANGE 事件的侦听器将更新 valueLabel 的 text 属性，以显示与滑块位置相对应的百分比。

下面的示例使用 ActionScript 创建一个 Slider。该示例下载一幅花朵的图像并使用此 Slider 让用户减淡或加亮此图像，它是通过更改此图像的 alpha 属性使之与此 Slider 的值相对应实现的。

使用 ActionScript 创建具有 Slider 组件的应用程序：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将 Label 组件和 Slider 组件从“组件”面板拖到当前文档的“库”面板中。

此操作将组件添加到库中，但不会在应用程序中显示它们。
3. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下代码创建并放置组件实例：

```
import fl.controls.Slider;
import fl.events.SliderEvent;
import fl.controls.Label;
import fl.containers.UILoader;
```



```

var sliderLabel:Label = new Label();
sliderLabel.width = 120;
sliderLabel.text = "< Fade - Brighten >";
sliderLabel.move(170, 350);

var aSlider:Slider = new Slider();
aSlider.width = 200;
aSlider.snapInterval = 10;
aSlider.tickInterval = 10;
aSlider.maximum = 100;
aSlider.value = 100;
aSlider.move(120, 330);

var aLoader:UILoader = new UILoader();
aLoader.source = "http://www.flash-mx.com/images/imagel.jpg";
aLoader.scaleContent = false;

addChild(sliderLabel);
addChild(aSlider);
addChild(aLoader);

aLoader.addEventListener(Event.COMPLETE, completeHandler);

function completeHandler(event:Event) {
    trace("Number of bytes loaded: " + aLoader.bytesLoaded);
}

aSlider.addEventListener(SliderEvent.CHANGE, changeHandler);

function changeHandler(event:SliderEvent):void {
    aLoader.alpha = event.value * .01;
}

```

4. 选择“控制” > “测试影片”，运行应用程序。
5. 将滑块向左移动可以减淡图像；将滑块向右移动可以加亮图像。

使用 TextArea

TextArea 组件是本机 **ActionScript TextField** 对象的包装。可以使用 **TextArea** 组件来显示文本，如果 **editable** 属性为 **true**，也可以用它来编辑和接收文本输入。如果 **wordWrap** 属性设置为 **true**，则此组件可以显示或接收多行文本，并将较长的文本行换行。可以使用 **restrict** 属性限制用户能输入的字符，使用 **maxChars** 属性指定用户能输入的最大字符数。如果文本超出了文本区域的水平或垂直边界，则会自动出现水平和垂直滚动条，除非其关联的属性 **horizontalScrollPolicy** 和 **verticalScrollPolicy** 设置为 **off**。

在需要多行文本字段的任何地方都可使用 `TextArea` 组件。例如，可以在表单中使用 `TextArea` 组件作为注释字段。可以设置侦听器来检查当用户切换到该字段外时，该字段是否为空。该侦听器可以显示一条错误消息，指明必须在该字段中输入注释。

如果需要单行文本字段，请使用 `TextInput` 组件。

可以使用 `setStyle()` 方法来设置 `textFormat` 样式，以更改 `TextArea` 实例中所显示文本的样式。还可以在 `ActionScript` 中通过使用 `htmlText` 属性用 HTML 来设置 `TextArea` 组件的格式，并且可以将 `displayAsPassword` 属性设置为 `true`，以用星号遮蔽文本。如果将 `condenseWhite` 属性设置为 `true`，则 **Flash** 会删除新文本中由于空格、换行符等造成的多余空白。这对控件中已经存在的文本没有影响。

用户与 `TextArea` 的交互

在应用程序中可以启用或禁用 `TextArea` 组件。在禁用状态下，它不接收鼠标或键盘输入。当启用时，它遵循与 `ActionScript TextField` 对象相同的焦点、选择和导航规则。当 `TextArea` 实例具有焦点时，可以使用以下按键来控制它：

键	说明
箭头键	在文本内将插入点向上、向下、向左或向右移动（如果文本可编辑）。
Page Down	将插入点移到文本末尾（如果文本可编辑）。
Page Up	将插入点移到文本开头（如果文本可编辑）。
Shift+Tab	将焦点移到 Tab 键循环中的上一个对象。
Tab	将焦点移到 Tab 键循环中的下一个对象。

有关控制焦点的详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 `FocusManager` 类以及第 48 页的“使用 `FocusManager`”。

`TextArea` 参数

可以在“属性”检查器或“组件”检查器中为每个 `TextArea` 组件实例设置以下创作参数：`condenseWhite`、`editable`、`horizontalScrollPolicy`、`maxChars`、`restrict`、`text`、`verticalScrollPolicy` 和 `wordwrap`。其中每个参数都有对应的同名 `ActionScript` 属性。有关这些参数的可能值的信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 `TextArea` 类。

每个 `TextArea` 实例的实时预览反映在创作过程中对“属性”检查器或“组件”检查器中的参数所做的更改。如果需要滚动条，它会出现实时预览中，但并不起作用。在实时预览中，文本是不可选定的，并且无法在舞台上的组件实例中输入文本。

可以通过编写 `ActionScript` 代码使用 `TextArea` 组件的属性、方法和事件来控制它的这些选项和其它选项。有关详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 `TextArea` 类。

创建具有 TextArea 的应用程序

以下过程解释了如何在创作时将 **TextArea** 组件添加到应用程序。该示例在 **TextArea** 实例上设置了一个 `focusOut` 事件处理函数，用来验证用户在将焦点移到界面其它部分前是否在文本区域中键入了内容。

创建具有 TextArea 组件的应用程序：

1. 创建一个新的 Flash 文件 (ActionScript 3.0)。
2. 将一个 **TextArea** 组件从“组件”面板拖到舞台上，并为其指定实例名称 **aTa**。使其参数保留默认设置。
3. 将第二个 **TextArea** 组件从“组件”面板拖到舞台上，放在第一个组件的下方，并为其指定实例名称 **bTa**。使其参数保留默认设置。
4. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 ActionScript 代码：

```
import flash.events.FocusEvent;

aTa.restrict = "a-z, '\\\" \";
aTa.addEventListener(Event.CHANGE, changeHandler);
aTa.addEventListener(FocusEvent.KEY_FOCUS_CHANGE, k_m_fHandler);
aTa.addEventListener(FocusEvent.MOUSE_FOCUS_CHANGE, k_m_fHandler);

function changeHandler(ch_evt:Event):void {
    bTa.text = aTa.text;
}
function k_m_fHandler(kmf_event:FocusEvent):void {
    kmf_event.preventDefault();
}
```

此示例将允许在 aTa 文本区域中输入的字符限制为小写字符、逗号、撇号和空格。它还在 aTa 文本区域上设置了 `change`、`KEY_FOCUS_CHANGE` 和 `MOUSE_FOCUS_CHANGE` 事件的事件处理函数。`changeHandler()` 函数对每个 `change` 事件都将 `aTa.text` 赋给 `bTa.text`，从而使在 aTa 文本区域中输入的文本自动出现在 bTa 文本区域中。`KEY_FOCUS_CHANGE` 和 `MOUSE_FOCUS_CHANGE` 事件的 `k_m_fHandler()` 函数防止您不输入任何文本就按 **Tab** 键移到下一个字段。它通过防止默认行为来实现此目的。

5. 选择“控制” > “测试影片”。

如果不输入任何文本就按 **Tab** 键将焦点移到第二个文本区域，则会显示一条错误消息，且焦点应当回到第一个文本区域。当您在第一个文本区域中输入文本时，可以看到这些文本同时也出现在第二个文本区域中。

下面的示例使用 ActionScript 创建一个 **TextArea** 组件。它将 `condenseWhite` 属性 (property) 设置为 `true`，以压缩空白，并将文本赋给 `htmlText` 属性 (property)，以利用 HTML 文本的格式设置属性 (attribute)。

使用 ActionScript 创建 TextArea 实例:

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将 TextArea 组件拖到 “库” 面板中。
3. 打开 “动作” 面板，在主时间轴中选择第 1 帧，然后输入以下 ActionScript 代码:

```
import fl.controls.TextArea;

var aTa:TextArea = new TextArea();

aTa.move(100,100);
aTa.setSize(200, 200);
aTa.condenseWhite = true;
aTa.htmlText = '      <b>Lorem ipsum dolor</b> sit amet, consectetur
adipiscing elit. <u>Vivamus quis nisl vel tortor nonummy vulputate.</u>
Quisque sit amet eros sed purus euismod tempor. Morbi tempor. <font
color="#FF0000">Class aptent taciti sociosqu ad litora torquent per
conubia nostra, per inceptos hymenaeos.</font> Curabitur diam.
Suspendisse at purus in ipsum volutpat viverra. Nulla pellentesque
libero id libero.';
addChild(aTa);
```

此示例使用 htmlText 属性 (property) 将 HTML 粗体和下划线属性 (attribute) 应用于文本块，并将其显示在 a_ta 文本区域中。该示例还将 condenseWhite 属性设置为 true，以压缩文本块中的空白。setSize() 方法用于设置文本区域的高和宽，move() 方法用于设置文本区域的位置。addChild() 方法用于将 TextArea 实例添加到舞台上。

4. 选择 “控制” > “测试影片”。

使用 TextInput

TextInput 组件是单行文本组件，该组件是本机 ActionScript TextField 对象的包装。如果需要多行文本字段，请使用 TextArea 组件。例如，可以在表单中将 TextInput 组件用作密码字段。您还可以设置一个侦听器，检查用户在切换到字段之外时，该字段是否有足够的字符。该侦听器可以显示一条错误消息，指明必须输入正确数目的字符。

可以使用 setStyle() 方法来设置 textFormat 属性，以更改 TextInput 实例中所显示文本的样式。TextInput 组件还可以用 HTML 进行格式设置，或用作遮蔽文本的密码字段。

用户与 TextInput 的交互

在应用程序中，可以启用或禁用 `TextInput` 组件。在禁用状态下，它不接收鼠标或键盘输入。当启用时，它遵循与 `ActionScript TextField` 对象相同的焦点、选择和导航规则。当一个 `TextInput` 实例具有焦点时，还可以使用以下按键来控制它：

键	说明
箭头键	将插入点向左或向右移动一个字符。
Shift+Tab	将焦点移到上一个对象。
Tab	将焦点移到下一个对象。

有关控制焦点的详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 [FocusManager](#) 类以及第 48 页的“使用 [FocusManager](#)”。

每个 `TextInput` 实例的实时预览反映在创作过程中对“属性”检查器或“组件”检查器中的参数所做的更改。在实时预览中，文本是不可选定的，并且无法在舞台上的组件实例中输入文本。

在将 `TextInput` 组件添加到应用程序时，您可以使用“辅助功能”面板来使其可由屏幕阅读器访问。

TextInput 参数

可以在“属性”检查器或“组件”检查器中为每个 `TextInput` 组件实例设置以下创作参数：`editable`、`displayAsPassword`、`maxChars`、`restrict` 和 `text`。其中每个参数都有对应的同名 `ActionScript` 属性。有关这些参数的可能值的信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 `TextInput` 类。

您可以编写 `ActionScript`，以便使用 `TextInput` 组件的属性、方法和事件来控制该组件的这些选项和其它选项。有关详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 `TextInput` 类。

创建具有 TextInput 的应用程序

以下过程解释了如何将 `TextInput` 组件添加到应用程序。此示例使用两个 `TextInput` 字段来接收和确认密码。它使用一个事件侦听器来查看输入的字符数是否不少于八位以及这两个字段的文本是否匹配。

创建具有 TextInput 组件的应用程序：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将一个 Label 组件从“组件”面板拖到舞台上，并在“属性”检查器中为该组件输入以下值：
 - 输入实例名称 **pwdLabel**。
 - 为 W 输入值 **100**。
 - 为 X 输入值 **50**。
 - 为 Y 输入值 **150**。
 - 在“参数”部分中，为 text 参数输入值 **Password:**。
3. 从“组件”面板将第二个 Label 组件拖到舞台上并为它分配以下值：
 - 输入实例名称 **confirmLabel**。
 - 为 W 输入值 **100**。
 - 为 X 输入值 **50**。
 - 为 Y 输入值 **200**。
 - 在“参数”部分中，为 text 参数输入值 **Confirm Password:**。
4. 从“组件”面板将 TextInput 组件拖到舞台上并为它分配以下值：
 - 输入实例名称 **pwdTi**。
 - 为 W 输入值 **150**。
 - 为 X 输入值 **190**。
 - 为 Y 输入值 **150**。
 - 在“参数”部分中，双击 displayAsPassword 参数的值，然后选择“**true**”。这样，就会用星号掩盖文本字段中输入的值。
5. 从“组件”面板将第二个 TextInput 字段拖到舞台上并为它分配以下值：
 - 输入实例名称 **confirmTi**。
 - 为 W 输入值 **150**。
 - 为 X 输入值 **190**。
 - 为 Y 输入值 **200**。
 - 在“参数”部分中，双击 displayAsPassword 参数的值，然后选择“**true**”。这样，就会用星号掩盖文本字段中输入的值。

6. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 **ActionScript** 代码：

```
function tiListener(evt_obj:Event){
    if(confirmTi.text != pwdTi.text || confirmTi.length < 8)
    {
        trace("Password is incorrect. Please reenter it.");
    }
    else {
        trace("Your password is: " + confirmTi.text);
    }
}
confirmTi.addEventListener("enter", tiListener);
```

此代码在名为 `confirmTi` 的 **TextInput** 实例上设置 `enter` 事件处理函数。如果这两个密码不匹配或者用户键入的字符数少于八位，此示例将显示消息：“密码不正确。请重新输入。”如果密码为八位或八位以上并且互相匹配，此示例将在“输出”面板中显示输入的值。

7. 选择“控制” > “测试影片”。

下面的示例使用 **ActionScript** 创建一个 **TextInput** 组件。此示例还创建了一个用以提示用户输入其名字的 **Label**。此示例将此组件的 `restrict` 属性设置为只允许使用若干大写和小写字母、一个句点和一个空格。它还创建了一个用以设置 **Label** 和 **TextInput** 组件中文本格式的 **TextFormat** 对象。

使用 **ActionScript** 创建 **TextInput** 实例：

1. 创建一个新的 **Flash** 文件 (**ActionScript 3.0**) 文档。
2. 将 **TextInput** 组件从“组件”面板拖到“库”面板中。
3. 将 **Label** 组件从“组件”面板拖到“库”面板中。
4. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 **ActionScript** 代码：

```
import fl.controls.Label;
import fl.controls.TextInput;

var nameLabel:Label = new Label();
var nameTi:TextInput = new TextInput();
var tf:TextFormat = new TextFormat();

addChild(nameLabel);
addChild(nameTi);

nameTi.restrict = "A-Z .a-z";

tf.font = "Georgia";
tf.color = 0x0000CC;
tf.size = 16;
```

```
nameLabel.text = "Name: ";
nameLabel.setSize(50, 25);
nameLabel.move(100,100);
nameLabel.setStyle("textFormat", tf);
nameTi.move(160, 100);
nameTi.setSize(200, 25);
nameTi.setStyle("textFormat", tf);
```

5. 选择 “控制” > “测试影片”，运行应用程序。

使用 TileList

TileList 组件由一个列表组成，该列表由通过数据提供者提供数据的若干行和列组成。*项目* 是指在 **TileList** 中的单元格中存储的数据单元。项目源自数据提供者，通常有一个 `label` 属性和一个 `source` 属性。`label` 属性标识要在单元格中显示的内容，而 `source` 则为它提供值。

您可以创建一个 **Array** 实例，也可以从服务器检索一个。**TileList** 组件具有代理到其数据提供者的方法，如 `addItem()` 和 `removeItem()` 方法。如果没有为列表提供外部数据提供者，则这些方法会自动创建一个数据提供者实例，该实例通过 `List.dataProvider` 被公开。

与 TileList 的用户交互

TileList 用实现 `ICellRenderer` 接口的 **Sprite** 来呈现每个单元格。您可以使用 **TileList** 的 `cellRenderer` 属性指定此渲染器。**TileList** 组件的默认 **CellRenderer** 是显示一个图像（类、位图、实例或 **URL**）和一个可选标签的 **ImageCell**。此标签是始终与单元格底部对齐的单行。您只能沿一个方向滚动 **TileList**。

当 **TileList** 实例获得焦点时，您还可以使用以下键访问其内部的项目：

键	说明
向上箭头和向下箭头	用于在列中向上和向下移动。如果 <code>allowMultipleSelection</code> 属性为 <code>true</code> ，则您可以使用这些键与 Shift 键的组合来选择多个单元格。
向左箭头和向右箭头	用于在行中向左或向右移动。如果 <code>allowMultipleSelection</code> 属性为 <code>true</code> ，则您可以使用这些键与 Shift 键的组合来选择多个单元格。
Home	选择 TileList 中的第一个单元格。如果 <code>allowMultipleSelection</code> 属性为 <code>true</code> ，则按住 Shift 键并按 Home 键将会选择从您当前所选位置到第一个单元格的所有单元格。
End	选择 TileList 中的最后一个单元格。如果 <code>allowMultipleSelection</code> 属性为 <code>true</code> ，则按住 Shift 键并按 End 键将会选择从您当前所选位置到最后一个单元格的所有单元格。
Ctrl	如果 <code>allowMultipleSelection</code> 属性设置为 <code>true</code> ，则使用它可以不按特定顺序选择多个单元格。

向应用程序添加 **TileList** 组件时，可以通过添加以下 **ActionScript** 代码行使其可由屏幕读取器访问：

```
import fl.accessibility.TileListAccImpl;
```

```
TileListAccImpl.enableAccessibility();
```

无论组件有多少实例，都只对组件启用一次辅助功能。有关详细信息，请参阅《[使用 Flash](#)》中的第 18 章“[创建具有辅助功能的内容](#)”。

TileList 参数

可以在“属性”检查器或“组件”检查器中为每个 **TileList** 组件实例设置以下创作参数：`allowMultipleSelection`、`columnCount`、`columnWidth`、`dataProvider`、`direction`、`horizontalScrollLineSize`、`horizontalScrollPageSize`、`labels`、`rowCount`、`rowHeight` 和 `ScrollPolicy`、`verticalScrollLineSize`、`verticalScrollPageSize`。其中每个参数都有对应的同名 **ActionScript** 属性。有关使用 `dataProvider` 参数的信息，请参阅第 50 页的“[使用 dataProvider 参数](#)”。

您可以使用 **TileList** 实例的方法、属性和事件来编写 **ActionScript**，以为其设置其它选项。有关详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 **TileList** 类。

创建具有 TileList 的应用程序

本示例使用 **MovieClips** 填充具有颜料颜色数组的 **TileList**。

创建具有 TileList 组件的应用程序：

1. 创建一个新的 **Flash** 文件 (**ActionScript 3.0**) 文档。
2. 将一个 **TileList** 组件拖到舞台上，然后为其指定实例名称 **aTL**。
3. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 **ActionScript** 代码：

```
import fl.data.DataProvider;
import flash.display.DisplayObject;

var aBoxes:Array = new Array();
var i:uint = 0;
var colors:Array = new Array(0x000000, 0xFF0000, 0x0000CC, 0x00CC00,
    0xFFFF00);
var colorNames:Array = new Array("Midnight", "Cranberry", "Sky",
    "Forest", "July");
var dp:DataProvider = new DataProvider();
for(i=0; i < colors.length; i++) {
    aBoxes[i] = new MovieClip();
    drawBox(aBoxes[i], colors[i]); // draw box w next color in array
    dp.addItem( {label:colorNames[i], source:aBoxes[i]} );
}
```

```

aTl.dataProvider = dp;
aTl.columnWidth = 110;
aTl.rowHeight = 130;
aTl.setSize(280,150);
aTl.move(150, 150);
aTl.setStyle("contentPadding", 5);

function drawBox(box:MovieClip,color:uint):void {
    box.graphics.beginFill(color, 1.0);
    box.graphics.drawRect(0, 0, 100, 100);
    box.graphics.endFill();
}

```

4. 选择“控制” > “测试影片”对应用程序进行测试。

下一个示例动态创建 **TileList** 实例并向其添加 **ColorPicker**、**ComboBox**、**NumericStepper** 和 **CheckBox** 组件的实例。本示例创建一个包含标签和组件名称的 **Array**，然后显示 **Array** (dp) 并将其分配给 **TileList** 的 **dataProvider** 属性。本示例使用 **columnWidth** 和 **rowHeight** 属性以及 **setSize()** 方法来布置 **TileList**；使用 **move()** 方法来调整 **TileList** 在舞台上的位置；使用 **contentPadding** 样式在 **TileList** 实例的边框和内容之间添加空格；使用 **sortItemsOn()** 方法来按照内容的标签对内容进行排序。

使用 ActionScript 创建 TileList 组件：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 从“组件”面板将以下组件拖到“库”面板中：**ColorPicker**、**ComboBox**、**NumericStepper**、**CheckBox** 和 **TileList**。
3. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 ActionScript 代码：

```

import fl.controls.CheckBox;
import fl.controls.ColorPicker;
import fl.controls.ComboBox;
import fl.controls.NumericStepper;
import fl.controls.TileList;
import fl.data.DataProvider;

var aCp:ColorPicker = new ColorPicker();
var aCb:ComboBox = new ComboBox();
var aNs:NumericStepper = new NumericStepper();
var aCh:CheckBox = new CheckBox();
var aTl:TileList = new TileList();

var dp:Array = [
    {label:"ColorPicker", source:aCp},
    {label:"ComboBox", source:aCb},
    {label:"NumericStepper", source:aNs},
    {label:"CheckBox", source:aCh},
];
aTl.dataProvider = new DataProvider(dp);

```

```
aTl.columnWidth = 110;
aTl.rowHeight = 100;
aTl.setSize(280,130);
aTl.move(150, 150);
aTl.setStyle("contentPadding", 5);
aTl.sortItemsOn("label");
addChild(aTl);
```

4. 选择“控制” > “测试影片”对应用程序进行测试。

使用 UILoader

UILoader 组件是可以显示 SWF、JPEG、渐进式 JPEG、PNG 和 GIF 文件的容器。每当需要从远程位置检索内容并将其拖到 Flash 应用程序中时，您都可以使用 **UILoader**。例如，您可以使用 **UILoader** 在表单中添加公司徽标（JPEG 文件）。也可以在显示照片的应用程序中使用 **UILoader** 组件。使用 `load()` 方法加载内容；使用 `percentLoaded` 属性确定已加载内容的多少；使用 `complete` 事件确定何时完成加载。

您可以缩放 **UILoader** 的内容，或者调整 **UILoader** 自身的大小来匹配内容的大小。默认情况下，将缩放内容以适应 **UILoader**。您也可以在运行时加载内容并监控加载进度（不过内容加载一次后会被缓存，所以进度会快速跳进到 100%）。在 **UILoader** 中加载内容时，如果指定位置，必须将位置（X 和 Y 坐标）指定为 0, 0。

与 UILoader 的用户交互

UILoader 组件不能获得焦点。但是，**UILoader** 组件中加载的内容可以接受焦点，并且可以有自己的焦点交互操作。有关控制焦点的详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 **FocusManager** 类以及第 48 页的“使用 **FocusManager**”。

UILoader 参数

可以在“属性”检查器或“组件”检查器中为每个 **UILoader** 组件实例设置以下创作参数：`autoLoad`、`maintainAspectRatio`、`source` 和 `scaleContent`。其中每个参数都有对应的同名 **ActionScript** 属性。

每个 **UILoader** 实例的实时预览反映在创作过程中对“属性”检查器或“组件”检查器中的参数所做的更改。

您可以使用 **UILoader** 实例的方法、属性和事件来编写 **ActionScript**，以为其设置其它选项。有关详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 **UILoader** 类。

创建具有 UILoader 的应用程序

以下过程说明如何在创作时将 UILoader 组件添加到应用程序中。在本示例中，加载器加载一个徽标的 GIF 图像。

创建具有 UILoader 组件的应用程序：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 从“组件”面板将 UILoader 组件拖到舞台上。
3. 在“属性”检查器中，输入实例名称“aUI”。
4. 在舞台上和“组件”检查器中选择加载器，然后为 source 参数输入 **<http://www.helpexamples.com/images/logo.gif>**。

本示例使用 ActionScript 创建 UILoader 组件，并加载一个花朵的 JPEG 图像。当 complete 事件发生时，它将在“输出”面板中显示已加载的字节数。

使用 ActionScript 创建 UILoader 组件实例：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 从“组件”面板将 UILoader 组件拖到“库”面板中。
3. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 ActionScript 代码：

```
import fl.containers.UILoader;

var aLoader:UILoader = new UILoader();
aLoader.source = "http://www.flash-mx.com/images/imagel.jpg";
aLoader.scaleContent = false;
addChild(aLoader);

aLoader.addEventListener(Event.COMPLETE, completeHandler);
function completeHandler(event:Event) {
    trace("Number of bytes loaded: " + aLoader.bytesLoaded);
}
```

4. 选择“控制” > “测试影片”。

使用 UIScrollBar

使用 **UIScrollBar** 组件可以将滚动条添加到文本字段中。您可以在创作时将滚动条添加到文本字段中，或使用 **ActionScript** 在运行时添加。若要使用 **UIScrollBar** 组件，请在舞台上创建一个文本字段，并从“组件”面板将 **UIScrollBar** 组件拖到文本字段的边框的任意象限中。

如果滚动条的长度小于其滚动箭头的加总尺寸，则滚动条将无法正确显示。一个箭头按钮将隐藏在另一个的后面。**Flash** 对此不提供错误检查。在这种情况下，最好使用 **ActionScript** 隐藏滚动条。如果调整滚动条的尺寸以至没有足够的空间留给滚动框（滑块），则 **Flash** 会使滚动框变为不可见。

UIScrollBar 组件的功能与其它所有滚动条类似。它两端各有一个箭头按钮，按钮之间有一个滚动轨道和滚动框（滑块）。它可以附加至文本字段的任何一边，既可以垂直使用也可以水平使用。

有关 **TextField** 的信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 **TextField** 类。

与 UIScrollBar 的用户交互

与许多其它组件不同，**UIScrollBar** 组件能够接收连续鼠标输入（例如，用户按住鼠标按钮），而不需要重复单击。

UIScrollBar 组件和键盘不存在交互。

UIScrollBar 参数

可以在“属性”检查器或“组件”检查器中为每个 **UIScrollBar** 组件实例设置以下创作参数：**direction** 和 **scrollTargetName**。其中每个参数都有对应的同名 **ActionScript** 属性。

您可以使用 **UIScrollBar** 实例的方法、属性和事件来编写 **ActionScript**，以设置其它选项。有关详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 **UIScrollBar** 类。

创建具有 UIScrollBar 的应用程序

以下过程描述如何在创作时将 **UIScrollBar** 组件添加到应用程序中。

创建具有 UIScrollBar 组件的应用程序：

1. 创建一个新的 **Flash** 文件 (**ActionScript 3.0**) 文档。
2. 创建一个动态文本字段，其高度足以容纳一行或两行文本，然后在“属性”检查器中为其指定实例名称 **“myText”**。
3. 在“属性”检查器中，将该文本输入字段的“线条类型”设置为“多行”或“多行不换行”（如果您计划在水平方向使用滚动条）。

4. 打开“动作”面板，在主时间轴上选择第 1 帧，然后输入以下 **ActionScript** 代码来填充 `text` 属性，以便用户需要滚动字段才能查看全部文本：

```
myText.text="When the moon is in the seventh house and Jupiter aligns with  
Mars, then peace will guide the planet and love will rule the stars."
```



确保舞台上的文本字段足够小，以便需要滚动字段才能查看全部文本。否则，滚动条将不会显示，或者只显示为不带滑块抓手（拖动以滚动内容的部分）的两行。

5. 验证对象贴紧功能是否已打开（“视图” > “贴紧” > “贴紧至对象”）。
6. 从“组件”面板将一个 **UIScrollBar** 实例拖到文本输入字段上靠近要附加该实例的一边。为了能将该组件正确地绑定到文本字段，松开鼠标时该组件必须与文本字段重叠。为其指定实例名称 **mySb**。

在“属性”检查器和“组件”检查器中，该组件的 `scrollTargetName` 属性自动填充为该文本字段实例的名称。如果“参数”选项卡上未显示该属性，则可能是重叠 **UIScrollBar** 实例的程度还不够。

7. 选择“控制” > “测试影片”。

您也可以使用 **ActionScript** 在运行时创建 **UIScrollBar** 实例，并将它与文本字段相关联。下面的示例创建一个水平方向的 **UIScrollBar** 实例，并将其附加到名为 **myTxt** 的文本字段实例的底部（该字段加载来自某个 URL 的文本）。该示例还设置控制条的大小以使其与文本字段的大小相匹配：

使用 **ActionScript** 创建 **UIScrollBar** 组件实例：

1. 创建一个新的 **Flash** 文件 (**ActionScript 3.0**) 文档。
2. 将 **ScrollBar** 组件拖到“库”面板中。
3. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下 **ActionScript** 代码：

```
import flash.net.URLLoader;
import fl.controls.UIScrollBar;
import flash.events.Event;

var myTxt:TextField = new TextField();
myTxt.border = true;
myTxt.width = 200;
myTxt.height = 16;
myTxt.x = 200;
myTxt.y = 150;

var mySb:UIScrollBar = new UIScrollBar();
mySb.direction = "horizontal";
// Size it to match the text field.
mySb.setSize(myTxt.width, myTxt.height);
```

```

// Move it immediately below the text field.
mySb.move(myTxt.x, myTxt.height + myTxt.y);

// put them on the Stage
addChild(myTxt);
addChild(mySb);
// load text
var loader:URLLoader = new URLLoader();
var request:URLRequest = new URLRequest("http://www.helpexamples.com/
    flash/lorem.txt");
loader.load(request);
loader.addEventListener(Event.COMPLETE, loadcomplete);

function loadcomplete(event:Event) {
    // move loaded text to text field
    myTxt.text = loader.data;
    // Set myTxt as target for scroll bar.
    mySb.scrollTarget = myTxt;
}

```

4. 选择“控制” > “测试影片”。

自定义 UI 组件

本章说明如何自定义 Flash ActionScript 3.0 UI 组件。其中包括以下主题：

关于 UI 组件自定义	130
设置样式	130
创建新外观	136
自定义 Button	137
自定义 CheckBox	140
自定义 ColorPicker	142
自定义 ComboBox	144
自定义 DataGrid	147
自定义 Label	153
自定义 List	154
自定义 NumericStepper	157
自定义 ProgressBar	159
自定义 RadioButton	161
自定义 ScrollPane	163
自定义 Slider	164
自定义 TextArea	166
自定义 TextInput	168
自定义 TileList	170
自定义 UI Loader	172
自定义 UI ScrollBar	172

有关自定义 FLVPlayback 组件的信息，请参阅第 5 章“使用 FLVPlayback 组件”

关于 UI 组件自定义

您可以通过修改以下两个元素之一或全部两个元素，自定义应用程序中组件的外观：

样式 — 每个组件都有一组样式，您可以设置这些样式来指定 **Flash** 使用什么值呈现组件的外观。样式通常指定组件在不同状态下使用的外观和图标，以及指定要使用的文本格式和填充值。

外观 — “外观”由构成给定状态下组件图形外观的元件集合组成。当样式指定了要使用的外观时，外观就是 **Flash** 用于绘制组件的图形元素。“设置外观”是通过修改或替换组件的图形更改组件外观的过程。



可以将 **ActionScript 3.0** 组件的默认外观视为一个主题 (**Aeon Halo**)，但这些外观是组件内置的。**ActionScript 3.0** 组件不支持 **ActionScript 2.0** 组件支持的外部主题文件。

设置样式

使用 **Flash** 绘制各种状态下的组件时，通常由组件的样式来指定组件的外观、图标、文本格式和填充的值。例如，当您在按钮上单击鼠标按键时，**Flash** 将使用一个外观来绘制按钮以显示按钮的按下状态，该外观不同于在显示按钮的弹起或正常状态时所使用的外观。当按钮处于禁用状态（通过将 `enabled` 属性设置为 `false` 来实现）时，**Flash** 也会使用不同的外观。

您可以在文档、类和实例级别设置组件的样式。另外，某些样式属性可以从父组件继承。例如，**List** 组件便在继承 **BaseScrollPane** 时继承了 **ScrollBar** 样式。

您可以通过以下方式设置样式以自定义组件：

- 在组件实例上设置样式。
您可以更改单个组件实例的颜色和文本属性。这种方式在有些情况下是有效的，但是，如果您需要在文档中设置所有组件上的各项属性，这种方式就会很耗时。
- 在文档中设置给定类型的所有组件的样式。
如果您要对给定类型的所有组件（例如对文档中的所有 **CheckBox** 或所有 **Button**）应用一致的外观，则可以在组件级别设置样式。

容器组件将继承在容器上设置的样式属性的值。

在使用“实时预览”功能查看舞台上的组件时，**Flash** 并不会显示对样式属性所做的更改。

了解样式设置

以下是关于使用样式的几个要点：

继承 — 组件子级在设计时被设置为从父组件继承样式。不能在 **ActionScript** 中设置样式的继承。

优先级 — 如果以多种方式设置组件样式，则 **Flash** 会根据样式的优先级顺序使用它遇到的第一种样式。**Flash** 按如下顺序查找样式，直到找到一个值：

1. **Flash** 查找组件实例上的样式属性。
2. 如果样式是继承样式的一种，**Flash** 将彻底检查父层次结构来查找继承的值。
3. **Flash** 查找组件上的样式。
4. **Flash** 查找 **StyleManager** 上的全局设置。
5. 如果尚未定义该属性，则属性值为 `undefined`。

访问组件的默认样式

您可以使用组件类的静态 `getDefaultStyles()` 方法访问组件的默认样式。例如，下面的代码检索 **ComboBox** 组件的默认样式并显示 `buttonWidth` 和 `downArrowDownSkin` 属性的默认值：

```
import fl.controls.ComboBox;
var styleObj:Object = ComboBox.getDefaultStyles();
trace(styleObj.buttonWidth); // 21
trace(styleObj.downArrowDownSkin); // downArrowDownSkin
```

在组件实例上设置和获取样式

任何 **UI** 组件实例都可以直接调用 `setStyle()` 和 `getStyle()` 方法来设置或检索样式。以下语法为组件实例设置样式和值：

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

此语法检索组件实例的样式：

```
var a_style:Object = new Object();
a_style = instanceName.getStyle("styleName");
```

请注意，`getStyle()` 方法返回类型 **Object**，因为它可以返回具有不同数据类型的多种样式。例如，以下代码为 **TextArea** 实例 (`aTa`) 设置字体样式，然后使用 `getStyle()` 方法检索该样式。该示例将返回值转换为 **TextFormat** 对象以将其赋给 **TextFormat** 变量。如果未进行转换，在试图将 **Object** 变量强制转换为 **TextFormat** 变量时编译器将发出错误。

```
import flash.text.TextFormat;
```

```
var tf:TextFormat = new TextFormat();
```

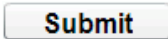
```
tf.font = "Georgia";
aTa.setStyle("textFormat",tf);
aTa.text = "Hello World!";
var aStyle:TextFormat = aTa.getStyle("textFormat") as TextFormat;
trace(aStyle.font);
```

使用 TextFormat 设置文本属性

使用 **TextFormat** 对象可以设置组件实例的文本格式。**TextFormat** 对象具有一些属性，可以通过这些属性指定文本特性，例如 **bold**、**bullet**、**color**、**font**、**italic**、**size** 等。您可以设置 **TextFormat** 对象中的这些属性，然后调用 **setStyle()** 方法将它们应用于组件实例。例如，以下代码设置 **TextFormat** 对象的 **font**、**size** 和 **bold** 属性，并将它们应用于 **Button** 实例：

```
/* Create a new TextFormat object to set text formatting properties. */
var tf:TextFormat = new TextFormat();
tf.font = "Arial";
tf.size = 16;
tf.bold = true;
a_button.setStyle("textFormat", tf);
```

下图显示了在具有“提交”标签的按钮上进行这些设置的效果：



在组件实例上通过 **setStyle()** 设置的样式属性具有最高优先级，可以覆盖所有其它样式设置。然而，您对一个组件实例使用 **setStyle()** 设置的属性越多，组件在运行时呈现得越慢。

为组件的所有实例设置样式

可以使用 **StyleManager** 类的静态 **setComponentStyle()** 方法为组件类的所有实例设置样式。例如，您可以首先将一个 **Button** 拖到舞台上，然后在时间轴第 1 帧的“动作”面板中添加以下 **ActionScript** 代码，将所有 **Button** 的文本颜色设置为红色：

```
import fl.managers.StyleManager;
import fl.controls.Button;

var tf:TextFormat = new TextFormat();
tf.color = 0xFF0000;
StyleManager.setComponentStyle(Button, "textFormat", tf);
```

您后续添加到舞台中的所有 **Button** 都将具有红色标签。

为所有组件设置样式

可以使用 `StyleManager` 类的静态 `setStyle()` 方法为所有组件设置样式。

为所有组件设置样式：

1. 将一个 `List` 组件拖到舞台上，然后为其指定实例名称 `aList`。
2. 将一个 `Button` 组件拖到舞台上，然后为其指定实例名称 `aButton`。
3. 如果尚未打开“动作”面板，可按“F9”或从“窗口”菜单选择“动作”来打开该面板，然后在时间轴的第 1 帧中输入以下代码，将所有组件的文本颜色设置为红色：

```
import fl.managers.StyleManager;

var tf:TextFormat = new TextFormat();
tf.color = 0xFF0000;
StyleManager.setStyle("textFormat", tf);
```

4. 将以下代码添加到“动作”面板以用文本填充 `List`。

```
aList.addItem({label:"1956 Chevy (Cherry Red)", data:35000});
aList.addItem({label:"1966 Mustang (Classic)", data:27000});
aList.addItem({label:"1976 Volvo (Xcllnt Cond)", data:17000});
aList.allowMultipleSelection = true;
```

5. 选择“控制”>“测试影片”或按 `Ctrl+Enter` 以编译代码并测试内容。按钮标签和列表中的文本都应该为红色。

关于外观

组件的外观由多个图形元素（例如轮廓、填充颜色、图标，甚至其它组件）构成。例如，`ComboBox` 包含一个 `List` 组件，而 `List` 组件包含一个 `ScrollBar`。多个图形元素共同构成了 `ComboBox` 的外观。不过，组件的外观会根据它们当前的状态发生变化。例如，`CheckBox`（不包括其标签）显示在应用程序中时可能会如下所示：



处于正常弹起状态的 `CheckBox`

如果您在 **CheckBox** 上单击鼠标按键并且不松开按键，其外观将更改如下：



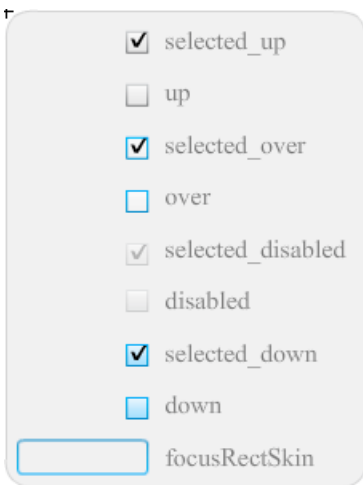
处于按下状态的 CheckBox

释放鼠标按键时，**CheckBox** 将还原为其原始外观，但此时将有一个复选标记以显示它已被选中。



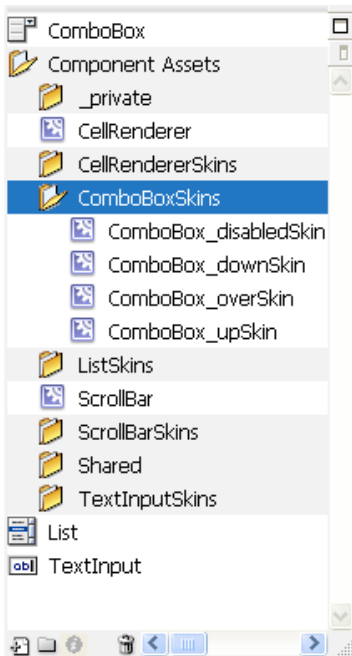
处于选中状态的 CheckBox

表示各种状态的组件的图标统称为组件的“外观”。您可以通过在 **Flash** 中编辑组件的外观更改任意或所有状态下的组件外观，就像对待任何其它 **Flash** 元件一样。可以用两种方法访问组件的外观。最简单的方法是将组件拖到舞台上并双击它。这将打开一个组件外观调色板，**CheckBox** 的外观调色板如下所示。



CheckBox 的外观

您还可以从“库”面板单独访问组件的外观。将组件拖到舞台时，您还可以将其与其资源文件夹及其包含的任何其它组件一同复制到库。例如，如果将 **ComboBox** 拖到舞台，您将在“库”面板中找到以下项目：



ComboBox 的“库”面板

除 **ComboBox** 之外，“库”面板还包含 **List**、**ScrollBar** 和 **TextInput** 组件（这些组件都内置于 **ComboBox** 中），以及其中每个组件的外观文件夹和一个包含这些组件所共享元素的 **Shared Assets** 文件夹。通过打开组件的外观文件夹（**ComboBoxSkins**、**ListSkins**、**ScrollBarSkins** 或 **TextInputSkins**）并双击您要编辑的外观的图标，您可以编辑其中任何组件的外观。例如，双击 **ComboBox_downSkin** 将在元件编辑模式下打开外观，如下图所示：



ComboBox_downSkin

创建新外观

如果您要在文档中创建组件的新外观，则可以编辑组件的外观以更改其外观。若要访问组件的外观，只需双击舞台上的组件打开其外观调色板即可。然后，双击您要编辑的外观，在元件编辑模式下打开它。例如，双击舞台上的 **TextArea** 组件，在元件编辑模式下打开其资源。将缩放控制设置为 **400%** 或更高（如果您喜欢），然后编辑元件以更改其外观。在完成编辑后，所做的更改将影响文档中组件的所有实例。作为一种替代方法，您可以双击“库”面板中的某个特定外观，以元件编辑模式在舞台上打开该外观。

您可以通过以下方式修改组件外观：

- 为所有实例创建新外观
- 为部分实例创建新外观

为所有实例创建外观

编辑组件的外观时，默认情况下，可以更改文档中组件所有实例的外观。如果您要为相同组件创建不同的外观，必须重制您要更改的外观并为它们指定不同的名称，编辑这些外观，然后设置适当的样式以应用这些外观。有关详细信息，请参阅第 136 页的“为部分实例创建外观”。

本章说明如何更改每个 UI 组件的一个或多个外观。如果您按照其中一个过程更改了 UI 组件的一个或多个外观，将会更改文档中所有实例的此外观。

为部分实例创建外观

使用以下常规过程可以为组件的部分实例创建外观：

- 在“库”面板中组件的 **Assets** 文件夹中选择外观。
- 重制外观并为其赋予唯一的类名称。
- 编辑外观，为其指定所需的外观。
- 调用组件实例的 `setStyle()` 方法，为外观样式赋予新外观。

以下过程为两个 **Button** 实例中的一个创建新的 **selectedDownSkin**。

为 **Button** 创建新的 **selectedDownSkin**：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将两个 **Button** 从“组件”面板拖到舞台上，并为其指定实例名称 **aButton** 和 **bButton**。
3. 打开“库”面板，然后打开其中的 **Component Assets** 和 **ButtonSkins** 文件夹。
4. 单击 **selectedDownSkin** 外观将其选中。
5. 右击打开上下文菜单，然后选择“直接复制”。
6. 在“重制元件”对话框中，为新外观指定唯一名称，例如 **“Button_mySelectedDownSkin”**。然后单击“确定”。

7. 在“库” > Component Assets > ButtonSkins 文件夹中，选择 **Button_mySelectedDownSkin**，并右键单击以打开上下文菜单。选择“链接”打开“链接属性”对话框。
8. 单击“为 **ActionScript** 导出”复选框。将“在第一帧导出”复选框保持选中状态并确保类名称是唯一的。单击“确定”，然后再次单击“确定”以响应警告，该警告指示无法找到类定义并将创建一个类定义。
9. 双击“库”面板中的 **Button_mySelectedDownSkin** 外观，在元件编辑模式下打开该外观。
10. 单击外观中心的蓝色填充，直至其颜色出现在“属性”检查器的“填充颜色选择器”中。单击颜色选择器并为外观填充选择颜色 **#00CC00**。
11. 单击舞台上方编辑栏左侧的“返回”按钮，返回到文档编辑模式。
12. 在“属性”检查器中，单击每个按钮的“参数”选项卡并将 **toggle** 参数设置为 **true**。
13. 将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
bButton.setStyle("selectedDownSkin", Button_mySelectedDownSkin);  
bButton.setStyle("downSkin", Button_mySelectedDownSkin);
```
14. 选择“控制” > “测试影片”。
15. 单击每个按钮。请注意，**bButton** 对象的按下外观（选中和取消选中）使用新的外观元件。

自定义 Button

在创作过程中和运行时，可以在水平和垂直方向上将 **Button** 组件变形。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改” > “变形”命令。在运行时，可使用 `setSize()` 方法或 **Button** 类的任何适用属性，如 `height`、`width`、`scaleX` 和 `scaleY`。

调整按钮大小不会更改图标或标签的大小。**Button** 的边界框对应于 **Button** 的边框，它同时也指定了实例的点击区域。如果您增加实例的大小，也就增加了点击区域的大小。如果边框太小而无法容纳标签，标签会被裁剪以适合边框。

如果 **Button** 的图标比它本身还大，则图标会延伸到 **Button** 的边框外。

对 Button 使用样式

Button 的样式通常指定在绘制各种状态下的组件时 **Button** 的外观、图标、文本格式和填充的值。

以下过程在舞台上放置两个 **Button**，并在用户单击其中一个 **Button** 时将这两个 **Button** 的 `emphasized` 属性都设置为 `true`。它还会在用户单击第二个 **Button** 时将其 `emphasizedSkin` 样式设置为 `selectedOverSkin` 样式，以便这两个 **Button** 在相同状态下显示不同的外观。

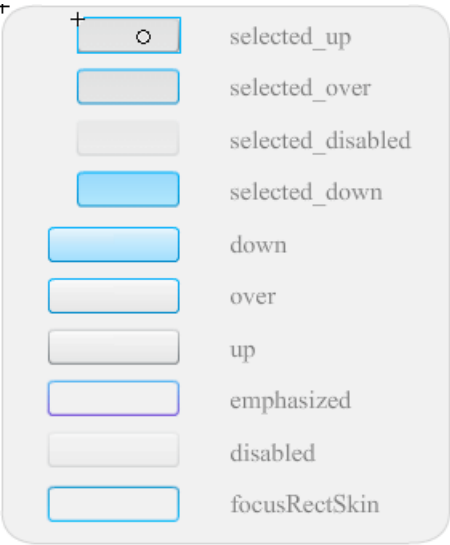
更改 Button 的 `emphasizedSkin` 样式：

1. 创建一个 Flash 文件 (ActionScript 3.0)。
2. 将两个 **Button** 拖到舞台上（一次拖动一个），并分别为它们指定实例名称 **aBtn** 和 **bBtn**。在“属性”检查器的“参数”选项卡中，将它们的标签分别指定为 **Button A** 和 **Button B**。
3. 将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
bBtn.emphasized = true;
aBtn.emphasized = true;
bBtn.addEventListener(MouseEvent.CLICK, Btn_handler);
function Btn_handler(evt:MouseEvent):void {
    bBtn.setStyle("emphasizedSkin", "Button_selectedOverSkin");
}
```
4. 选择“控制” > “测试影片”。
5. 单击其中一个按钮，查看每个按钮上 `emphasizedSkin` 样式的效果。

对 Button 使用外观

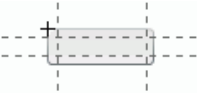
对应于不同状态 **Button** 组件分别使用以下外观。若要编辑一个或多个外观来更改 **Button** 的外观，请双击舞台上的 **Button** 实例，打开其外观调色板，如下图所示：



Button 外观

如果按钮已启用，当指针移动到它的上方时，它会显示指针经过状态。在按下按钮时，按钮将接收输入焦点并显示按下状态。当松开鼠标后，按钮又返回到指针经过状态。如果鼠标按下时指针移离按钮，按钮会恢复到原始状态。如果将 **toggle** 参数设置为 **true**，将用 **selectedDownSkin** 显示按下状态，用 **selectedUpSkin** 显示弹起状态，用 **selectedOverSkin** 显示经过状态。

如果 **Button** 被禁用，不管用户进行什么交互操作，它都会显示其禁用状态。
若要编辑一个外观，请双击该外观以在元件编辑模式下打开它，如下图所示：

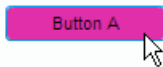


此时，您可以使用 **Flash** 创作工具将该外观编辑为您喜欢的样式。
以下过程更改 **Button** 的 **selected_over** 外观的颜色。

更改 Button 的 `selected_over` 外观的颜色：

1. 创建新的 Flash 文件 (ActionScript 3.0)。
2. 将一个 Button 从“组件”面板拖到舞台上。在“参数”选项卡中，将 `toggle` 参数设置为 `true`。
3. 双击该 Button 打开其外观调色板。
4. 双击 `selected_over` 外观，在元件编辑模式下打开它。
5. 将缩放控制设置为 400%，以便放大图标进行编辑。
6. 双击背景，直至其颜色显示在“属性”检查器的“填充颜色选择器”中。
7. 在“填充颜色选择器”中选择颜色 `#CC0099` 以将其应用于 `selected_over` 外观的背景。
8. 单击舞台上编辑栏左侧的“返回”按钮，返回到文档编辑模式。
9. 选择“控制”>“测试影片”。
10. 单击按钮，使其处于选中状态。

将鼠标指针移动到 Button 上方时，`selected_over` 状态应按下图所示显示。



自定义 CheckBox

在创作过程中和运行时，可以在水平和垂直方向上将 `CheckBox` 组件变形。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改”>“变形”命令。在运行时，可使用 `setSize()` 方法或 `CheckBox` 类的适用属性。例如，您可以通过设置 `CheckBox` 的 `height`、`width`、`scaleX` 和 `scaleY` 属性来更改其大小。调整 `CheckBox` 的大小不会改变标签或复选框图标的大小；它只会改变边框的大小。

`CheckBox` 实例的边框是不可见的，它同时也指定了该实例的点击区域。如果您增加实例的大小，也就增加了点击区域的大小。如果边框太小而无法容纳标签，标签会被裁剪以适合边框。

对 CheckBox 使用样式

您可以设置样式属性以更改 `CheckBox` 实例的外观。例如，下面的过程更改了 `CheckBox` 标签的大小和颜色。

更改 CheckBox 标签的大小和颜色:

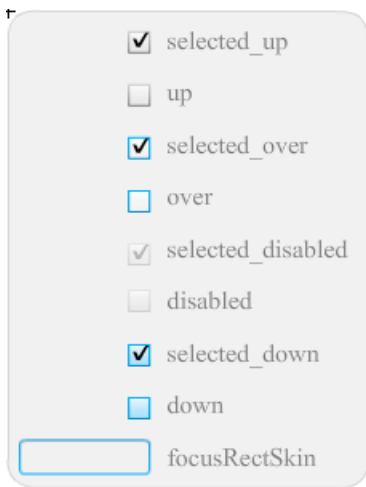
1. 将 CheckBox 组件从“组件”面板拖动到舞台上，并为其指定实例名称 **myCb**。
2. 在“属性”检查器中单击“参数”选项卡，然后为标签参数输入以下值：“**Less than \$500?**”
3. 在主时间轴第 1 帧的“动作”面板中，输入以下代码：

```
var myTf:TextFormat = new TextFormat();
myCb.setSize(150, 22);
myTf.size = 16;
myTf.color = 0xFF0000;
myCb.setStyle("textFormat", myTf);
```

有关详细信息，请参阅第 130 页的“设置样式”。有关设置样式属性以更改组件的图标和外观的信息，请参阅第 136 页的“创建新外观”和第 141 页的“对 CheckBox 使用外观”。

对 CheckBox 使用外观

CheckBox 组件具有以下外观，您可以对它们进行编辑以更改 CheckBox 组件的外观。



CheckBox 外观

此示例更改此组件在处于 up 和 selectedUp 状态时的轮廓颜色和背景颜色。您可以按照类似步骤来更改其它状态的外观。

自定义 CheckBox 外观:

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将 CheckBox 组件拖动到舞台上，这同时会将它放置在包含其资源文件夹的库中。
3. 在舞台上双击此 CheckBox 组件以打开包含其外观图标的面板。
4. 双击 `selected_up` 图标，以在元件编辑模式下将其打开。
5. 将缩放控制设置为 800%，以便放大图标进行编辑。
6. 单击 CheckBox 的边框将其选中。使用“属性”检查器的“填充颜色选择器”来选择颜色 #0033FF，并将其应用于边框。
7. 双击此 CheckBox 的背景以选中它，然后再次使用“填充颜色选择器”将此背景的颜色设置为 #00CCFF。
8. 对 CheckBox 弹起外观重复执行第 4 步至第 8 步。
9. 选择“控制”>“测试影片”。

自定义 ColorPicker

只能通过 `ColorPicker` 的样式来调整其大小：`swatchWidth`、`swatchHeight`、`backgroundPadding`、`textFieldWidth` 和 `textFieldHeight`。如果尝试使用“变形”工具或者使用 `ActionScript` 中的 `setSize()` 方法或 `width`、`height`、`scaleX` 或 `scaleY` 属性来更改 `ColorPicker` 的大小，则在创建 SWF 文件时将忽略这些值，`ColorPicker` 将以其默认大小显示。调色板背景将调整大小以与使用 `columnCount` 样式的 `setStyle()` 设置的列数匹配。默认的列数为 18。您可以将自定义颜色设置为 1024，调色板将调整其在垂直方向上的大小以与样本的数量匹配。

对 ColorPicker 使用样式

可以设置若干样式来更改 `ColorPicker` 组件的外观。例如，下面的过程将 `ColorPicker` 中的列数 (`columnCount`) 更改为 12，并更改了颜色样本的高度 (`swatchHeight`) 和宽度 (`swatchWidth`)，然后同时更改了文本字段的填充 (`textPadding`) 和背景填充 (`backgroundPadding`)。

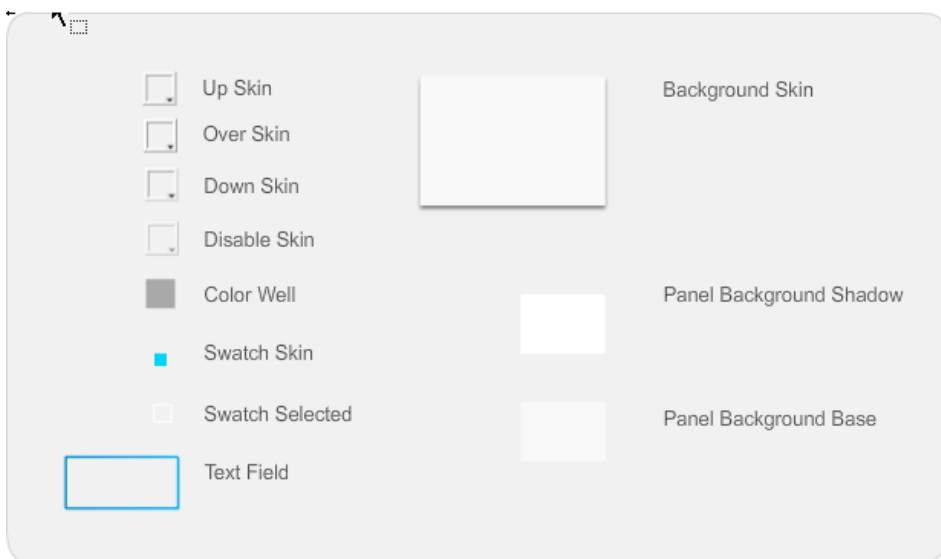
使用样式更改 ColorPicker 的外观:

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将 ColorPicker 组件拖动到舞台上, 并为其指定实例名称 **aCp**。
3. 打开“动作”面板, 在主时间轴中选择第 1 帧, 然后输入以下代码:

```
aCp.setStyle("columnCount", 12);  
aCp.setStyle("swatchWidth", 8);  
aCp.setStyle("swatchHeight", 12);  
aCp.setStyle("swatchPadding", 2);  
aCp.setStyle("backgroundPadding", 3);  
aCp.setStyle("textPadding", 7);
```
4. 选择“控制” > “测试影片”。
5. 单击 ColorPicker 将其打开, 查看这些设置对它的外观有哪些改变。

对 ColorPicker 使用外观

ColorPicker 组件使用以下外观来表示其可视状态。



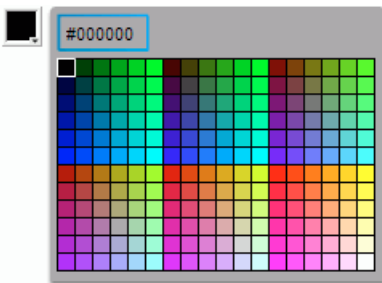
ColorPicker 外观

可以更改背景外观的颜色, 从而更改调色板背景的颜色。

更改背景外观的颜色：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将 ColorPicker 组件拖动到舞台上。
3. 双击它以打开其外观调色板。
4. 双击背景外观，直至选中它并且“填充颜色选择器”出现在“属性”检查器中。
5. 使用“填充颜色选择器”选择颜色 #999999，以将其应用于背景外观。
6. 单击舞台上方编辑栏左侧的“返回”按钮，返回到文档编辑模式。
7. 选择“控制” > “测试影片”。

单击此 ColorPicker 时，调色板的背景应为灰色，如下图所示。



自定义 ComboBox

在创作过程中和运行时，可以在水平和垂直方向上将 **ComboBox** 组件变形。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改” > “变形”命令。在运行时，可使用 `setSize()` 方法或 **ComboBox** 类的适用属性，如 `height` 和 `width` 属性，以及 `scaleX` 和 `scaleY`。

ComboBox 将调整大小以适合指定宽度和高度。除非已设置 `dropdownWidth` 属性，否则列表将调整大小以适合组件的宽度。

如果文本太长而不能在 **ComboBox** 中完全显示，将会裁剪此文本以适合 **ComboBox**。您必须调整 **ComboBox** 的大小并设置 `dropdownWidth` 属性以适合此文本。

对 ComboBox 使用样式

您可以设置样式属性来更改 **ComboBox** 组件的外观。样式指定了组件外观、单元格渲染器、填充和按钮宽度的值。下面的示例设置了 `buttonWidth` 和 `textPadding` 样式。

`buttonWidth` 样式设置按钮点击区域的宽度，并在 **ComboBox** 可编辑时生效，您只能按此按钮来打开下拉列表。`textPadding` 样式指定文本字段的外侧边框与文本之间的空间量。如果增加 **ComboBox** 的高度，则将文本在文本字段中垂直居中会很有用。否则，文本可能会显示在文本字段的顶部。

对 ComboBox 设置样式：

1. 创建一个新的 **Flash** 文件 (ActionScript 3.0) 文档。
2. 将 **ComboBox** 组件拖动到舞台上，并为其指定实例名称 “**aCb**”。
3. 打开 “动作” 面板，在主时间轴中选择第 1 帧，然后输入以下代码：

```
import fl.data.DataProvider;

aCb.setSize(150, 35);
aCb.setStyle("textPadding", 10);
aCb.setStyle("buttonWidth", 10);
aCb.editable = true;

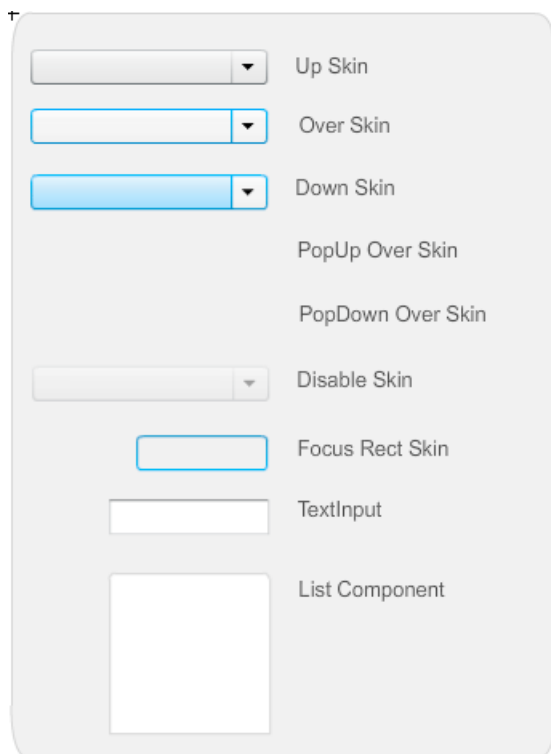
var items:Array = [
    {label:"San Francisco", data:"601 Townsend St."},
    {label:"San Jose", data:"345 Park Ave."},
    {label:"San Diego", data:"10590 West Ocean Air Drive, Suite 100"},
    {label:"Santa Rosa", data:"2235 Mercury Way, Suite 105"},
    {label:"San Luis Obispo", data:"3220 South Higuera Street, Suite 311"}
];
aCb.dataProvider = new DataProvider(items);
```

4. 选择 “控制” > “测试影片”。

请注意，可以用来单击以打开下拉列表的按钮区域只是右侧的一个狭窄区域。还请注意，文本已在文本字段中垂直居中。您可以尝试删去这两个 `setStyle()` 语句，再运行此示例，看看产生的效果。

对 ComboBox 使用外观

ComboBox 使用以下外观来表示其可视状态：



ComboBox 外观

可以通过更改 Up 外观的颜色来更改组件在舞台上处于非活动状态时的颜色。

更改背景外观的颜色：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将 ComboBox 组件拖动到舞台上。
3. 双击它以打开其外观调色板。
4. 双击 Up 外观，直至将其选中并打开以供编辑。
5. 将缩放控件设置为 400%。
6. 单击此外观的中心区域，直至其颜色出现在“属性”检查器中的“填充颜色选择器”中。
7. 使用“填充颜色选择器”选择颜色 #33FF99 以将其应用于此 Up 外观。

8. 单击舞台上上方编辑栏左侧的“返回”按钮，返回到文档编辑模式。
9. 选择“控制” > “测试影片”。

此 **ComboBox** 在舞台上的外观应该如下图所示。



自定义 DataGrid

在创作过程中和运行时，可以沿水平方向和垂直方向将 **DataGrid** 组件变形。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改” > “变形”命令。在运行时，可使用 `setSize()` 方法或适用的属性，如 `width`、`height`、`scaleX` 和 `scaleY`。如果没有水平滚动条，列宽度将按比例进行调整。如果调整了列大小（因而也调整了单元格大小），则单元格中的文本可能会被裁剪。

对 DataGrid 使用样式

您可以设置样式属性以更改 **DataGrid** 组件的外观。**DataGrid** 组件从 **List** 组件继承样式（请参阅第 154 页的“对 **List** 使用样式”。）

为单个列设置样式

DataGrid 对象可以有多个列，并且可以为每个列指定不同的单元格渲染器。**DataGrid** 的每个列均由一个 **DataGridColumn** 对象表示，并且 **DataGridColumn** 类包含 `cellRenderer` 属性，可以通过此属性为该列定义 **CellRenderer**。

在 **DataGrid** 中创建一个多行列：

1. 创建一个新的 Flash 文件 (ActionScript 3.0)
2. 将 **DataGrid** 组件拖动到“库”面板中。
3. 将以下代码添加到时间轴第一帧的“动作”面板上。这段代码创建了一个在第三列中包含一个较长文本字符串的 **DataGrid**。最后，它将该列的 `cellRenderer` 属性设置为用来渲染多行单元格的单元格渲染器的名称。

```
/* This is a simple cell renderer example. It invokes
the MultilineCell cell renderer to display a multiple
line text field in one of a DataGrid's columns. */
```

```
import fl.controls.DataGrid;
import fl.controls.dataGridClasses.DataGridColumn;
import fl.data.DataProvider;
import fl.controls.ScrollPolicy;
```

```

// Create a new DataGrid component instance.
var aDg:DataGrid = new DataGrid();

var aLongString:String = "An example of a cell renderer class that
    displays a multiple line TextField"
var myDP:Array = new Array();
myDP = [{firstName:"Winston", lastName:"Elstad", note:aLongString,
    item:100},
    {firstName:"Ric", lastName:"Dietrich", note:aLongString, item:101},
    {firstName:"Ewing", lastName:"Canepa", note:aLongString, item:102},
    {firstName:"Kevin", lastName:"Wade", note:aLongString, item:103},
    {firstName:"Kimberly", lastName:"Dietrich", note:aLongString,
    item:104},
    {firstName:"AJ", lastName:"Bilow", note:aLongString, item:105},
    {firstName:"Chuck", lastName:"Yushan", note:aLongString, item:106},
    {firstName:"John", lastName:"Roo", note:aLongString, item:107},
];

// Assign the data provider to the DataGrid to populate it.
// Note: This has to be done before applying the cellRenderers.
aDg.dataProvider = new DataProvider(myDP);

/* Set some basic grid properties.
Note: The data grid's row height should reflect
the number of lines you expect to show in the multiline cell.
The cell renderer will size to the row height.
About 40 for 2 lines or 60 for 3 lines.*/

aDg.columns = ["firstName", "lastName", "note", "item"];
aDg.setSize(430,190);
aDg.move(40,40);
aDg.rowHeight = 40; // Allows for 2 lines of text at default text size.
aDg.columns[0].width = 70;
aDg.columns[1].width = 70;
aDg.columns[2].width = 230;
aDg.columns[3].width = 60;
aDg.resizableColumns = true;
aDg.verticalScrollPolicy = ScrollPolicy.AUTO;
addChild(aDg);
// Assign cellRenderers.
var col3:DataGridColumn = new DataGridColumn();
col3 = aDg.getColumnAt(2);
col3.cellRenderer = MultiLineCell;

```

4. 将 FLA 文件另存为 **MultiLineGrid.fla**。

5. 创建一个新的 **ActionScript** 文件。

6. 将以下 **ActionScript** 代码复制到 “脚本” 窗口中：

```
package {

    import fl.controls.listClasses.CellRenderer;

    public class MultiLineCell extends CellRenderer
    {

        public function MultiLineCell()
        {
            textField.wordWrap = true;
            textField.autoSize = "left";
        }
        override protected function drawLayout():void {
            textField.width = this.width;
            super.drawLayout();
        }
    }
}
```

7. 将此 **ActionScript** 文件另存为 **MultiLineCell.as**，保存在您用来保存 **MultiLineGrid.fla** 的同一文件夹中。

8. 返回到 **MultiLineGrid.fla** 应用程序，并选择 “控制” > “测试影片”。

DataGrid 应如下所示：

firstName	lastName	note	item
Winston	Elstad	An example of a cell renderer class that displays a multiple line TextField	100
Ric	Dietrich	An example of a cell renderer class that displays a multiple line TextField	101
Ewing	Canepa	An example of a cell renderer class that displays a multiple line TextField	102
Kevin	Wade	An example of a cell renderer class that displays a multiple line TextField	103

设置标题样式

可以使用 `headerTextFormat` 样式来设置标题行的文本样式。下面的示例使用 `TextFormat` 对象将 `headerTextFormat` 样式设置为使用 **Arial** 字体、红色、字体大小 14 和斜体。

为 **DataGrid** 设置 `headerTextFormat` 样式：

1. 创建一个新的 **Flash** 文件 (ActionScript 3.0) 文档。
2. 将 **DataGrid** 组件拖动到舞台上，并为其指定实例名称 **aDg**。
3. 打开“动作”面板，在主时间轴中选择第 1 帧，然后输入以下代码：

```
import fl.data.DataProvider;
import fl.controls.dataGridClasses.DataGridColumn;

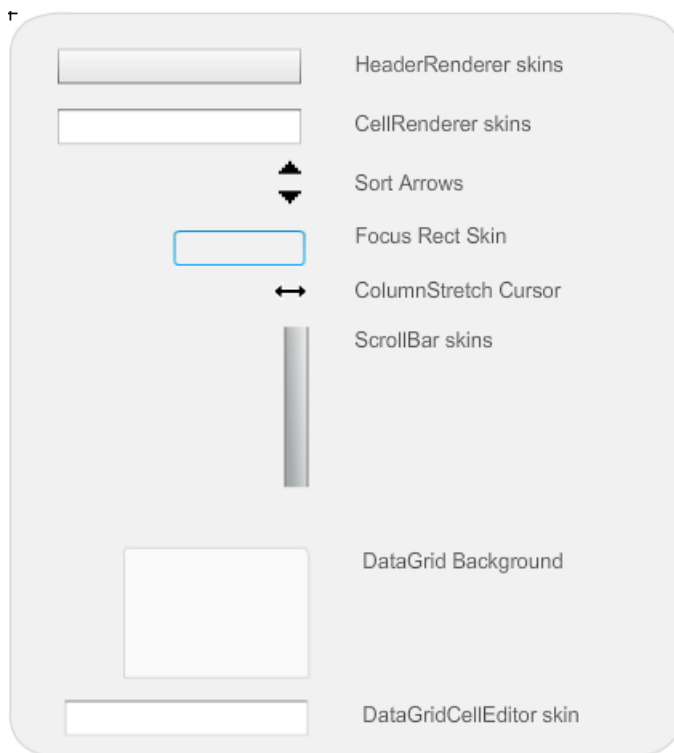
var myDP:Array = new Array();
myDP = [{FirstName:"Winston", LastName:"Elstad"},
        {FirstName:"Ric", LastName:"Dietrich"},
        {FirstName:"Ewing", LastName:"Canepa"},
        {FirstName:"Kevin", LastName:"Wade"},
        {FirstName:"Kimberly", LastName:"Dietrich"},
        {FirstName:"AJ", LastName:"Bilow"},
        {FirstName:"Chuck", LastName:"Yushan"},
        {FirstName:"John", LastName:"Roo"}
];

// Assign the data provider to the DataGrid to populate it.
// Note: This has to be done before applying the cellRenderers.
aDg.dataProvider = new DataProvider(myDP);
aDg.setSize(160,190);
aDg.move(40,40);
aDg.columns[0].width = 80;
aDg.columns[1].width = 80;
var tf:TextFormat = new TextFormat();
tf.size = 14;
tf.color = 0xff0000;
tf.italic = true;
tf.font = "Arial"
aDg.setStyle("headerTextFormat", tf);
```

4. 选择“控制” > “测试影片”，运行应用程序。

对 DataGrid 使用外观

DataGrid 组件使用以下外观来表示其可视状态：



DataGrid 外观

CellRenderer 外观是用于 DataGrid 正文单元格的外观，而 HeaderRenderer 外观则用于标题行。下面的过程更改了标题行的背景颜色，但通过编辑 CellRenderer 外观，还可以使用该过程来更改 DataGrid 正文单元格的背景颜色。

更改 DataGrid 标题行的背景颜色：

1. 创建一个新的 Flash 文件 (ActionScript 3.0)。
2. 将 DataGrid 组件拖动到舞台上，并为其指定实例名称 **adg**。
3. 双击此组件以打开其外观调色板。
4. 将缩放控件设置为 400%，放大图标以进行编辑。

- 5. 双击该 HeaderRenderer 外观打开 HeaderRenderer 外观调色板。
- 6. 双击 Up_Skin 将其在元件编辑模式下打开，并单击其背景，直至选中它并且“填充颜色选择器”出现在“属性”检查器中。
- 7. 使用“填充颜色选择器”选择颜色 #00CC00 以将其应用于 Up_Skin HeaderRenderer 外观的背景。
- 8. 单击舞台上方编辑栏左侧的“返回”按钮，返回到文档编辑模式。
- 9. 将以下代码添加到时间轴第 1 帧的“动作”面板中，以向 DataGrid 添加数据：

```
import fl.data.DataProvider;

bldRosterGrid(aDg);
var aRoster:Array = new Array();
aRoster = [
    {Name:"Wilma Carter", Home: "Redlands, CA"},
    {Name:"Sue Pennypacker", Home: "Athens, GA"},
    {Name:"Jill Smithfield", Home: "Spokane, WA"},
    {Name:"Shirley Goth", Home: "Carson, NV"},
    {Name:"Jennifer Dunbar", Home: "Seaside, CA"}
];
aDg.dataProvider = new DataProvider(aRoster);
function bldRosterGrid(dg:DataGrid){
    dg.setSize(400, 130);
    dg.columns = ["Name", "Home"];
    dg.move(50,50);
    dg.columns[0].width = 120;
    dg.columns[1].width = 120;
};
```

- 10. 选择“控制” > “测试影片”对应用程序进行测试。

DataGrid 的外观应该如下图所示，标题行的背景为绿色。

Name	Home
Wilma Carter	Redlands, CA
Sue Pennypacker	Athens, GA
Jill Smithfield	Spokane, WA
Shirley Goth	Carson, NV
Jennifer Dunbar	Seaside, CA

自定义 Label

在创作过程中和运行时，可以在水平和垂直方向上将 **Label** 组件变形。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改”>“变形”命令。您也可以设置 `autoSize` 创作参数；设置此参数不会改变实时预览中的边框，但是会调整 **Label** 的大小。**Label** 的大小是根据 `wordwrap` 参数调整的。如果此参数为 `true`，则沿垂直方向调整 **Label** 大小以适合文本。如果此参数为 `false`，则沿水平方向调整 **Label** 大小。在运行时，使用 `setSize()` 方法。有关详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 `Label.setSize()` 方法和 `Label.autoSize` 属性。另请参阅第 88 页的“创建具有 **Label** 组件的应用程序”。

对 Label 使用样式

您可以设置样式属性来更改标签实例的外观。**Label** 组件实例中的所有文本必须采用相同的样式。**Label** 组件具有 `textFormat` 样式，此样式与 **TextFormat** 对象具有相同的属性；并且，对于您可以为常规 **Flash TextField** 设置的属性，您同样可以用此样式为 `Label.text` 的内容设置这些属性。下面的示例将标签中文本的颜色设置为红色。

更改标签文本的颜色：

1. 将 **Label** 组件从“组件”面板拖动到舞台上，并为其指定实例名称 `a_label`。

2. 单击“参数”选项卡并将 `text` 属性的值替换为下面的文本：

Color me red

3. 在主时间轴中选择第 1 帧，打开“动作”面板，然后输入以下代码：

```
/* Create a new TextFormat object, which allows you to set multiple text
   properties at a time. */

var tf:TextFormat = new TextFormat();
tf.color = 0xFF0000;
/* Apply this specific text format (red text) to the Label instance. */
a_label.setStyle("textFormat", tf);
```

4. 选择“控制”>“测试影片”。

有关 **Label** 样式的详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 **Label** 类。

对 Label 使用外观

Label 组件没有任何可设置外观的可视元素。

自定义 List

在创作过程中和运行时，可以在水平和垂直方向上将 **List** 组件变形。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改”>“变形”命令。在运行时，可使用 `setSize()` 方法和 **List** 类的适用属性，如 `height`、`width`、`scaleX` 和 `scaleY`。

调整列表的大小后，列表的行会在水平方向收缩，剪下其中的任何文本。在垂直方向，列表根据需要增加或删除行。滚动条会根据需要自动调整位置。

对 List 使用样式

您可以设置样式属性以更改 **List** 组件的外观。样式指定了绘制组件时组件的外观和填充的值。

不同的外观样式允许您指定不同的类用于外观。有关使用外观样式的详细信息，请参阅第 133 页的“关于外观”。

下面的过程为 **List** 组件设置了 `contentPadding` 样式的值。请注意，会从 **List** 的大小中减去此设置的值以获得内容周围的填充值，因此您可能需要增加 **List** 的大小以防止 **List** 中的文本被裁切。

为 List 设置 contentPadding 样式：

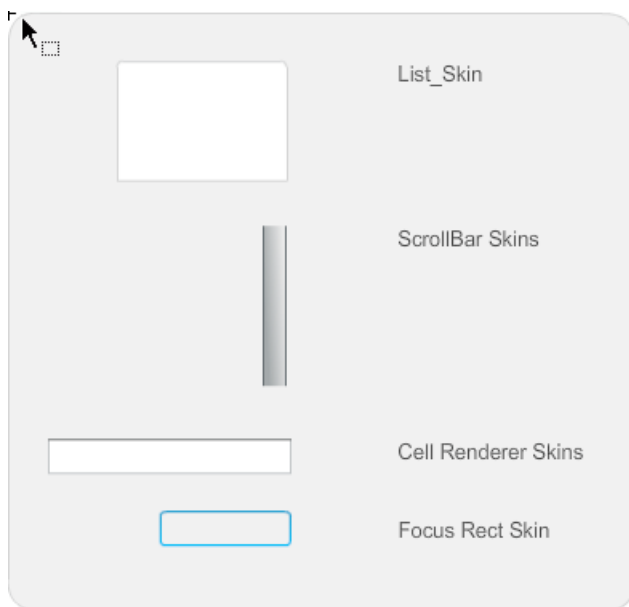
1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将 **List** 组件从“组件”面板中拖到舞台上，并为其指定实例名称 **aList**。
3. 在主时间轴中选择第 1 帧，打开“动作”面板，然后输入以下代码，这些代码用于设置 `contentPadding` 样式并向 **List** 添加数据：

```
aList.setStyle("contentPadding", 5);
aList.setSize(145, 200);
aList.addItem({label:"1956 Chevy (Cherry Red)", data:35000});
aList.addItem({label:"1966 Mustang (Classic)", data:27000});
aList.addItem({label:"1976 Volvo (Xcllnt Cond)", data:17000});
aList.rowCount = aList.length;
```

4. 选择“控制”>“测试影片”。

对 List 使用外观

List 组件使用以下外观来表示其可视状态：



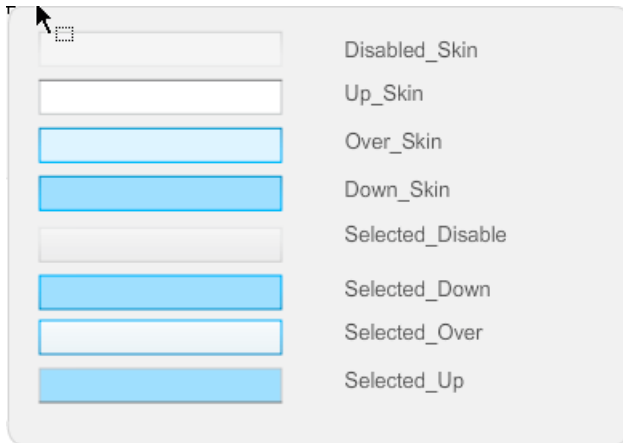
List 外观

有关设置 ScrollBar 外观的详细信息，请参阅第 172 页的“自定义 UIScrollBar”。有关设置 Focus Rect 外观的信息，请参阅第 166 页的“自定义 TextArea”。



如果更改一个组件的 ScrollBar 外观，将会更改使用 ScrollBar 的所有其它组件的 ScrollBar 外观。

双击“单元格渲染器”外观，为 List 单元格的不同状态打开另一个外观调色板。



List 单元格渲染器外观

您可以通过编辑这些外观来更改 List 的单元格外观。以下过程更改 Up 外观的颜色以更改 List 在正常的非活动状态下的外观。

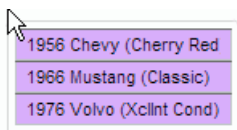
更改 List 单元格渲染器 Up_Skin 的颜色：

1. 创建一个新的 Flash 文件 (ActionScript 3.0) 文档。
2. 将 List 组件从“组件”面板中拖到舞台上，并为其指定实例名称 **aList**。
3. 双击该 List 组件打开其外观调色板。
4. 双击“单元格渲染器”外观打开“单元格渲染器”外观的调色板。
5. 双击 Up_Skin 外观将其打开以进行编辑。
6. 单击此外观的填充区域将其选中。“属性”检查器中将显示一个带有外观当前填充颜色的“填充颜色选择器”。
7. 使用“填充颜色选择器”选择颜色 #CC66FF 以将其应用于 Up_Skin 外观的填充。
8. 单击舞台上编辑栏左侧的“返回”按钮，返回到文档编辑模式。
9. 将以下代码添加到时间轴第 1 帧的“动作”面板中，以将数据添加到 List 组件：

```
aList.setStyle("contentPadding", 5);
aList.setSize(145, 200);
aList.addItem({label:"1956 Chevy (Cherry Red)", data:35000});
aList.addItem({label:"1966 Mustang (Classic)", data:27000});
aList.addItem({label:"1976 Volvo (Xcllnt Cond)", data:17000});
aList.rowCount = aList.length;
```

10. 选择 “控制” > “测试影片”。

List 的外观应如下图所示：



这些帧是由于设置了 `contentPadding` 样式而生成的。

自定义 NumericStepper

在创作过程中和运行时，可以在水平和垂直方向上将 **NumericStepper** 组件变形。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改”>“变形”命令。在运行时，可使用 `setSize()` 方法或 **NumericStepper** 类的任何适用属性和方法，比如 `width`、`height`、`scaleX` 和 `scaleY`。

调整 **NumericStepper** 组件的大小不会改变上下箭头按钮的宽度。如果将步进器的大小调整为大于默认的高度，默认行为会将箭头按钮固定在组件顶部和底部。否则，由 9 切片缩放确定如何绘制箭头。箭头按钮会始终出现在文本框的右侧。

对 NumericStepper 使用样式

您可以设置 **NumericStepper** 组件的样式属性以更改其外观。该样式在绘制组件时指定了组件的外观、填充和文本格式的值。`textFormat` 样式允许您更改 **NumericStepper** 的值的尺寸和外观。不同的外观样式允许您指定不同的类用于外观。有关使用外观样式的详细信息，请参阅第 133 页的“关于外观”。

更改 Numeric Stepper 的值的外观：

此过程使用 `textFormat` 样式更改 **NumericStepper** 显示的值的外观。

1. 创建一个新的 Flash 文件 (ActionScript 3.0)。
2. 将 **NumericStepper** 组件从“组件”面板中拖到舞台上，并为其指定实例名称 **myNs**。
3. 将以下代码添加到主时间轴第 1 帧的“动作”面板上。

```
var tf:TextFormat = new TextFormat();
myNs.setSize(100, 50);
tf.color = 0x0000CC;
tf.size = 24;
tf.font = "Arial";
tf.align = "center";
myNs.setStyle("textFormat", tf);
```

4. 选择 “控制” > “测试影片”。

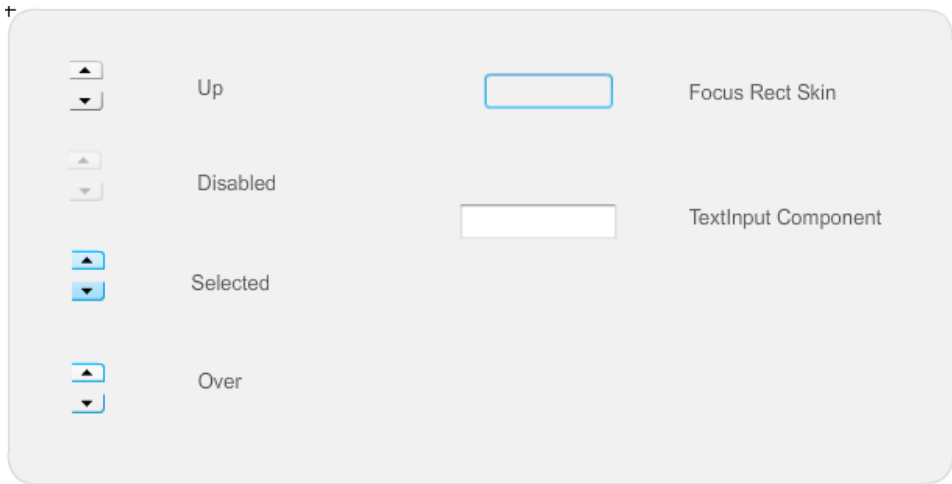
对 NumericStepper 使用外观

NumericStepper 组件具有表示其按钮的弹起、按下、禁用和选择状态的外观。

如果启用了步进器，当指针移到向上按钮和向下按钮的上方时，这些按钮会显示其悬停状态。这些按钮在被按下时，会显示为按下状态。当释放鼠标后，这些按钮又会返回其悬停状态。如果鼠标按下时指针移离按钮，按钮会恢复到其原始状态。

如果禁用了步进器，不论用户进行什么交互性操作，它都会显示其禁用状态。

NumericStepper 组件具有以下外观：



NumericStepper 外观

更改 NumericStepper 文本背景和处于弹起状态的按钮的颜色：

1. 创建一个新的 FLA 文件。
2. 将 NumericStepper 组件拖动到舞台上。
3. 将“缩放”控件设置为 400%，从而放大图像以便进行编辑。
4. 双击外观面板中 TextInput 外观的背景，直到进入到“组”级别和在“属性”检查器中的“填充颜色选择器”中显示了背景色。
5. 使用“属性”检查器中的“填充颜色选择器”，选择颜色 #9999FF 以将其应用于 TextInput 外观的背景。
6. 单击舞台上方编辑栏左侧的“返回”按钮，返回到文档编辑模式。
7. 再次双击 NumericStepper 以重新打开外观调色板。
8. 双击 Up 组中的向上箭头按钮的背景，直到选中了背景且背景颜色出现在“属性”检查器的“填充颜色选择器”中。

9. 选择颜色 #9966FF 以将其应用于向上箭头按钮的背景。
10. 对 Up 组中的向下箭头重复步骤 8 和步骤 9。
11. 选择 “控制” > “测试影片”。

显示的 NumericStepper 实例应如下图所示：



自定义 ProgressBar

在创作过程中和运行时，可以在水平和垂直方向上将 **ProgressBar** 组件变形。在创作时，在舞台上选择组件并使用 “任意变形” 工具或任何 “修改” > “变形” 命令。在运行时，可使用 **ProgressBar** 类的 `setSize()` 方法或适合的属性，比如 `height`、`width`、`scaleX` 和 `scaleY`。

ProgressBar 有三个外观：轨道外观、进度栏外观和不确定外观。它使用 9 切片缩放来缩放资源。

对 ProgressBar 使用样式

您可以设置样式属性来更改 **ProgressBar** 实例的外观。**ProgressBar** 的样式在绘制组件时指定组件的外观和填充的值。以下示例放大 **ProgressBar** 实例的大小并设置其 `barPadding` 样式。

为 **ProgressBar** 实例设置填充样式：

1. 创建一个新的 FLA 文件。
2. 将 **ProgressBar** 组件从 “组件” 面板中拖到舞台上，并为其指定实例名称 **myPb**。
3. 在主时间轴第 1 帧的 “动作” 面板中，输入以下代码：

```
myPb.width = 300;  
myPb.height = 30;  
  
myPb.setStyle("barPadding", 3);
```

4. 选择 “控制” > “测试影片”。

有关设置外观样式的信息，请参阅第 133 页的 “关于外观”。

对 ProgressBar 使用外观

ProgressBar 组件使用表示进度栏轨道、已完成栏和不确定栏的外观，如下图所示。



ProgressBar 外观

该进度栏放置于轨道外观上，使用 `barPadding` 来确定位置。使用 9 切片缩放来缩放资源。在 `ProgressBar` 实例的 `indeterminate` 属性设置为 `true` 时，会使用不确定栏。在垂直方向和水平方向上调整此外观的大小以适合 `ProgressBar` 的大小。

您可以编辑这些外观以更改 `ProgressBar` 的外观。例如，以下示例更改了不确定栏的颜色。

通过编辑不确定栏的外观更改其颜色：

1. 创建一个新的 FLA 文件。
2. 将一个 `ProgressBar` 组件拖到舞台上，并双击该组件以打开其外观图标面板。
3. 双击不确定栏外观。
4. 将缩放控制设置为 400%，以便放大图标进行编辑。
5. 双击某个对角栏，然后在按住 `Shift` 键的同时单击其余每个对角栏。当前颜色将显示在“属性”检查器的“填充颜色选择器”中。
6. 单击“属性”检查器中的“填充颜色选择器”将其打开，选择颜色 `#00CC00` 以应用于选择的对角栏。
7. 单击舞台上编辑栏左侧的“返回”按钮，返回到文档编辑模式。
8. 选择“控制” > “测试影片”。

显示的 `ProgressBar` 应如下图所示。



自定义 RadioButton

在创作过程中和运行时，可以在水平和垂直方向上将 **RadioButton** 组件变形。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改”>“变形”命令。在运行时，使用 `setSize()` 方法。

RadioButton 组件的边框是不可见的，它同时也指定了组件的点击区。如果您增加组件的大小，也就增加了点击区的大小。

如果组件边框太小而无法容纳组件标签，将会裁剪标签以适合边框。

对 RadioButton 使用样式

您可以设置样式属性以更改 **RadioButton** 的外观。**ScrollPane** 的样式属性在绘制组件时指定组件的外观、图标、文本格式和填充的值。**ScrollPane** 的样式在绘制组件时指定组件布局的外观和填充的值。

以下示例从 **CheckBox** 组件中检索 `textFormat` 样式，然后将其应用于 **RadioButton** 以使它们的标签样式保持相同。

将来自 CheckBox 的 textFormat 样式应用于 RadioButton:

1. 创建一个新的 Flash 文件 (ActionScript 3.0)。
2. 将一个 **CheckBox** 组件拖到舞台上，然后在“属性”检查器中为其指定实例名称“**myCh**”。
3. 将一个 **RadioButton** 拖到舞台上，然后在“属性”检查器中为其指定实例名称“**myRb**”。
4. 将以下代码添加到时间轴第一帧的“动作”面板上。

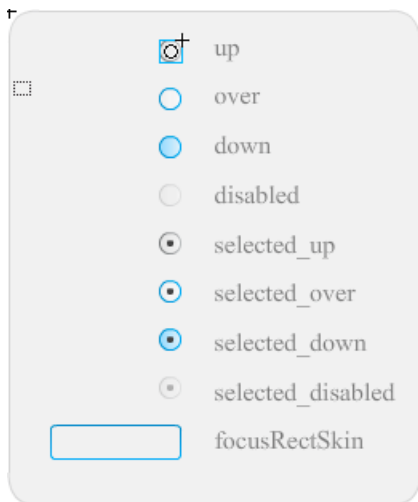
```
var tf:TextFormat = new TextFormat();
tf.color = 0x00FF00;
tf.font = "Georgia";
tf.size = 18;
myCh.setStyle("textFormat", tf);
myRb.setStyle("textFormat", myCh.getStyle("textFormat"));
```

该代码为 **CheckBox** 设置 `textFormat` 样式，然后通过对 **CheckBox** 调用 `getStyle()` 方法将其应用到 **RadioButton**。

5. 选择“控制”>“测试影片”。

对 RadioButton 使用外观

RadioButton 具有以下外观，您可以对它们进行编辑以改变其外观：



RadioButton 外观

如果 **RadioButton** 已启用但并未选中，当用户将指针移到它上方时，它会显示其 **over** 外观。用户单击 **RadioButton** 时，它将获得输入焦点并显示其 **selected_down** 外观。用户释放鼠标按键时，**RadioButton** 将显示其 **selected_up** 外观。如果用户在按住鼠标按键的同时将指针移到 **RadioButton** 的点击区域之外，**RadioButton** 将重新显示其弹起外观。

如果 **RadioButton** 被禁用，不管用户进行什么交互操作，它都会显示其禁用状态。

以下示例替换表示选中状态的 **selected_up** 外观。

创建一个新的 RadioButton selected_up 外观：

1. 创建一个新的 Flash 文件 (ActionScript 3.0)。
2. 将 **RadioButton** 组件拖到舞台中并双击它以打开其外观调色板。
3. 将缩放控制设置为 800%，以便放大图标进行编辑。
4. 双击 **selected_up** 外观以选择它，同时按下 **Delete** 键删除该外观。
5. 从“工具”面板中选择“矩形”工具。
6. 在“属性”检查器中，将线条颜色设置为红色 (#FF0000)，将填充颜色设置为黑色 (#000000)。
7. 从标记元件注册点（也称为“原点”或“零点”）的十字线开始，单击并拖动指针来绘制一个矩形。

8. 单击舞台上方编辑栏左侧的“返回”按钮，返回到文档编辑模式。

9. 选择“控制”>“测试影片”。

10. 单击 `RadioButton` 将其选中。

处于选中状态的 `RadioButton` 应显示与下图所示类似的外观。



标签

自定义 ScrollPane

在创作过程中和运行时，可以在水平和垂直方向上将 `ScrollPane` 组件变形。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改”>“变形”命令。在运行时，可使用 `setSize()` 方法或 `ScrollPane` 类的任何适用属性和方法，比如 `height`、`width`、`scaleX` 和 `scaleY`。

`ScrollPane` 组件具有以下图形特征：

- 其内容的注册点（也称为“原点”或“零点”）位于窗格的左上角。
- 当关闭水平滚动条时，垂直滚动条沿滚动窗格右侧从上到下显示。当关闭垂直滚动条时，水平滚动条沿滚动窗格底部从左到右显示。也可以同时关闭两个滚动条。
- 如果滚动窗格太小，则内容可能无法正确显示。
- 调整滚动窗格大小时，滚动轨道和滚动框（滑块）会扩展或收缩，其点击区域的大小也会进行调整。按钮的大小将保持不变。

对 ScrollPane 使用样式

`ScrollPane` 组件的样式属性在绘制组件时指定组件布局的外观和填充的值。不同的外观样式允许您指定不同的类用于组件外观。有关使用外观样式的详细信息，请参阅第 133 页的“关于外观”。

为 `ScrollPane` 设置 `contentPadding` 样式：

1. 创建一个新的 Flash 文件。
2. 将一个 `ScrollPane` 组件拖到舞台上，然后为其指定实例名称 `mySp`。
3. 单击“属性”检查器中的“参数”选项卡并为 `source` 参数输入以下值：
“<http://www.helpexamples.com/flash/images/image1.jpg>”。

4. 在主时间轴的第 1 帧中，将以下代码添加到 “动作” 面板中。

```
mySp.setStyle("contentPadding", 5);
```

请注意，在滚动条之外，组件边框及其内容之间的部分会应用填充。

5. 选择 “控制” > “测试影片”。

对 ScrollPane 使用外观

ScrollPane 组件使用边框和滚动条滚动资源。有关对滚动条应用外观的信息，请参阅第 173 页的 “对 **UIScrollBar** 使用外观”。

自定义 Slider

在创作过程中和运行时，可以在水平方向上将 **Slider** 组件变形。在创作时，在舞台上选择组件并使用 “任意变形” 工具或任何 “修改” > “变形” 命令。在运行时，可使用 `setSize()` 方法或 **Slider** 类的任何适用属性，如 `width` 和 `scaleX` 属性。

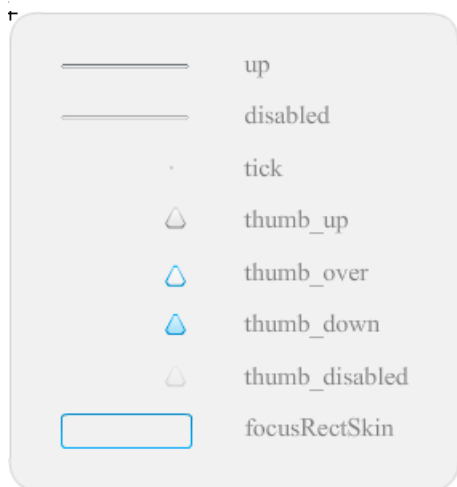
您只能使 **Slider** 组件更长一些。不能增加其高度。**Flash** 将忽略 `height` 属性及 `setSize()` 方法的 `height` 参数。但是，您可以创建垂直滑块并使其在垂直方向更长一些。

对 Slider 使用样式

Slider 组件的样式仅指定了其外观的类并为 `FocusRectPadding` 指定了一个值，该值指定在组件边框及其外部边界之间进行填充所用的像素数。有关使用外观样式的详细信息，请参阅第 133 页的 “关于外观”。

对 Slider 使用外观

Slider 组件使用以下外观，您可以对它们进行编辑以改变其外观。



Slider 外观

以下示例编辑 **up** 轨道以将其颜色改为蓝色。

更改 Slider 的 **up** 轨道的颜色：

1. 创建一个新的 Flash 文件 (ActionScript 3.0)。
2. 将 Slider 组件从“组件”面板拖到舞台中。
3. 双击 Slider 组件，打开其外观面板。
4. 双击 **up** 轨道上的注册标记以元件编辑模式打开它。
5. 将缩放控制设置为 800%，以便放大图标进行编辑。请注意，Slider 的轨道有三个栏组成。
6. 单击顶部栏以将其选中。选中后，该栏的颜色将显示在“属性”检查器的“填充颜色选择器”中。
7. 使用“属性”检查器中的“填充颜色选择器”，选择颜色 #000066 以将其应用到 Slider 轨道的顶部栏。
8. 单击 Slider 轨道的中间栏以将其选中。选中后，该栏的颜色将显示在“属性”检查器的“填充颜色选择器”中。
9. 使用“属性”检查器中的“填充颜色选择器”，选择颜色 #0066FF 以将其应用到 Slider 轨道的中间栏。

10. 单击 **Slider** 轨道的底部栏以将其选中。选中后，该栏的颜色将显示在“属性”检查器的“填充颜色选择器”中。
11. 使用“属性”检查器中的“填充颜色选择器”，选择颜色 **#00CCFF** 以将其应用到 **Slider** 轨道的底部栏。
12. 单击舞台上方编辑栏左侧的“返回”按钮，返回到文档编辑模式。
13. 选择“控制” > “测试影片”。

Slider 的外观应如下图所示。



自定义 TextArea

在创作过程中和运行时，可以在水平和垂直方向上将 **TextArea** 组件变形。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改” > “变形”命令。在运行时，可使用 `setSize()` 方法或任何适用的属性，如 **TextArea** 类的 `height`、`width`、`scaleX` 和 `scaleY`。

调整了 **TextArea** 组件的大小后，边框大小就会调整到新的边框。如果需要滚动条，它们会被放置在下边缘和右边缘。然后，文本区域的大小会在其余区域内进行调整；**TextArea** 组件中没有大小固定的元素。如果 **TextArea** 组件的宽度太窄而无法显示文本的大小，将对文本进行裁剪。

对 TextArea 使用样式

TextArea 组件的样式指定在绘制组件时组件的外观、填充和文本格式的值。`textFormat` 和 `disabledTextFormat` 样式控制 **TextArea** 显示的文本的样式。有关外观样式属性的详细信息，请参阅第 167 页的“对 **TextArea** 使用外观”。

以下示例设置 `disabledTextFormat` 样式以更改 **TextArea** 禁用时文本的外观，而这一过程也可用于为处于启用状态的 **TextArea** 设置 `textFormat` 样式。

为 **TextArea** 设置 `disabledTextFormat` 样式：

1. 创建一个新的 **Flash** 文件。
2. 将 **TextArea** 组件拖到舞台上，然后为其指定实例名称 **myTa**。
3. 将以下代码添加到主时间轴第 1 帧的“动作”面板上。

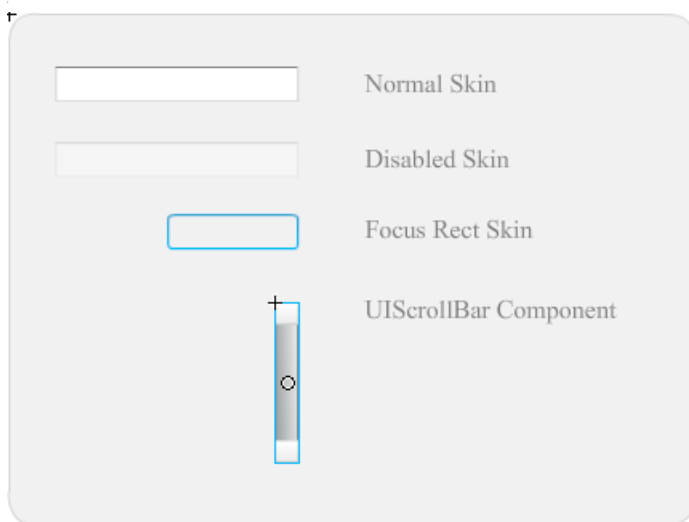
```
var tf:TextFormat = new TextFormat();
tf.color = 0xCC99FF;
tf.font = "Arial Narrow";
```

```
tf.size = 24;
myTa.setStyle("disabledTextFormat", tf);
myTa.text = "Hello World";
myTa.setSize(120, 50);
myTa.move(200, 50);
myTa.enabled = false;
```

4. 选择“控制” > “测试影片”。

对 TextArea 使用外观

TextArea 组件使用以下外观，您可以编辑这些外观来改变组件的外观。



TextArea 外观



如果更改一个组件的 ScrollBar 外观，将会更改使用 ScrollBar 的所有其它组件的 ScrollBar 外观。

以下过程更改焦点矩形外观（在 TextArea 具有焦点时显示）和 Normal 外观的边框颜色。

更改 TextArea 边框的颜色：

1. 创建一个新的 Flash 文件。
2. 将一个 TextArea 组件拖到舞台上，并双击该组件以打开其外观图标面板。
3. 双击“焦点矩形外观”。
4. 单击焦点矩形外观的边框将其选中。选中后，其当前颜色将显示在“属性”检查器的“填充颜色选择器”中。

5. 单击“属性”检查器的“填充颜色选择器”将其打开，并选择颜色 #CC0000 以将其应用于边框。
6. 单击舞台上方编辑栏左侧的“返回”按钮，返回到文档编辑模式。
7. 双击 `TextArea` 组件以打开其外观图标面板。
8. 双击 `Normal` 外观。
9. 选择 `Normal` 外观边框的每个边缘（一次选择一个），并将其颜色设置为 #990099。
10. 单击舞台上方编辑栏左侧的“返回”按钮，返回到文档编辑模式。
11. 选择“控制” > “测试影片”。

当您选中 `TextArea` 并输入文本时，其边框应按下图所示：



外侧边框为焦点矩形外观，内侧边框为 `Normal` 外观的边框。

有关编辑 `UIScrollBar` 外观的信息，请参阅第 172 页的“自定义 `UIScrollBar`”。

自定义 `TextInput`

在创作过程中和运行时，可以更改 `TextInput` 实例的大小。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改” > “变形”命令。在运行时，可使用 `setSize()` 方法或 `TextInput` 类的适用属性，如 `height`、`width`、`scaleX` 和 `scaleY`。

在调整 `TextInput` 组件大小时，边框将相应调整为新边框。`TextInput` 组件不使用滚动条，但当用户与文本交互操作时插入点会自动滚动。然后在剩余区域中调整文本字段大小，在 `TextInput` 组件中没有固定大小的元素。如果 `TextInput` 组件太小而无法显示文本，则该文本将会被裁剪。

对 `TextInput` 使用样式

`TextInput` 组件的样式指定在绘制组件时组件的外观、填充和文本格式的值。`texFormat` 和 `disabledTextFormat` 样式控制在组件中显示的文本的样式。有关外观样式属性的详细信息，请参阅第 169 页的“对 `TextInput` 使用外观”。

以下示例设置 `texFormat` 样式以设置在 `TextInput` 组件中显示的文本的字体、大小和颜色。这一过程也适用于设置在组件被禁用时应用的 `disabledTextFormat` 样式。

为 TextInput 实例设置 textFormat 样式：

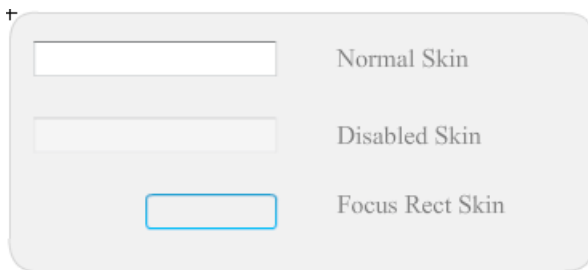
1. 创建一个新的 Flash 文件。
2. 将 TextInput 组件拖到舞台上，然后为其指定实例名称 **myTi**。
3. 将以下代码添加到主时间轴第 1 帧的“动作”面板上。

```
var tf:TextFormat = new TextFormat();
tf.color = 0x0000FF;
tf.font = "Verdana";
tf.size = 30;
tf.align = "center";
tf.italic = true;
myTi.setStyle("textFormat", tf);
myTi.text = "Enter your text here";
myTi.setSize(350, 50);
myTi.move(100, 50);
```

4. 选择“控制” > “测试影片”。

对 TextInput 使用外观

TextInput 组件使用以下外观，您可以编辑这些外观来改变组件的外观：



TextInput 标题

以下过程更改 TextInput 组件的边框和背景颜色：

更改 TextInput 组件边框和背景的颜色：

1. 创建一个新的 Flash 文件。
2. 将一个 TextInput 组件拖到舞台上，并双击该组件以打开其外观面板。
3. 双击 Normal 外观。
4. 将缩放控制设置为 800%，以便放大图标进行编辑。
5. 选择 Normal 外观边框的每个边缘（一次选择一个），并将其颜色设置为 #993399 以应用它。

6. 双击背景，直至其颜色显示在“属性”检查器的“填充颜色选择器”中。选择颜色 #99CCCC 以将其应用于背景。
7. 单击舞台上方编辑栏左侧的“返回”按钮，返回到文档编辑模式。
8. 选择“控制” > “测试影片”。

TextInput 组件的外观应如下图所示：



自定义 TileList

在创作过程中和运行时，可以在水平方向和垂直方向上将 **TileList** 组件变形。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改” > “变形”命令。在运行时，可使用 `setSize()` 方法或适当的属性，如 `width`、`height`、`columnCount`、`rowCount`、`scaleX` 和 `scaleY` 属性。**TileList** 包含的 **ScrollBar** 随列表框一同缩放。

对 TileList 使用样式

TileList 的样式指定在绘制组件时组件的外观、填充和文本格式的值。`textFormat` 和 `disabledTextFormat` 样式控制在组件中显示的文本的样式。有关外观样式的详细信息，请参阅第 171 页的“对 **TileList** 使用外观”。

下面的示例使用 `textFormat` 样式调用 `setRendererStyle()` 方法，设置在 **TileList** 实例中显示的标签的字体、大小、颜色和文本属性。这一过程也可用于设置当 `enabled` 属性设置为 `false` 时应用的 `disabledTextFormat` 样式。

为 **TileList** 实例设置 `textFormat` 样式：

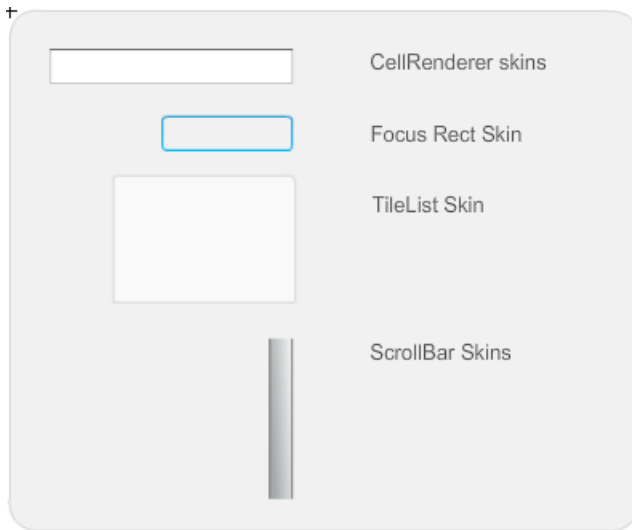
1. 创建一个新的 Flash 文件 (ActionScript 3.0)。
2. 将 **TileList** 组件拖动到舞台上，并为其指定实例名称 **myTl**。
3. 将以下代码添加到时间轴第一帧的“动作”面板上。

```
myTl.setSize(100, 100);
myTl.addItem({label:"#1"});
myTl.addItem({label:"#2"});
myTl.addItem({label:"#3"});
myTl.addItem({label:"#4"});
var tf:TextFormat = new TextFormat();
tf.font = "Arial";
tf.color = 0x00FF00;
tf.size = 16;
tf.italic = true;
tf.bold = true;
```

```
tf.underline = true;
tf.align = "center";
myTl.setRendererStyle("textFormat", tf);
```

对 TileList 使用外观

TileList 组件具有 TileList 外观、CellRenderer 外观和 ScrollBar 外观。您可以编辑这些外观来更改 TileList 的外观：



TileList 外观



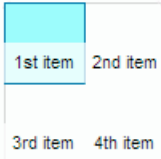
如果更改一个组件的 ScrollBar 外观，将会更改使用 ScrollBar 的所有其它组件中的 ScrollBar 外观。

以下过程更改 TileList 的 CellRenderer Selected_Up 外观的颜色。

更改 TileList 的 CellRenderer 外观的颜色：

1. 创建一个 Flash 文件 (ActionScript 3.0)。
2. 将 TileList 组件拖到舞台上，并双击该组件以打开其外观面板。
3. 双击 CellRenderer 外观，再双击 Selected_Up 外观，然后单击矩形背景。
4. 使用“属性”检查器中的“填充颜色选择器”选择颜色 #99FFFF 以将其应用于 Selected_Up 外观。
5. 单击舞台上编辑栏左侧的“返回”按钮，直至返回到文档编辑模式。

- 在“属性”检查器的“参数”选项卡上，双击 `dataProvider` 行的第二列以打开“值”对话框。添加具有以下标签的项目：`1st item`、`2nd item`、`3rd item`、`4th item`。
- 选择“控制”>“测试影片”。
- 单击 `TileList` 中的一个单元格将其选中，然后将鼠标移离选中的单元格。
选中的单元格的外观应如下图所示：



自定义 UILoader

在创作过程中和运行时，可以在水平和垂直方向上将 `UILoader` 组件变形。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改”>“变形”命令。在运行时，可使用 `setSize()` 方法或适当的属性，如 `width`、`height`、`scaleX` 和 `scaleY` 属性。

`UILoader` 组件的调整大小行为由 `scaleContent` 属性控制。当 `scaleContent` 为 `true` 时，内容会进行缩放以适合加载器的边界（并在调用 `setSize()` 时重新进行缩放）。当 `scaleContent` 为 `false` 时，组件的大小固定为内容的大小，并且 `setSize()` 和调整大小属性都会失去作用。

`UILoader` 组件没有可以向其应用样式或外观的用户界面元素。

自定义 UIScrollBar

在创作过程中和运行时，可以在水平和垂直方向上将 `UIScrollBar` 组件变形。但垂直 `UIScrollBar` 不允许您修改宽度，而水平 `UIScrollBar` 不允许您修改高度。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改”>“变形”命令。在运行时，可使用 `setSize()` 方法或 `UIScrollBar` 类的任何适用属性，如 `width`、`height`、`scaleX` 和 `scaleY` 属性。

提醒

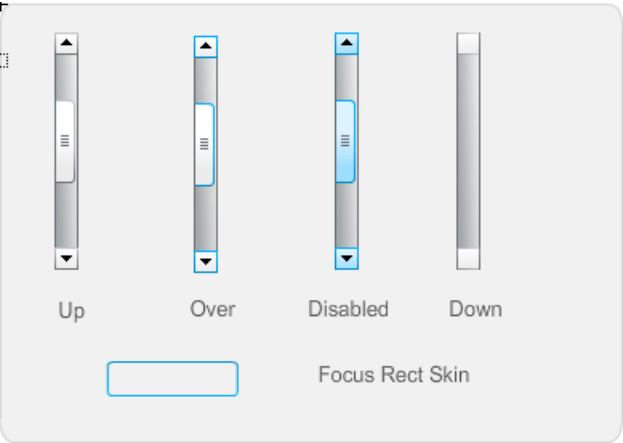
如果使用 `setSize()` 方法，则只能更改水平滚动条的宽度或垂直滚动条的高度。在创作时，您可以设置水平滚动条的高度或垂直滚动条的宽度，但影片发布时将重置这些值。只有对应滚动条长度的尺寸能够改变。

对 UIScrollBar 使用样式

UIScrollBar 组件的样式仅指定外观的类并为 FocusRectPadding 指定一个值，该值指定了在组件的边框及其外边界之间进行填充所使用的像素数。有关使用外观样式的详细信息，请参阅第 133 页的“关于外观”。

对 UIScrollBar 使用外观

UIScrollBar 组件使用以下外观。



UIScrollBar 外观

水平和垂直滚动条都使用相同的外观；在显示水平滚动条时，UIScrollBar 组件会相应地旋转外观。

提醒 如果更改一个组件的 ScrollBar 外观，将会更改使用 ScrollBar 的所有其它组件中的 ScrollBar 外观。

下面的示例演示如何更改 UIScrollBar 的滑块和箭头按钮的颜色。

更改 UIScrollBar 外观的颜色：

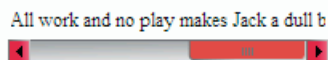
1. 创建一个新的 Flash 文件 (ActionScript 3.0)。
2. 将 UIScrollBar 组件拖到舞台上，并为其指定实例名称 **mySb**。在“参数”选项卡中，将方向设置为水平。
3. 双击滚动条以打开其外观面板。
4. 单击“弹起”外观将其选中。

5. 将缩放控制设置为 400%，以便放大图标进行编辑。
6. 双击向右箭头的背景（或垂直滚动条的向上箭头），直至背景被选中且其颜色显示在“属性”检查器的“填充颜色选择器”中。
7. 选择颜色 #CC0033 以将其应用于按钮背景。
8. 单击舞台上方编辑栏左侧的“返回”按钮，直至返回到文档编辑模式。
9. 对于滑块和向左箭头（或垂直滚动条的向下箭头）元素，重复步骤 6、7 和 8。
10. 将以下代码添加到时间轴第 1 帧的“动作”面板以将滚动条附加到某个 TextField。

```
var tf:TextField = new TextField();
addChild(tf);
tf.x = 150;
tf.y = 100;
mySb.width = tf.width = 200;
tf.height = 22;
tf.text = "All work and no play makes Jack a dull boy. All work and no
    play makes Jack a dull boy. All . . .";
mySb.y = tf.y + tf.height;
mySb.x = tf.x + tf.width;x
mySb.scrollTarget = tf;
```

11. 选择“控制” > “测试影片”。

UIScrollBar 组件的外观应如下图所示。



使用 FLVPlayback 组件

通过 FLVPlayback 组件，您可以轻松地将视频播放器包括在 Adobe Flash CS3 Professional 应用程序中，以便播放通过 HTTP 渐进式下载的 Adobe Flash 视频 (FLV) 文件，或者播放来自 Adobe 的 Macromedia Flash Media Server 或 Flash Video Streaming Service (FVSS) 的 FLV 流文件。

易于使用的 FLVPlayback 组件具有以下特性和优点：

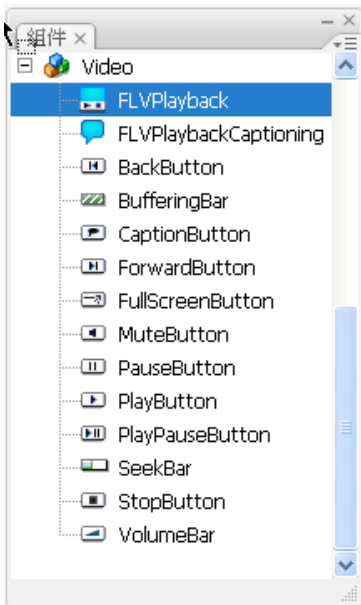
- 可拖到舞台并顺利地快速实现
- 支持全屏大小
- 提供预先设计的外观集合，这些外观可用于自定义其播放控件的外观
- 允许您为预先设计的外观选择颜色和 Alpha 值
- 允许高级用户创建他们自己的外观
- 在创作过程中提供实时预览
- 提供布局属性，以便在调整大小时使 FLV 文件保持居中
- 允许在下载足够的渐进式下载 FLV 文件时开始回放
- 提供可用于将视频与文本、图形和动画同步的提示点
- 保持合理大小的 SWF 文件

使用 FLVPlayback 组件

FLVPlayback 组件的使用过程基本上由两个步骤组成：第一步是将该组件放置在舞台上，第二步是指定一个供它播放的 FLV 文件。除此之外，您还可以设置不同的参数，以控制其行为并描述 FLV 文件。

FLVPlayback 组件还包括一个 ActionScript 应用程序编程接口 (API)。API 包括以下类，[《ActionScript 3.0 语言和组件参考》](#) 中对这些类进行了完整介绍：[CuePointType](#)、[FLVPlayback](#)、[FLVPlaybackCaptioning](#)、[NCManager](#)、[NCManagerNative](#)、[VideoAlign](#)、[VideoError](#)、[VideoPlayer](#)、[VideoState](#) 以及若干个事件类 — [AutoLayoutEvent](#)、[LayoutEvent](#)、[MetadataEvent](#)、[SkinErrorEvent](#)、[SoundEvent](#)、[VideoEvent](#) 和 [VideoProgressEvent](#)。

FLVPlayback 组件包括 FLV 回放自定义用户界面组件。FLVPlayback 组件是显示区域（或视频播放器）的组合，从中可以查看 FLV 文件以及允许您对该文件进行操作的控件。FLV 回放自定义用户界面组件提供控制按钮和机制，可用于播放、停止、暂停 FLV 文件以及对该文件进行其它控制。这些控件包括 BackButton、BufferingBar、CaptionButton（用于 FLVPlaybackCaptioning）、ForwardButton、FullScreenButton、MuteButton、PauseButton、PlayButton、PlayPauseButton、SeekBar、StopButton 和 VolumeBar。FLVPlayback 组件和 FLV 回放自定义用户界面控件显示在“组件”面板中，如下图所示：



“组件”面板中的 FLVPlayback 组件

用于将回放控件添加到 FLVPlayback 组件的过程称作 *外观设置*。FLVPlayback 组件具有一个初始默认外观 SkinOverAll.swf，它提供了播放、停止、后退、快进、搜索栏、静音、音量、全屏和字幕控件。若要更改此外观，您具有以下选择：

- 从预先设计的外观集合中进行选择
- 创建自定义外观并将该外观添加到预先设计的外观集合
- 从 FLV 回放自定义用户界面组件中选择各个控件并对它们进行自定义

选择预先设计的外观时，您可以在创作过程中或在运行时单独选择外观颜色和 Alpha 值。有关详细信息，请参阅第 194 页的“选择预先设计的外观”。

在选择了不同外观后，所选的外观将成为新的默认外观。

有关选择或创建 FLVPlayback 组件外观的详细信息，请参阅第 193 页的“自定义 FLVPlayback 组件”。

创建具有 FLVPlayback 组件的应用程序

您可以通过以下方式将 FLVPlayback 组件包括在您的应用程序中：

- 将 FLVPlayback 组件从“组件”面板拖到舞台上，并且为 source 参数指定一个值。
- 使用视频导入向导在舞台上创建组件，并通过选择外观来自定义该组件。
- 使用 FLVPlayback() 构造函数动态地在舞台上创建一个 FLVPlayback 实例，并且假定该组件位于库中。

提醒

如果您用 ActionScript 创建 FLVPlayback 实例，则还必须用 ActionScript 设置 skin 属性以便为其指定一个外观。如果通过这种方式应用外观，则外观不会自动随 SWF 文件一同发布。必须将应用程序 SWF 文件和外观 SWF 文件复制到您的应用程序服务器，否则在您运行该应用程序时将没有外观 SWF 文件可用。

从“组件”面板中拖出 FLVPlayback 组件：

1. 在“组件”面板中，单击加号 (+) 按钮打开视频条目。
2. 将 FLVPlayback 组件拖到舞台上。
3. 在选中了舞台上的 FLVPlayback 组件的情况下，在“组件”检查器的“参数”选项卡上找到 source 参数的 Value 单元格，并输入指定以下内容之一的字符串：
 - 指向 FLV 文件的本地路径
 - 指向 FLV 文件的 URL
 - 指向某个同步多媒体集成语言 (SMIL) 文件的 URL，该文件说明如何播放 FLV 文件有关如何创建 SMIL 文件以描述一个或多个 FLV 文件的信息，请参阅第 207 页的“使用 SMIL 文件”。
4. 在选中舞台上的 FLVPlayback 组件的情况下，在“组件”检查器的“参数”选项卡上单击 skin 参数的 Value 单元格。
5. 单击放大镜图标以打开“选择外观”对话框。
6. 选择以下选项之一：
 - 从“外观”下拉列表中，选择一种预先设计的外观以将一组回放控件附加到组件中。
 - 如果已创建了一种自定义外观，可从下拉菜单中选择“自定义外观 URL”，然后在“URL”框中输入包含此外观的 SWF 文件的 URL。
 - 选择“无”，然后将单个 FLV 回放自定义用户界面组件拖到舞台上，以添加回放控件。

提醒

在前两种情况下，会在弹出菜单上部的查看窗格中显示外观的预览效果。您可以使用颜色选择器更改外观的颜色。
若要更改自定义用户界面控件的颜色，必须自定义该控件。有关使用自定义用户控件的详细信息，请参阅第 195 页的“分别在各个 FLV 回放自定义用户界面组件内设置外观”。

7. 单击“确定”以关闭“选择外观”对话框。
8. 选择“控制”>“测试影片”，以执行 SWF 文件并启动视频。

以下过程使用视频导入向导添加 FLVPlayback 组件：

使用视频导入向导：

1. 选择“文件”>“导入”>“导入视频”。
2. 通过选择以下选项之一，指示视频文件的位置：
 - 在我的本地计算机上
 - 已经部署到 Web 服务器、Flash Video Streaming Service 或 Flash Media Server
3. 根据您的选择，输入指定视频文件位置的路径或 URL；然后单击“下一步”。
4. 如果您选择了文件路径，则会在旁边看到“部署”对话框，可以从中选择列出的选项之一，以指定视频的部署方式：
 - 从标准 Web 服务器渐进式下载
 - 从 Flash Video Streaming Service 流式加载
 - 从 Flash Media Server 流式加载
 - 在 SWF 文件中嵌入视频并在时间轴中播放



不要选择“嵌入视频”选项。FLVPlayback 组件仅播放外部视频流。此选项不将 FLVPlayback 组件放置于舞台上。

5. 单击“下一步”。
6. 选择以下选项之一：
 - 从“外观”下拉列表中，选择一种预先设计的外观以将一组回放控件附加到组件中。
 - 如果已经为该组件创建了一种自定义外观，则可从下拉菜单中选择“自定义外观 URL”，然后在“URL”框中输入包含此外观的 SWF 文件的 URL。
 - 选择“无”，然后将单个 FLV 回放自定义用户界面组件拖到舞台上，以添加回放控件。



在前两种情况下，会在弹出菜单上部的查看窗格中显示外观的预览效果。

7. 单击“确定”以关闭“选择外观”对话框。
8. 阅读“完成视频导入”对话框中的内容，注意下一步的操作，然后单击“完成”。
9. 如果您尚未保存 FLA 文件，将出现“另存为”对话框。
10. 选择“控制”>“测试影片”，以执行 SWF 文件并启动视频。

以下过程将使用 ActionScript 添加 FLVPlayback 组件。

使用 ActionScript 动态创建实例：

1. 将 FLVPlayback 组件从“组件”面板拖到“库”面板（“窗口” > “库”）中。
2. 将以下代码添加到时间轴第一帧的“动作”面板上。将 *install_drive* 更改为安装了 Flash 的驱动器，并修改路径以反映您的系统中 **Skins** 文件夹的位置：

在 Windows 计算机上：

```
import fl.video.*;
var my_FLVPlayback = new FLVPlayback();
my_FLVPlayback.x = 100;
my_FLVPlayback.y = 100;
addChild(my_FLVPlayback);
my_FLVPlayback.skin = "file:///install_drive|Program Files/Adobe/Adobe
Flash CS3/en/Configuration/FLVPlayback Skins/ActionScript 3.0/
SkinOverPlaySeekMute.swf"
my_FLVPlayback.source = "http://www.helpexamples.com/flash/video/water.flv";
```

在 Macintosh 计算机上：

```
import fl.video.*;
var my_FLVPlayback = new FLVPlayback();
my_FLVPlayback.x = 100;
my_FLVPlayback.y = 100;
addChild(my_FLVPlayback);
my_FLVPlayback.skin = "file:///Macintosh HD:Applications:Adobe Flash
CS3:Configuration:FLVPlayback Skins:ActionScript
3.0SkinOverPlaySeekMute.swf"
my_FLVPlayback.source = "http://www.helpexamples.com/flash/video/water.flv";
```



如果不设置 `source` 和 `skin` 属性，则生成的影片剪辑将显示为空。

3. 选择“控制” > “测试影片”，以执行 SWF 文件并启动 FLV 文件。

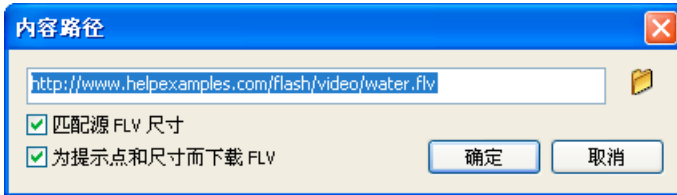
FLVPlayback 组件参数

对于 FLVPlayback 组件的每个实例，您可以在“组件”检查器或“属性”检查器中设置以下参数：`align`、`autoPlay`、`cuePoints`、`preview`、`scaleMode`、`skin`、`skinAutoHide`、`skinBackgroundAlpha`、`skinBackgroundColor`、`source` 和 `volume`。其中每个参数都有对应的同名 **ActionScript** 属性。为这些参数赋值时，将设置应用程序中属性的初始状态。在 **ActionScript** 中设置的属性会覆盖在对应参数中设置的值。有关这些参数的可能值的信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 FLVPlayback 类。

指定 source 参数

source 参数允许您指定 FLV 文件的名称和位置，这两项可指示 Flash 如何播放该文件。

通过在“组件”检查器中双击 source 参数的 Value 单元格，可以打开“内容路径”对话框。该对话框如下所示：




FLVPlayback 的“内容路径”对话框

该对话框提供两个复选框，可用于确定 FLVPlayback 实例的尺寸并指定是否从 FLV 文件获取尺寸和提示点信息。有关详细信息，请参阅第 181 页的“FLV 文件选项”。

源

输入说明 FLV 文件播放方式的 FLV 文件或 XML 文件的 URL 或本地路径。如果您不知道 FLV 文件的确切位置，则通过单击文件夹图标打开“浏览器”对话框，可以帮助您找到正确的位置。浏览查找 FLV 文件时，如果它位于目标 SWF 文件所在位置或该位置以下，则 Flash 会自动使该路径成为相对于该位置的路径，以便通过 Web 服务器使用。否则，该路径是 Windows 或 Macintosh 绝对路径。若要输入本地 XML 文件的名称，您必须键入路径和名称。

如果指定 HTTP URL，则 FLV 文件将作为一个渐进式下载播放。如果您指定属于 RTMP URL 的某个 URL，则该 FLV 文件从 Flash Media Server 或 FVSS 以流的方式加载。指向 XML 文件的 URL 也可以是来自 Flash Media Server 或 FVSS 的 FLV 文件流。

 在“内容路径”对话框中单击“确定”后，该组件将更新 cuePoints 参数的值，因为在内容路径更改的情况下它可能不再适用。因此，您可以丢失所有禁用的提示点，但不可以丢失 ActionScript 提示点。（如果新的 FLV 文件包含相同的提示点，则将不会丢失禁用的提示点，在您只更改路径时可能会发生这一情况。）因此，您可能要通过 ActionScript 而不是通过“提示点”对话框来禁用非 ActionScript 提示点。

您也可以指定说明如何针对多个带宽播放多个 FLV 文件流的 SMIL 文件的位置。该文件使用同步多媒体集成语言 (SMIL) 来描述 FLV 文件。有关 SMIL 文件的说明，请参阅第 207 页的“使用 SMIL 文件”。

您还可以使用 **ActionScript** `FLVPlayback.source` 属性以及 `FLVPlayback.play()` 和 `FLVPlayback.load()` 方法，指定 **FLV** 文件的名称和位置。这三个替代方法优先于“组件”检查器中的 `source` 参数。有关详细信息，请参阅 [《ActionScript 3.0 语言和组件参考》](#) 中 **FLVPlayback** 类的 `FLVPlayback.source`、`FLVPlayback.play()` 和 `FLVPlayback.load()` 条目。

FLV 文件选项

“内容路径”对话框也具有两个选项。第一个选项是“匹配源 **FLV** 尺寸”，它指定舞台上的 **FLVPlayback** 实例是否应匹配源 **FLV** 文件的尺寸。源 **FLV** 文件包含用于播放的首选高、宽尺寸。如果您选择第一个选项，将调整 **FLVPlayback** 实例的尺寸以匹配这些首选尺寸。但是，只有在同时选中了第二个选项的情况下，此选项才可用。

第二个选项是“为提示点和尺寸下载 **FLV**”，只有在内容路径是 **HTTP** 或 **RTMP URL** 的情况下才启用该选项，这意味着该 **FLV** 文件不是本地的。（任何不是以 `.flv` 结尾的路径都被认为是网络路径，因为它必须是 **XML** 文件并可能指向位于任何位置的 **FLV** 文件。）此选项指定以下内容：为了获取 **FLV** 文件尺寸以及在该文件中嵌入的任何提示点定义，是下载还是流式加载该 **FLV** 文件的某一部分。**Flash** 使用尺寸来调整 **FLVPlayback** 实例的大小，并且它将提示点定义加载到“组件”检查器的 `cuePoints` 参数中。如果未选择此选项，则禁用第一个选项。

使用实时预览

FLVPlayback preview 参数使您能够查看舞台上组件中的源 FLV 文件的帧，还可以查看您对组件所做的更改。单击 preview 参数将打开以下对话框，该对话框可播放源 FLV 文件的 SWF 文件。



用于选择实时预览帧的对话框

当 FLV 文件到达您希望捕获的场景以预览舞台上的组件时，单击“确定”。显示舞台上组件中的 FLV 文件帧使您能够看到舞台上的该帧与应用程序的其它元素的关系。

您还可以导出您选择的帧以将其作为 PNG（便携网络图形）文件保存在您选择的位置。

全屏支持

FLVPlayback 的 ActionScript 3.0 版本支持全屏模式，该模式需要 Flash Player 9.0.28.0，还需要针对全屏查看正确设置 HTML。某些预先设计的外观包括一个切换按钮，用于打开和关闭全屏模式。FullScreenButton 显示在下图中控制栏的右侧。



控制栏上的全屏图标

全屏模式支持包括以下属性：`fullScreenBackgroundColor`、`fullScreenSkinDelay` 和 `fullScreenTakeOver`。有关这些属性的信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》。

用于播放多个 FLV 文件的布局对齐方式

ActionScript 3.0 FLVPlayback 具有一个 `align` 属性，该属性指定在调整大小时 FLV 文件应该居中还是置于组件的顶部、底部、左侧或右侧。除了组件的 `x`、`y`、`width` 和 `height` 属性外，**ActionScript 3.0** 组件还具有 `registrationX`、`registrationY`、`registrationWidth` 和 `registrationHeight` 属性。最初，这些属性与 `x`、`y`、`width` 和 `height` 属性相一致。加载后续的 FLV 文件时，自动进行的重新布局不会更改这些属性，因此新的 FLV 文件可以在同一位置居中。如果 `scaleMode = VideoScaleMode.MAINTAIN_ASPECT_RATIO`，则后续的 FLV 文件可以适合组件的原始尺寸，而不是导致组件的宽度和高度发生变化。

自动播放渐进式下载的 FLV 文件

加载渐进式下载的 FLV 文件时，只有下载了足够的 FLV 文件之后，**FLVPlayback** 才会开始播放该 FLV 文件，这样可以从头至尾地播放 FLV 文件。

如果您要在下载了足够的 FLV 文件之前播放该 FLV 文件，请调用 `play()` 方法（不带任何参数）。

如果您要返回到等待下载足够的 FLV 文件的状态，请调用 `pause()` 方法，然后再调用 `playWhenEnoughDownloaded()` 方法。

使用提示点

提示点是一个点，在播放 FLV 文件时，视频播放器在该点处调度一个 `cuePoint` 事件。您可以在想为网页上的其它元素执行动作时向 FLV 文件添加提示点。例如，您可能想要显示文本或图形，或者要与 **Flash** 动画同步，或者要影响 FLV 文件的播放，具体为暂停 FLV 文件、在视频中搜索不同点或切换到不同 FLV 文件。提示点允许您接收 **ActionScript** 代码中的控制以将 FLV 文件中的点与网页上的其它动作同步。

有三种类型的提示点：导航、事件和 **ActionScript**。导航提示点和事件提示点也称作 *嵌入式* 提示点，因为它们嵌入在 FLV 文件流和 FLV 文件的元数据包中。

“导航提示点”允许您搜索 FLV 文件中的特定帧，因为它在尽可能接近您指定的时间在 FLV 文件内创建关键帧。“关键帧”是在 FLV 文件流的图像帧之间出现的数据段。在您搜索导航提示点时，组件将搜索到该关键帧并启动 `cuePoint` 事件。

事件提示点 使您能够将 FLV 文件内的时间点与网页上的外部事件同步。`cuePoint` 事件在指定的时间精确发生。您可以使用视频导入向导或 **Flash Video Encoder**，在 FLV 文件中嵌入导航提示点和事件提示点。有关视频导入向导和 **Flash Video Encoder** 的详细信息，请参阅《[使用 Flash](#)》中的第 16 章，“使用视频”。

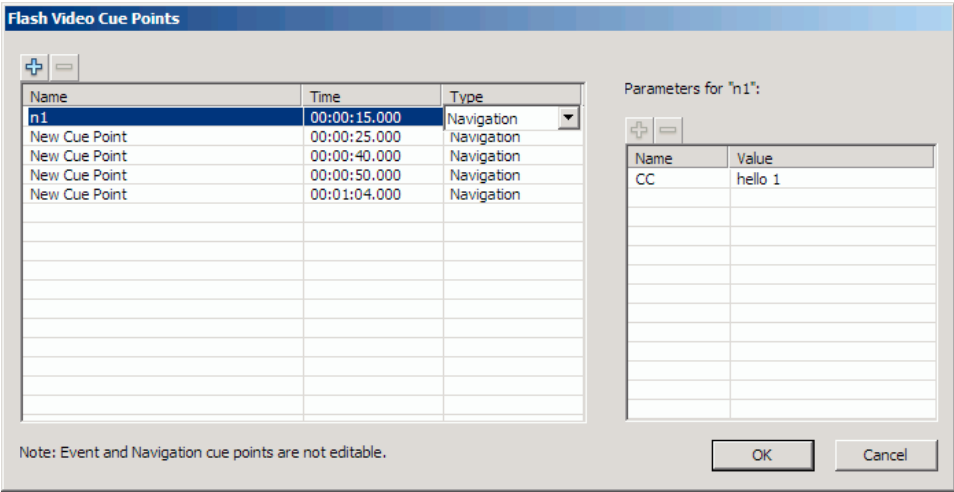
ActionScript 提示点 是一种外部提示点，您可以通过组件的“Flash 视频提示点”对话框或通过 `FLVPlayback.addASCuePoint()` 方法添加。该组件在 FLV 文件之外存储和跟踪 *ActionScript* 提示点，因此，这些提示点在精确性上要低于嵌入式提示点。*ActionScript* 提示点精确度为十分之一秒。您可以通过降低 `playheadUpdateInterval` 属性值来提高 *ActionScript* 提示点的精确度，因为组件在播放头更新时会为 *ActionScript* 提示点生成 `cuePoint` 事件。有关详细信息，请参阅《*ActionScript 3.0 语言和组件参考*》中的 `FLVPlayback.playheadUpdateInterval` 属性。

在 *ActionScript* 和 FLV 文件的元数据中，提示点表示为带有以下属性的对象：`name`、`time`、`type` 和 `parameters`。`name` 属性是一个字符串，其中包含为提示点分配的名称。`time` 属性是一个数字，表示提示点的发生时间，用小时、分钟、秒和毫秒 (HH:MM:SS.mmm) 表示。`type` 属性是一个字符串，根据您创建的提示点的类型，此字符串可以是“`navigation`”、“`event`”或“`actionscript`”。`parameters` 属性是包含指定的名称 - 值的数组。

在发生 `cuePoint` 事件时，可以通过 `info` 属性在事件对象中提供提示点对象。有关详细信息，请参阅第 186 页的“侦听 `cuePoint` 事件”。

使用“Flash 视频提示点”对话框

通过在“组件”检查器中双击 `cuePoints` 参数的 `Value` 单元格，可以打开“Flash 视频提示点”对话框。该对话框类似于下图：



提示点对话框

该对话框显示嵌入式提示点和 *ActionScript* 提示点。可以使用此对话框添加和删除 *ActionScript* 提示点以及提示点参数。还可以启用或禁用嵌入式提示点。但是，不能添加、更改或删除嵌入的提示点。

添加 ActionScript 提示点：

1. 双击“组件”检查器中 `cuePoints` 参数的 **Value** 单元格以打开“Flash 提示点”对话框。
2. 单击在提示点列表之上、位于左上角的加号 (+)，以添加默认的 **ActionScript** 提示点条目。
3. 单击“名称”列中的“新提示点”文本，并编辑该文本以命名提示点。
4. 单击“时间”值 `00:00:00:000` 以编辑它，并且为要发生的提示点分配一个时间。您可以按小时、分钟、秒和毫秒 (HH:MM:SS.mmm) 指定该时间。
如果存在多个提示点，该对话框会根据时间顺序，将新提示点移到列表中相应的位置。
5. 若要为所选提示点添加参数，请单击“参数”部分之上的加号 (+)，然后在“名称”列和“值”列中输入值。对每个参数均重复此步骤。
6. 若要添加多个 **ActionScript** 提示点，请为每个提示点重复第 2 步到第 5 步。
7. 单击“确定”以保存更改。

删除 ActionScript 提示点：

1. 双击“组件”检查器中 `cuePoints` 参数的 **Value** 单元格以打开“Flash 提示点”对话框。
2. 选择您要删除的提示点。
3. 单击提示点列表之上、位于左上角的减号 (-)，以删除它。
4. 为要删除的每个提示点重复第 2 步和第 3 步。
5. 单击“确定”以保存更改。

启用或禁用嵌入的 FLV 文件提示点：

1. 双击“组件”检查器中 `cuePoints` 参数的 **Value** 单元格以打开“Flash 提示点”对话框。
2. 选择您要启用或禁用的提示点。
3. 单击“类型”列中的值以触发弹出菜单，或者单击向下箭头。
4. 单击提示点类型的名称（如“事件”或“导航”）即可启用它。单击“禁用”可以禁用它。
5. 单击“确定”以保存更改。

在 ActionScript 中使用提示点

您可以使用 **ActionScript** 添加 **ActionScript** 提示点、侦听 `cuePoint` 事件、查找任何类型或特定类型的提示点、搜索导航提示点、启用或禁用提示点、检查是否启用了某个提示点以及删除提示点。

本节中的示例使用名为 `cuepoints.flv` 的 FLV 文件，它包含以下三个提示点：

名称	时间	类型
point1	00:00:00.418	导航
point2	00:00:07.748	导航
point3	00:00:16.020	导航

添加 `ActionScript` 提示点

您可以使用 `addASCuePoint()` 方法将 `ActionScript` 提示点添加到 FLV 文件。以下示例在 FLV 文件可供播放时向该 FLV 文件添加两个 `ActionScript` 提示点。该示例使用提示点对象添加第一个提示点，在其属性中指定该提示点的时间、名称和类型。第二个调用使用该方法的时间 `time` 和名称 `name` 参数指定时间和名称。

```
import fl.video.*;
import fl.video.MetadataEvent;
my_FLVPlybk.source = "http://www.helpexamples.com/flash/video/cuepoints.flv"
var cuePt:Object = new Object(); //create cue point object
cuePt.time = 2.02;
cuePt.name = "ASpt1";
cuePt.type = "actionscript";
my_FLVPlybk.addASCuePoint(cuePt); //add AS cue point
// add 2nd AS cue point using time and name parameters
my_FLVPlybk.addASCuePoint(5, "ASpt2");
```

有关详细信息，请参阅 [《ActionScript 3.0 语言和组件参考》](#) 中的 `FLVPlayback.addASCuePoint()` 方法。

侦听 `cuePoint` 事件

`cuePoint` 事件允许您在发生 `cuePoint` 事件时接收 `ActionScript` 代码中的控制。在以下示例中，当提示点出现时，`cuePoint` 侦听器会调用一个事件处理函数，该事件处理函数显示 `playheadTime` 属性的值以及提示点的名称和类型。

```
my_FLVPlybk.addEventListener(MetadataEvent.CUE_POINT, cp_listener);
function cp_listener(eventObject:MetadataEvent):void {
    trace("Elapsed time in seconds: " + my_FLVPlybk.playheadTime);
    trace("Cue point name is: " + eventObject.info.name);
    trace("Cue point type is: " + eventObject.info.type);
}
```

有关 `cuePoint` 事件的详细信息，请参阅 [《ActionScript 3.0 语言和组件参考》](#) 中的 `FLVPlayback.cuePoint` 事件。

查找提示点

通过使用 **ActionScript**，您可以查找任何类型的提示点、查找与某个时间最接近的提示点或者查找具有特定名称的下一个提示点。

在以下示例中，`ready_listener()` 事件处理函数调用 `findCuePoint()` 方法查找提示点 `ASpt1`，然后调用 `findNearestCuePoint()` 方法查找最接近提示点 `ASpt1` 的时间的导航提示点：

```
import fl.video.FLVPlayback;
import fl.video.CuePointType;
import fl.video.VideoEvent;
my_FLVPlayback.source = "http://www.helpexamples.com/flash/video/cuepoints.flv"
var rtn_obj:Object; //create cue point object
my_FLVPlayback.addASCuePoint(2.02, "ASpt1"); //add AS cue point
function ready_listener(eventObject:VideoEvent):void {
    rtn_obj = my_FLVPlayback.findCuePoint("ASpt1", CuePointType.ACTIONSCRIPT);
    traceit(rtn_obj);
    rtn_obj = my_FLVPlayback.findNearestCuePoint(rtn_obj.time,
        CuePointType.NAVIGATION);
    traceit(rtn_obj);
}
my_FLVPlayback.addEventListener(VideoEvent.READY, ready_listener);
function traceit(cuePoint:Object):void {
    trace("Cue point name is: " + cuePoint.name);
    trace("Cue point time is: " + cuePoint.time);
    trace("Cue point type is: " + cuePoint.type);
}
```

在以下示例中，`ready_listener()` 事件处理函数查找提示点 `ASpt`，并且调用 `findNextCuePointWithName()` 方法查找具有相同名称的下一个提示点：

```
import fl.video.*;
my_FLVPlayback.source = "http://www.helpexamples.com/flash/video/cuepoints.flv"
var rtn_obj:Object; //create cue point object
my_FLVPlayback.addASCuePoint(2.02, "ASpt"); //add AS cue point
my_FLVPlayback.addASCuePoint(3.4, "ASpt"); //add 2nd ASpt
my_FLVPlayback.addEventListener(VideoEvent.READY, ready_listener);
function ready_listener(eventObject:VideoEvent):void {
    rtn_obj = my_FLVPlayback.findCuePoint("ASpt", CuePointType.ACTIONSCRIPT);
    traceit(rtn_obj);
    rtn_obj = my_FLVPlayback.findNextCuePointWithName(rtn_obj);
    traceit(rtn_obj);
}
function traceit(cuePoint:Object):void {
    trace("Cue point name is: " + cuePoint.name);
    trace("Cue point time is: " + cuePoint.time);
    trace("Cue point type is: " + cuePoint.type);
}
```

有关查找提示点的详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 [FLVPlayback.findCuePoint\(\)](#)、[FLVPlayback.findNearestCuePoint\(\)](#) 和 [FLVPlayback.findNextCuePointWithName\(\)](#) 方法。

搜索导航提示点

您可以搜索导航提示点、搜索指定时间的下一个导航提示点以及搜索指定时间的前一个导航提示点。下面的示例播放 FLV 文件 **cuepoints.flv**，并在发生 ready 事件时搜索位于 7.748 的提示点。在发生 cuePoint 事件时，该示例调用 [seekToPrevNavCuePoint\(\)](#) 方法以搜索第一个提示点。在 cuePoint 事件发生时，该示例调用 [seekToNextNavCuePoint\(\)](#) 方法，通过将 `eventObject.info.time`（这是当前提示点的时间）加上 10 秒来搜索最后一个提示点。

```
import fl.video.*;

my_FLVPlayback.addEventListener(VideoEvent.READY, ready_listener);
function ready_listener(eventObject:Object):void {
    my_FLVPlayback.seekToNavCuePoint("point2");
}
my_FLVPlayback.addEventListener(MetadataEvent.CUE_POINT, cp_listener);
function cp_listener(eventObject:MetadataEvent):void {
    trace(eventObject.info.time);
    if(eventObject.info.time == 7.748)
        my_FLVPlayback.seekToPrevNavCuePoint(eventObject.info.time - .005);
    else
        my_FLVPlayback.seekToNextNavCuePoint(eventObject.info.time + 10);
}
my_FLVPlayback.source = "http://helpexamples.com/flash/video/cuepoints.flv";
```

有关详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 [FLVPlayback.seekToNavCuePoint\(\)](#)、[FLVPlayback.seekToNextNavCuePoint\(\)](#) 和 [FLVPlayback.seekToPrevNavCuePoint\(\)](#) 方法。

启用和禁用嵌入式 FLV 文件提示点

您可以通过 [setFLVCuePointEnabled\(\)](#) 方法启用和禁用嵌入式 FLV 文件提示点。禁用的提示点不触发 `cuePoint` 事件，并且不与 [seekToCuePoint\(\)](#)、[seekToNextNavCuePoint\(\)](#) 和 [seekToPrevNavCuePoint\(\)](#) 方法一起使用。不过，您可以使用 [findCuePoint\(\)](#)、[findNearestCuePoint\(\)](#) 和 [findNextCuePointWithName\(\)](#) 方法查找禁用的提示点。

您可以使用 [isFLVCuePointEnabled\(\)](#) 方法来测试是否启用了嵌入式 FLV 文件提示点。以下示例在视频准备播放时禁用嵌入式提示点 `point2` 和 `point3`。但在发生第一个 `cuePoint` 事件时，事件处理函数将进行测试以确定提示点 `point3` 是否被禁用，如果确定该提示点被禁用，则启用它。

```

import fl.video.*;
my_FLVPlayback.source = "http://www.helpexamples.com/flash/video/
  cuepoints.flv";
my_FLVPlayback.addEventListener(VideoEvent.READY, ready_listener);
function ready_listener(eventObject:VideoEvent):void {
    my_FLVPlayback.setFLVCuePointEnabled(false, "point2");
    my_FLVPlayback.setFLVCuePointEnabled(false, "point3");
}
my_FLVPlayback.addEventListener(MetadataEvent.CUE_POINT, cp_listener);
function cp_listener(eventObject:MetadataEvent):void {
    trace("Cue point time is: " + eventObject.info.time);
    trace("Cue point name is: " + eventObject.info.name);
    trace("Cue point type is: " + eventObject.info.type);
    if (my_FLVPlayback.isFLVCuePointEnabled("point2") == false) {
        my_FLVPlayback.setFLVCuePointEnabled(true, "point2");
    }
}
}

```

有关详细信息，请参阅 [《ActionScript 3.0 语言和组件参考》](#) 中的 [FLVPlayback.isFLVCuePointEnabled\(\)](#) 和 [FLVPlayback.setFLVCuePointEnabled\(\)](#) 方法。

删除 ActionScript 提示点

您可以使用 `removeASCuePoint()` 方法删除 **ActionScript** 提示点。以下示例在提示点 ASpt1 发生时删除提示点 ASpt2：

```

import fl.video.*;
my_FLVPlayback.source = "http://www.helpexamples.com/flash/video/cuepoints.flv"
my_FLVPlayback.addASCuePoint(2.02, "ASpt1"); //add AS cue point
my_FLVPlayback.addASCuePoint(3.4, "ASpt2"); //add 2nd Aspt
my_FLVPlayback.addEventListener(MetadataEvent.CUE_POINT, cp_listener);
function cp_listener(eventObject:MetadataEvent):void {
    trace("Cue point name is: " + eventObject.info.name);
    if (eventObject.info.name == "ASpt1") {
        my_FLVPlayback.removeASCuePoint("ASpt2");
        trace("Removed cue point ASpt2");
    }
}
}

```

有关详细信息，请参阅 [《ActionScript 3.0 语言和组件参考》](#) 中的 [FLVPlayback.removeASCuePoint\(\)](#)。

播放多个 FLV 文件

您只需通过在前一个 FLV 文件完成播放时在 `source` 属性中加载新的 URL，就可以在 **FLVPlayback** 实例中连续播放多个 FLV 文件。例如，以下 **ActionScript** 代码侦听 `complete` 事件，该事件在 FLV 文件完成播放后发生。在发生此事件时，代码在 `source` 属性中设置新 FLV 文件的名称和位置，并调用 `play()` 方法以播放新视频。

```
import fl.video.*;
my_FLVPlayback.source = "http://www.helpexamples.com/flash/video/clouds.flv";
my_FLVPlayback.addEventListener(VideoEvent.COMPLETE, complete_listener);
// listen for complete event; play new FLV
function complete_listener(eventObject:VideoEvent):void {
    if (my_FLVPlayback.source == "http://www.helpexamples.com/flash/video/
        clouds.flv") {
        my_FLVPlayback.play("http://www.helpexamples.com/flash/video/water.flv");
    }
};
```

使用多个视频播放器

您还可以在 **FLVPlayback** 组件的单个实例内打开多个视频播放器，以播放多个视频并在这些视频播放时在它们之间切换。

您在将 **FLVPlayback** 组件拖到舞台上时创建初始视频播放器。该组件自动为初始视频播放器分配数字 **0**，并使其成为默认播放器。若要创建其它视频播放器，只需将 `activeVideoPlayerIndex` 属性设置为新的数字。通过设置 `activeVideoPlayerIndex` 属性，还可以使指定的视频播放器成为活动视频播放器，**FLVPlayback** 类的属性和方法将会影响该播放器。但是，设置 `activeVideoPlayerIndex` 属性并不会使视频播放器可见。若要使视频播放器可见，请将 `visibleVideoPlayerIndex` 属性设置为该视频播放器的编号。有关这些属性如何与 **FLVPlayback** 类的方法和属性交互的详细信息，请参阅《**ActionScript 3.0 语言和组件参考**》中的 `FLVPlayback.activeVideoPlayerIndex` 和 `FLVPlayback.visibleVideoPlayerIndex` 属性。

以下 **ActionScript** 代码加载 `source` 属性，以便在默认视频播放器中播放 FLV 文件并为其添加提示点。在发生 `ready` 事件时，事件处理函数通过将 `activeVideoPlayerIndex` 属性设置为数字 **1**，打开另一个视频播放器。它为第二个视频播放器指定 FLV 文件和提示点，然后再次使默认播放器 (**0**) 成为活动的视频播放器。

```
/**
 * Requires:
 *   - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
// add a cue point to the default player
import fl.video.*;
my_FLVPlayback.source = "http://www.helpexamples.com/flash/video/clouds.flv";
my_FLVPlayback.addASCuePoint(3, "1st_switch");
my_FLVPlayback.addEventListener(VideoEvent.READY, ready_listener);
```

```
function ready_listener(eventObject:VideoEvent):void {
    // add a second video player and create a cue point for it
    my_FLVPlayback.activeVideoPlayerIndex = 1;
    my_FLVPlayback.source = "http://www.helpexamples.com/flash/video/water.flv";
    my_FLVPlayback.addASCuePoint(3, "2nd_switch");
    my_FLVPlayback.activeVideoPlayerIndex = 0;
};
```

若要在播放一个 FLV 文件时切换到另一个 FLV 文件，必须在您的 **ActionScript** 代码中进行该切换。通过提示点可使用 cuePoint 事件在 FLV 文件的特定点介入。以下代码为 cuePoint 事件创建一个侦听器，并且调用用于暂停活动视频播放器 (0) 的处理函数，切换到第二个播放器 (1)，然后播放其 FLV 文件：

```
import fl.video.*;
// add listener for a cuePoint event
my_FLVPlayback.addEventListener(MetadataEvent.CUE_POINT, cp_listener);
// add the handler function for the cuePoint event
function cp_listener(eventObject:MetadataEvent):void {
    // display the no. of the video player causing the event
    trace("Hit cuePoint event for player: " + eventObject.vp);
    // test for the video player and switch FLV files accordingly
    if (eventObject.vp == 0) {
        my_FLVPlayback.pause(); //pause the first FLV file
        my_FLVPlayback.activeVideoPlayerIndex = 1; // make the 2nd player active
        my_FLVPlayback.visibleVideoPlayerIndex = 1; // make the 2nd player
        visible
        my_FLVPlayback.play(); // begin playing the new player/FLV
    } else if (eventObject.vp == 1) {
        my_FLVPlayback.pause(); // pause the 2nd FLV
        my_FLVPlayback.activeVideoPlayerIndex = 0; // make the 1st player active
        my_FLVPlayback.visibleVideoPlayerIndex = 0; // make the 1st player
        visible
        my_FLVPlayback.play(); // begin playing the 1st player
    }
}
my_FLVPlayback.addEventListener(VideoEvent.COMPLETE, complete_listener);
function complete_listener(eventObject:VideoEvent):void {
    trace("Hit complete event for player: " + eventObject.vp);
    if (eventObject.vp == 0) {
        my_FLVPlayback.activeVideoPlayerIndex = 1;
        my_FLVPlayback.visibleVideoPlayerIndex = 1;
        my_FLVPlayback.play();
    } else {
        my_FLVPlayback.closeVideoPlayer(1);
    }
};
```

在您创建新的视频播放器时，`FLVPlayback` 实例将其属性设置为默认视频播放器的值，但 `source`、`totalTime` 和 `isLive` 属性除外，`FLVPlayback` 实例始终将它们分别设置为以下默认值：空字符串、`0` 和 `false`。该实例将 `autoPlay` 属性（对于默认视频播放器，默认值为 `true`）设置为 `false`。`cuePoints` 属性不会产生任何影响，并且不会影响以后加载到默认视频播放器中。

控制音量、位置、尺寸、可见性和用户界面控件的方法和属性始终是全局的，并且设置 `activeVideoPlayerIndex` 属性后并不影响它们的行为。有关这些方法和属性以及设置 `activeVideoPlayerIndex` 属性后的影响的详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 `FLVPlayback.activeVideoPlayerIndex` 属性。其余的属性和方法针对 `activeVideoPlayerIndex` 属性值所标识的视频播放器。

但是，控制尺寸的属性和方法与 `visibleVideoPlayerIndex` 属性会 *相互影响*。有关详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 `FLVPlayback.visibleVideoPlayerIndex` 属性。

从 Flash Media Server 流式加载 FLV 文件

从 Flash Media Server 流式加载 FLV 文件的要求会有所不同，具体取决于 Flash Video Streaming Service 供应商是否提供本机带宽检测。本机带宽检测的意思是带宽检测功能内置在流服务器中，从而提高了性能。请与供应商核实，以确定其是否提供本机带宽检测。

若要访问 Flash Media Server 上的 FLV 文件，请使用 `rtmp://my_servername/my_application/stream.flv` 之类的 URL。

在使用 Flash Media Server 播放实时流时，需要将 `FLVPlayback` 属性 `isLive` 设置为 `true`。有关详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 `FLVPlayback.isLive` 属性。

有关管理 Flash Media Server 的详细信息，包括如何设置实时流，请参阅位于 www.adobe.com/support/documentation/en/flashmediaserver/ 的 Flash Media Server 文档。

关于本机带宽检测或无带宽检测

`NCManagerNative` 类是支持本机带宽检测的 `NCManager` 的子类（某些 Flash Video Streaming Service 供应商可能支持本机带宽检测）。使用 `NCManagerNative` 时，Flash Media Server 上不需要有特定的文件。如果不需要带宽检测，则不需要 `main.asc` 文件也可以使用 `NCManagerNative` 连接至任一版本的 Flash Media Server。

若要使用 `NCManagerNative` 而非默认的 `NCManager` 类，请在 FLA 文件的第一帧中添加以下代码行：

```
import fl.video*;
VideoPlayer.ncManagerClass = fl.video.NCManagerNative;
```


关于非本机带宽检测

如果 Flash Video Streaming Service 供应商不提供本机带宽检测，但您却需要进行带宽检测，则必须将 `main.asc` 文件添加到 Flash Media Server FLV 应用程序中。您可以在 Adobe Flash CS3/Samples 和 `Tutorials/Samples/Components/FLVPlayback/main.asc` 下的 Flash 应用程序文件夹中找到该 `main.asc` 文件。

若要设置 Flash Media Server 以流式加载 FLV 文件，请执行以下步骤：

1. 在您的 Flash Media Server 应用程序文件夹中创建一个文件夹，将其命名为 **my_application** 之类的名称。
2. 将 `main.asc` 文件复制到 `my_application` 文件夹中。
3. 在 `my_application` 文件夹中创建名为 **streams** 的文件夹。
4. 在 `streams` 文件夹中创建名为 **_definst_** 的文件夹。
5. 将您的 FLV 文件放置于 **_definst_** 文件夹中。

自定义 FLVPlayback 组件

本节说明如何自定义 FLVPlayback 组件。但是，用于自定义其它组件的大多数方法并不适用于 FLVPlayback 组件。若要自定义 FLVPlayback 组件，请仅使用在本节中介绍的技术。您可以通过下列选项来自定义 FLVPlayback 组件：选择预先设计的外观，分别在各个 FLV 回放自定义用户界面组件内设置外观或者创建新外观。您还可以使用 FLVPlayback 属性修改外观的行为。



为了使外观可用于您的 FLVPlayback 组件，您必须将外观 SWF 文件与应用程序 SWF 文件一起上载到 Web 服务器。

选择预先设计的外观

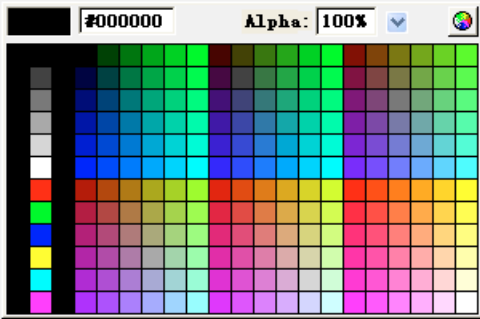
通过单击“组件”检查器中 `skin` 参数的 `value` 单元格，可以选择 `FLVPlayback` 组件的外观。然后，单击放大镜图标以打开下面的“选择外观”对话框，从中可以选择一种外观或者提供用于指定外观 `SWF` 文件位置的 `URL`。



FLVPlayback “选择外观”对话框

在“外观”弹出菜单中列出的外观位于 `Flash Configuration/FLVPlayback Skins/ActionScript 3.0/` 文件夹中。通过创建新外观并将 `SWF` 文件放置在此文件夹中，即可在该对话框中使用新外观。外观名称将显示在弹出菜单中，并且具有 `.swf` 扩展名。有关创建外观集的详细信息，请参阅第 202 页的“创建新外观”。

对于通过设置 `skin` 属性指定的外观，您可以通过在创作期间或在运行时使用 **ActionScript** 设置外观参数，从而独立于外观的选择来指定颜色和 **Alpha**（透明度）值。若要在创作期间指定颜色和 **Alpha** 值，请在“外观选择”对话框中打开“颜色选择器”，如此处所示。



若要选择颜色，请在面板中单击颜色样本或在文本框中输入数值。若要选择 **Alpha** 值，请使用滑块或在 **Alpha** 文本框中键入百分比值。

若要在运行时指定颜色和 **Alpha** 值，请设置 `skinBackgroundColor` 和 `skinBackgroundAlpha` 属性。将 `skinBackgroundColor` 属性设置为 **0xRRGGBB**（红色、绿色、蓝色）值。将 `skinBackgroundAlpha` 属性设置为 **0.0** 和 **1.0** 之间的数值。下面的示例将 `skinBackgroundColor` 设置为 **0xFF0000**（红色）并将 `skinBackgroundAlpha` 设置为 **.5**。

```
my_FLVPlayback.skinBackgroundColor = 0xFF0000;
my_FLVPlayback.skinBackgroundAlpha = .5;
```

默认值为用户上次选择的值。

如果要使用 **FLV** 回放自定义用户界面设置 **FLVPlayback** 组件的外观，请从弹出菜单中选择“无”。

分别在各个 **FLV** 回放自定义用户界面组件内设置外观

使用 **FLV** 回放自定义用户界面组件可在 **FLA** 文件内自定义 **FLVPlayback** 控件的外观，并且可在预览网页时看到结果。但是，根据设计，这些组件无法缩放。您应该编辑影片剪辑及其内容，使其具有特定大小。因此，通常最好使 **FLVPlayback** 组件按所需大小显示在舞台上，并且将 `scaleMode` 设置为 `exactFit`。

若要开始，只需将所需 **FLV** 回放自定义用户界面组件从“组件”面板中拖出，将它们放置于舞台上的适当位置，并为它们指定实例名称。

这些组件可以不通过任何 **ActionScript** 而发挥作用。如果将这些组件放在与 **FLVPlayback** 组件相同的时间轴和帧上并且尚未设置 **FLVPlayback** 组件的外观，则 **FLVPlayback** 组件将自动与这些组件相连接。如果舞台上有多于一个 **FLVPlayback** 组件，或者自定义控件和 **FLVPlayback** 不在同一时间轴上，则需要 **ActionScript**。

当组件位于舞台上后，即可像对待任何其它元件一样编辑它们。在打开这些组件后，可以看到各个组件之间在设置上都存在微小差异。

按钮组件

按钮组件具有类似的结构。按钮包括 **BackButton**、**ForwardButton**、**MuteButton**、**PauseButton**、**PlayButton**、**PlayPauseButton** 和 **StopButton**。大多数按钮都在第一帧上具有实例名称为 **placeholder_mc** 的单个影片剪辑。这通常是正常状态按钮的实例，但不一定必须是这样。在第 2 帧的舞台上有四个影片剪辑，分别针对以下显示状态：正常、指针经过、按下和禁用。（在运行时，组件实际上永远不会转到第 2 帧；这些影片剪辑放置于此是为了使编辑更方便，并且强制它们加载到 **SWF** 文件中，而不必在“元件属性”对话框内选中“在第一帧导出”复选框。不过，您仍然必须选择“为 **ActionScript** 导出”选项。）

为了设置按钮的外观，只需编辑上述各影片剪辑即可。您可以更改其大小及外观。

某些 **ActionScript** 通常在第一帧中出现。您应该不需要更改此脚本。它只停止第一帧上的播放头，并且指定哪些影片剪辑将用于哪些状态。

PlayPauseButton、**MuteButton**、**FullScreenButton** 和 **CaptionButton** 按钮
PlayPauseButton、**MuteButton**、**FullScreenButton** 和 **CaptionButton** 按钮在设置上与其它按钮不同；它们只具有带两个图层且没有脚本的一个帧。在该帧上，有两个按钮，彼此叠放 — 对于 **PlayPauseButton**，这两个按钮是 **Play** 和 **Pause** 按钮；对于 **MuteButton**，这两个按钮是 **Mute-on** 和 **Mute-off** 按钮；对于 **FullScreenButton**，这两个按钮是 **full-screen-on** 和 **full-screen-off** 按钮；对于 **CaptionButton**，这两个按钮是 **caption-on** 和 **caption-off** 按钮。若要设置这些按钮的外观，请按照第 195 页的“[分别在各个 FLV 回放自定义用户界面组件内设置外观](#)”中所述分别设置这两个内部按钮的外观；不需要进行额外的操作。

CaptionButton 是针对 **FLVPlaybackCaptioning** 组件的，必须将其附加到该组件，而非附加到 **FLVPlayback** 组件。

BackButton 和 ForwardButton 按钮

BackButton 和 **ForwardButton** 按钮在设置上也与其它按钮不同。在第 2 帧上，它们具有额外的影片剪辑，您可以将它们作为帧使用，并环绕这两个按钮或其中一个按钮。这些影片剪辑不是必需的，并且没有特殊功能；只是出于方便目的才提供它们。若要使用它们，只需将它们从“库”面板拖到舞台上，然后将它们放置于所需位置。如果您不想使用它们，则可以不使用它们，也可以从“库”面板中删除它们。

提供的大多数按钮都基于影片剪辑的公共集，以便您可以一次更改所有按钮的外观。您可以使用此功能，也可以替换这些公共剪辑以使每个按钮在外观上都不同。

BufferingBar 组件

缓冲栏组件十分简单：它由一个动画组成，该动画在组件进入缓冲状态时出现，并且它不需要任何特定的 **ActionScript** 来配置它。默认情况下，它是一个从左向右移动的有斑纹的条，在该条上有一个矩形遮罩，使其呈现“理发店招牌”效果，但对于此配置没有什么特殊设置。

尽管外观 **SWF** 文件中的缓冲栏使用 9 切片缩放（因为它们需要在运行时缩放），但 **BufferingBar** **FLV** 自定义用户界面组件没有并且“无法”使用 9 切片缩放，因为它嵌套了影片剪辑。如果要使 **BufferingBar** 更宽或更高，则可能需要更改其内容，而不是对其进行缩放。

SeekBar 和 VolumeBar 组件

SeekBar 和 **VolumeBar** 组件十分相似，尽管它们在功能上不同。它们都具有手柄，都使用相同的手柄跟踪机制，并且都支持在其中嵌套剪辑以跟踪进度和完成程度。

在许多地方，**FLVPlayback** 组件中的 **ActionScript** 代码都假定 **SeekBar** 或 **VolumeBar** 组件的注册点（也称为“原点”或“零点”）位于内容的左上角，因此，维持这一惯例十分重要。否则，手柄以及进度和完成程度影片剪辑可能会有问题。

尽管外观 **SWF** 文件中的搜索栏使用 9 切片缩放（因为它们需要在运行时缩放），但 **SeekBar** **FLV** 自定义用户界面组件没有并且“无法”使用 9 切片缩放，因为它嵌套了影片剪辑。如果要使 **SeekBar** 更宽或更高，则可能需要更改其内容，而不是缩放其大小。

手柄

手柄影片剪辑的一个实例位于第 2 帧。与 **BackButton** 和 **ForwardButton** 组件一样，该组件实际上永远不会转到第 2 帧；这些影片剪辑放置于此是为了使编辑更方便，并且以这种方式强制它们加载到 **SWF** 文件中，而不必在“元件属性”对话框中选中“在第一帧导出”复选框。不过，您仍然必须选择“为 **ActionScript** 导出”选项。

您可能注意到，手柄影片剪辑在背景中具有一个 **Alpha** 设置为 **0** 的矩形。此矩形增加了该手柄的点击区域的大小，这样，不必更改手柄外观就可以更容易地控制手柄，这与按钮的点击状态类似。因为该手柄是在运行时动态创建的，所以它必须是一个影片剪辑，而不是一个按钮。对于任何其它情况，**Alpha** 设置为 **0** 的这一矩形都不是必需的；通常，您可以用任何所需图像替代手柄内的内容。不过，如果将注册点保持在手柄影片剪辑的中部水平居中，则它的使用效果最好。

以下 **ActionScript** 代码位于 **SeekBar** 组件的第一帧上，用于管理手柄：

```
stop();
handleLinkageID = "SeekBarHandle";
handleLeftMargin = 2;
handleRightMargin = 2;
handleY = 11;
```

由于第 2 帧的内容，对 `stop()` 函数的调用是必需的。

第二行指定哪个元件要用作手柄；并且，如果您只编辑第 2 帧上的手柄影片剪辑实例，则应该无需对此进行更改。在运行时，**FLVPlayback** 组件将舞台上指定的影片剪辑的实例作为 **Bar** 组件实例的同级创建，也就是说它们具有相同的父影片剪辑。因此，如果您的栏位于根级别，则手柄也必须位于根级别。

变量 `handleLeftMargin` 确定手柄的原始位置 (**0%**)，而变量 `handleRightMargin` 确定在结束时它所处的位置 (**100%**)。这两个数值提供与栏控件左端和右端的偏移量，正数值标记栏内部的限制，负数值标记栏外部的限制。这两个偏移量根据手柄注册点指定手柄可以转到的位置。如果您将注册点放置于手柄的中间，则该手柄的左侧和右侧将越过边界。为了正常使用，搜索栏影片剪辑必须将其注册点放置于其内容的左上角。

变量 `handleY` 确定该手柄相对于栏实例的 *y* 位置。这基于每个影片剪辑的注册点。范例手柄的注册点位于三角形的尖端，以便相对于可见部分放置它，而不考虑不可见的点击状态矩形。此外，为了正常使用，栏影片剪辑必须将其注册点保持在其内容的左上角。

因此，以这些限制为例，如果在 (**100, 100**) 位置设置了某个栏控件，并且其宽度为 **100** 个像素，则手柄在水平方向上的范围可从 **102** 到 **198**，并且在垂直方向上保持 **111**。如果您将 `handleLeftMargin` 和 `handleRightMargin` 更改为 **-2**，并将 `handleY` 更改为 **-11**，则手柄在水平方向上的范围可从 **98** 到 **202**，并且在垂直方向上保持 **89**。

进度和完成程度影片剪辑

SeekBar 组件具有 *进度* 影片剪辑，**VolumeBar** 具有 *完成程度* 影片剪辑。但在实践中，任何 **SeekBar** 或 **VolumeBar** 都可以不具有或者同时具有这两个影片剪辑，或者具有其中一个影片剪辑。它们在结构上相同，并在行为上类似，但跟踪不同的值。进度影片剪辑随着 **FLV** 文件下载而填充（这仅适用于 **HTTP** 下载，因为如果从 **FMS** 进行流式加载，它始终是满的），而完成程度影片剪辑随着手柄从左向右移动而填充。

FLVPlayback 组件通过查找特定的实例名称找到这些影片剪辑实例，因此，您的进度影片剪辑实例必须使栏影片剪辑作为其父级，并且具有实例名称 `progress_mc`。完成程度影片剪辑实例必须具有实例名称 `fullness_mc`。

您可以设置进度和完成程度影片剪辑，而不管其内部是否嵌套 `fill_mc` 影片剪辑实例。

VolumeBar `fullness_mc` 影片剪辑使用 `fill_mc` 影片剪辑显示该方法， SeekBar `progress_mc` 影片剪辑则不使用 `fill_mc` 影片剪辑而显示该方法。

如果要进行的填充不通过扭曲外观无法缩放，则在内部嵌套 `fill_mc` 影片剪辑的方法十分有用。

在 VolumeBar `fullness_mc` 影片剪辑中，嵌套的 `fill_mc` 影片剪辑实例被遮罩。您可以在创建影片剪辑时遮罩它，也可以在运行时动态地创建遮罩。如果您用影片剪辑遮罩它，则将该实例命名为 `mask_mc` 并对它进行设置，以便 `fill_mc` 显示为就像百分比为 100% 时一样。如果您不遮罩 `fill_mc`，则动态创建的遮罩将是矩形，并且在 100% 时与 `fill_mc` 的大小相同。

`fill_mc` 影片剪辑通过两种方法之一与遮罩一起显示，具体方法取决于 `fill_mc.slideReveal` 是 `true` 还是 `false`。

如果 `fill_mc.slideReveal` 是 `true`，则 `fill_mc` 从左向右移动，以通过遮罩显示它。为 0% 时，它自始至终向左，因此不会有任何部分通过遮罩显示。随着百分比的增加，它向右移动，直到达到 100%，此时它位于舞台上的创建位置之后。

如果 `fill_mc.slideReveal` 是 `false` 或未定义（默认行为），则遮罩将从左向右调整大小，以显示 `fill_mc` 的更多部分。在它达到 0% 时，遮罩就在水平方向上缩放到 05，并且随着百分比的增加，`scaleX` 也将增加，直到达到 100%，这时它将显示所有 `fill_mc`。`scaleX = 100` 并不是必需的，因为在创建 `mask_mc` 时可能已对它进行了缩放。

没有 `fill_mc` 的方法比具有 `fill_mc` 的方法更简单，但它水平扭曲填充。如果不需要进行该扭曲，则必须使用 `fill_mc`。SeekBar `progress_mc` 阐释了这一方法。

进度或完成程度影片剪辑根据百分比进行水平缩放。为 0% 时，实例的 `scaleX` 被设置为 0，这使其不可见。随着该百分比增加，将对 `scaleX` 进行调整，直到达到 100%，此时该影片剪辑的大小将与创建它时在舞台上的大小相同。同样，`scaleX = 100` 并不是必需的，因为在创建该影片剪辑实例时可能已对它进行了缩放。

连接 FLV 回放自定义用户界面组件

如果将自定义用户界面组件放在与 FLVPlayback 组件相同的时间轴和帧上并且尚未设置 skin 属性，则 FLVPlayback 将不需要任何 ActionScript 而自动与这些组件相连接。

如果舞台上有多于一个 FLVPlayback 组件，或者自定义控件和 FLVPlayback 不在同一时间轴上，则必须通过编写 ActionScript 代码将自定义用户界面组件连接至 FLVPlayback 组件实例。首先，必须向 FLVPlayback 实例分配一个名称，然后使用 ActionScript 将您的 FLV 回放自定义用户界面组件实例分配给相应的 FLVPlayback 属性。在以下示例中，FLVPlayback 实例是 my_FLVPlybk，FLVPlayback 属性名称后跟有句点 (.)，并且 FLV 回放自定义用户界面控件实例位于等号 (=) 的右侧：

```
//FLVPlayback instance = my_FLVPlybk
my_FLVPlybk.playButton = playbtn; // set playButton prop. to playbtn, etc.
my_FLVPlybk.pauseButton = pausebtn;
my_FLVPlybk.playPauseButton = playpausebtn;
my_FLVPlybk.stopButton = stopbtn;
my_FLVPlybk.muteButton = mutebtn;
my_FLVPlybk.backButton = backbtn;
my_FLVPlybk.forwardButton = forbtn;
my_FLVPlybk.volumeBar = volbar;
my_FLVPlybk.seekBar = seekbar;
my_FLVPlybk.bufferingBar = bufbar;
```

示例

以下几个步骤创建自定义的 StopButton、PlayPauseButton、MuteButton 和 SeekBar 控件：

若要创建自定义的 StopButton、PlayPauseButton、MuteButton 和 SeekBar 控件，请执行以下步骤：

1. 将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my_FLVPlybk**。
2. 通过“组件”检查器将 source 参数设置为 **<http://www.helpexamples.com/flash/video/cuepoints.flv>**。
3. 将 Skin 参数设置为“无”。
4. 将 StopButton、PlayPauseButton 和 MuteButton 拖到舞台上，然后将它们放置于 FLVPlayback 实例上，并垂直堆叠在左侧。在“属性”检查器中为每个按钮提供一个实例名称（如 **my_stopbtn**、**my_plypausbtn** 和 **my_mutebtn**）。
5. 在“库”面板中，打开 FLVPlayback Skins 文件夹，然后打开该文件夹下面的 SquareButton 文件夹。
6. 选择 SquareBgDown 影片剪辑，并且双击它以在舞台上打开它。
7. 右键单击 (Windows) 或者按住 Control 键单击 (Macintosh)，从菜单中选择“全选”，然后删除该元件。

8. 选择椭圆工具，在相同位置上绘制一个椭圆，然后将填充设置为蓝色 (#0033FF)。
9. 在“属性”检查器中，将宽度 (W:) 设置为 40 并将高度 (H:) 设置为 20。将 x 坐标 (X:) 设置为 0.0 并将 y 坐标 (Y:) 设置为 0.0。
10. 对 SquareBgNormal 重复第 6 步到第 8 步，但将填充更改为黄色 (#FFFF00)。
11. 对 SquareBgOver 重复第 6 步到第 8 步，但将填充更改为绿色 (#006600)。
12. 编辑按钮内不同元件图标 (PauseIcon、PlayIcon、MuteOnIcon、MuteOffIcon 和 StopIcon) 的影片剪辑。您可以在 FLV Playback Skins/Label Button/Assets 下的“库”面板中找到这些影片剪辑，其中 *Label* 是按钮的名称，如“播放”、“暂停”等。为每个按钮执行以下步骤：
 - a. 选择“全选”选项。
 - b. 将颜色更改为红色 (#FF0000)。
 - c. 缩放 300%。
 - d. 将内容的 X: 位置更改为 7.0，以便更改图标在每个按钮状态中的水平位置。



通过用此方法更改该位置，您可以避免打开每个按钮状态和移动图标影片剪辑实例。

13. 单击时间轴上的蓝色“返回”箭头以返回到第 1 场景的第一帧。
14. 将一个 SeekBar 组件拖到该舞台上，并且将其放置在 FLVPlayback 实例的右下角。
15. 在“库”面板中，双击 SeekBar 以在舞台上打开它。
16. 将其缩放到 400%。
17. 选择轮廓，并且将颜色设置为红色 (#FF0000)。
18. 双击 FLVPlayback Skins/Seek Bar 文件夹中的 SeekBarProgress，并且将颜色设置为黄色 (#FFFF00)。
19. 双击 FLVPlayback Skins/Seek Bar 文件夹中的 SeekBarHandle，并且将颜色设置为红色 (#FF0000)。
20. 单击时间轴上的蓝色“返回”箭头以返回到第 1 场景的第一帧。
21. 选择舞台上的 SeekBar 实例，并为其指定实例名称 **my_seekbar**。
22. 在时间轴第一帧的“动作”面板上，为 video 类添加一条 import 语句，并且将按钮和搜索栏名称分配给相应的 FLVPlayback 属性，如下示例所示：


```
import fl.video.*;
my_FLVPlayback.stopButton = my_stopbtn;
my_FLVPlayback.playPauseButton = my_playpausebtn;
my_FLVPlayback.muteButton = my_mutebtn;
my_FLVPlayback.seekBar = my_seekbar;
```
23. 同时按下 Ctrl+Enter 以测试该影片。

创建新外观

创建外观 SWF 文件的最佳方式是复制与 Flash 一起提供的外观文件之一，并且将它用作开始点。您可以在 Flash 应用程序文件夹 **Configuration/FLVPlayback Skins/FLA/ActionScript 3.0/** 中找到这些外观的 FLA 文件。若要使已完成的外观 SWF 文件可用作“选择外观”对话框中的选项，请将它放置于 **Configuration/FLVPlayback Skins/ActionScript 3.0** 文件夹中，或者在 Flash 应用程序文件夹中或者在用户本地的 **Configuration/FLVPlayback Skins/ActionScript 3.0** 文件夹中。

由于可以独立于外观的选择来设置外观的颜色，因而无需编辑 FLA 文件来修改颜色。如果创建了具有特定颜色的外观并且不希望在“选择外观”对话框中能够对其进行编辑，请在外观 FLA 文件 **ActionScript** 代码中设置 `this.border_mc.colorMe = false;`。有关设置外观颜色的信息，请参阅第 194 页的“选择预先设计的外观”。

在查看已安装的 Flash 外观 FLA 文件时，舞台上的某些项似乎不是必需的，但它们中的很多项被放置于引导图层中。利用实时预览（使用缩放 9），可以快速查看 SWF 文件中实际显示的内容。

以下几节介绍一些对 **SeekBar**、**BufferingBar** 和 **VolumeBar** 影片剪辑的更复杂的自定义和更改操作。

使用外观布局

打开 Flash 外观 FLA 文件时，您会发现外观的影片剪辑都放置在主时间轴上。您在同一帧上发现的这些剪辑和 **ActionScript** 代码用于定义在运行时放置控件的方式。

尽管布局图层看起来很像外观在运行时的样子，但此图层的内容在运行时不可见。它只用于计算控件的放置位置。舞台上的其它控件在运行时使用。

布局图层内是用于名为 **video_mc** 的 **FLVPlayback** 组件的占位符。所有其它控件都相对于 **video_mc** 放置。如果您从某个 Flash FLA 文件开始并且更改控件的大小，则很可能通过移动这些占位符剪辑修复布局。

每个占位符剪辑都具有特定的实例名称。占位符剪辑的名称分别为 **playpause_mc**、**play_mc**、**pause_mc**、**stop_mc**、**captionToggle_mc**、**fullScreenToggle_mc**、**back_mc**、**bufferingBar_mc**、**bufferingBarFill_mc**、**seekBar_mc**、**seekBarHandle_mc**、**seekBarProgress_mc**、**volumeMute_mc**、**volumeBar_mc** 和 **volumeBarHandle_mc**。选择外观颜色时更改了颜色的剪辑的名称为 **border_mc**。

哪个剪辑用于控件并不重要。通常，对按钮使用正常状态剪辑。对于其它控件，使用该控件的剪辑，但这只是出于方便目的。所有重要之处在于 *x*（水平）和 *y*（垂直）位置以及占位符的高度和宽度。

您还可以在标准控件旁具有所需任何数目的附加剪辑。对于这些剪辑的唯一要求是：在“链接”对话框中为它们的库元件选中“为 **ActionScript** 导出”。布局图层中的自定义剪辑可以具有任何实例名称，但不可具有上面所列的保留实例名称。仅在设置剪辑的 **ActionScript** 以确定布局时才需要使用实例名称。

border_mc 剪辑是特殊的。如果将 `FLVPlayback.skinAutoHide` 属性设置为 `true`，则在鼠标经过 **border_mc** 剪辑时将显示外观。这对于显示在视频播放器的边界外的外观十分重要。有关 `skinAutoHide` 属性的信息，请参阅第 206 页的“修改外观行为”。

在 **Flash FLA** 文件中，**border_mc** 用于烙印，并用于环绕“快进”和“快退”按钮的边界。

border_mc 剪辑也是外观的一部分，外观的 **Alpha** 值和颜色可以通过 `skinBackgroundAlpha` 和 `skinBackgroundColor` 属性来更改。若要使颜色和 **Alpha** 值可自定义，外观 **FLA** 文件中的 **ActionScript** 必须包括：

```
border_mc.colorMe = true;
```

ActionScript

以下 **ActionScript** 代码通常适用于所有控件。某些控件具有定义附加行为的特定 **ActionScript**，在针对该控件的那一节中将对对此加以说明。

初始 **ActionScript** 是很大的一部分，它指定每个组件每种状态的类名称。可以在 **SkinOverAll.fla** 文件中看到所有这些类名称。其代码与 **Pause** 和 **Play** 按钮的代码相似，例如：

```
this.pauseButtonDisabledState = "fl.video.skin.PauseButtonDisabled";
this.pauseButtonDownState = "fl.video.skin.PauseButtonDown";
this.pauseButtonNormalState = "fl.video.skin.PauseButtonNormal";
this.pauseButtonOverState = "fl.video.skin.PauseButtonOver";
this.playButtonDisabledState = "fl.video.skin.PlayButtonDisabled";
this.playButtonDownState = "fl.video.skin.PlayButtonDown";
this.playButtonNormalState = "fl.video.skin.PlayButtonNormal";
this.playButtonOverState = "fl.video.skin.PlayButtonOver";
```

实际上，这些类名称并不具有外部类文件；只是在“链接”对话框中为库中的所有影片剪辑指定了它们。

在 **ActionScript 2.0** 组件中，在运行时确实使用了舞台上的一些影片剪辑。在 **ActionScript 3.0** 组件中，这些影片剪辑仍在 **FLA** 文件中，但只是为了便于编辑。现在，它们全部在引导图层中并且不可导出。例如，库中的所有外观资源均设置为在第一帧导出，并且它们是使用如下代码动态创建的。

```
new fl.video.skin.PauseButtonDisabled()
```

该部分之后的 **ActionScript** 代码用于定义外观的最小宽度和高度。“选择外观”对话框显示这些值，并且这些值在运行时使用，以防止外观缩放到小于其最小大小。如果您不想指定最小大小，则将其保留为未定义或者小于或等于零。

```
// minimum width and height of video recommended to use this skin,  
// leave as undefined or <= 0 if there is no minimum  
this.minWidth = 270;  
this.minHeight = 60;
```

每个占位符都可以具有适用于它的以下属性:

属性	说明
anchorLeft	布尔值。相对于 FLVPlayback 实例左侧放置控件。默认值为 true, 除非 anchorRight 被显式设置为 true, 此时其默认值为 false。
anchorRight	布尔值。相对于 FLVPlayback 实例的右侧放置控件。默认值为 false。
anchorBottom	布尔值。相对于 FLVPlayback 实例底部放置控件。默认值为 true (除非 anchorTop 被显式设置为 true), 然后它默认为 false。
anchorTop	布尔值。相对于 FLVPlayback 实例顶部放置控件。默认值为 false。

如果 anchorLeft 和 anchorRight 属性均为 true, 则在运行时水平缩放该控件。如果 anchorTop 和 anchorBottom 属性均为 true, 则在运行时垂直缩放该控件。

若要看到这些属性的效果, 请查看它们在 Flash 外观中的使用方式。BufferingBar 和 SeekBar 控件是可以缩放的唯一控件, 它们彼此层叠放置, 并且都将 anchorLeft 和 anchorRight 属性设置为 true。BufferingBar 和 SeekBar 左侧的所有控件都将 anchorLeft 设置为 true, 而其右侧的所有控件都将 anchorRight 设置为 true。所有控件都将 anchorBottom 设置为 true。

您可以尝试编辑布局图层上的影片剪辑, 以生成控件位于其顶部 (而不是底部) 的外观。您只需要将控件移到顶部 (相对于 video_mc), 并且对于所有控件将 anchorTop 设置为等于 true。

缓冲栏

缓冲栏具有两个影片剪辑: bufferingBar_mc 和 bufferingBarFill_mc。舞台上各剪辑与其它剪辑的相对位置很重要, 因为这种相对位置将一直保留。缓冲栏使用两个单独的剪辑, 因为组件缩放 bufferingBar_mc, 但不缩放 bufferingBarFill_mc。

bufferingBar_mc 剪辑对自身应用 9 切片缩放, 因此在缩放时边框将不会扭曲。bufferingBarFill_mc 剪辑非常宽, 因此它始终足够宽而不需要放大。在运行时将自动对它进行遮罩, 以便只显示高于伸展的 bufferingBar_mc 的部分。默认情况下, 遮罩的准确尺寸将根据 bufferingBar_mc 和 bufferingBarFill_mc 的 x (水平) 位置之间的差异在 bufferingBar_mc 内保持相同的左边距和右边距。您可以使用 ActionScript 代码自定义该放置。

如果缓冲栏不需要缩放或者不使用 9 切片缩放，则可以像设置 FLV 回放自定义用户界面 **BufferingBar** 组件一样设置它。有关详细信息，请参阅第 197 页的“**BufferingBar 组件**”。

缓冲栏具有以下附加属性：

属性	说明
<code>fill_mc:MovieClip</code>	指定缓冲栏填充的实例名称。默认值为 <code>bufferingBarFill_mc</code> 。

搜索栏和音量栏

搜索栏也具有两个影片剪辑：`seekBar_mc` 和 `seekBarProgress_mc`。布局图层上各剪辑与其它剪辑的相对位置很重要，因为这种相对位置将一直保留。尽管这两个剪辑都缩放，但 `seekBarProgress_mc` 无法嵌套在 `seekBar_mc` 内，因为 `seekBar_mc` 使用 9 切片缩放，而 9 切片缩放无法很好地使用嵌套的影片剪辑。

`seekBar_mc` 剪辑对自身应用 9 切片缩放，因此在缩放时边框将不会扭曲。

`seekBarProgress_mc` 剪辑也缩放，但它却扭曲。它并不使用 9 切片缩放，因为它是一个填充，而填充在扭曲时显示也很正常。

`seekBarProgress_mc` 剪辑在没有 `fill_mc` 的情况下工作，与 `progress_mc` 剪辑在 FLV 回放自定义用户界面组件中的工作方式十分相似。换言之，它不被遮罩并且水平缩放。在 100% 时，`seekBarProgress_mc` 的确切尺寸由 `seekBarProgress_mc` 剪辑内的左边距和右边距定义。默认情况下，这些尺寸是相等的并且基于 `seekBar_mc` 和 `seekBarProgress_mc` 的 *x*（水平）位置之间的差异。您可以在搜索栏影片剪辑中使用 **ActionScript** 自定义尺寸，如下示例所示：

```
this.seekBar_mc.progressLeftMargin = 2;
this.seekBar_mc.progressRightMargin = 2;
this.seekBar_mc.progressY = 11;
this.seekBar_mc.fullnessLeftMargin = 2;
this.seekBar_mc.fullnessRightMargin = 2;
this.seekBar_mc.fullnessY = 11;
```

可以将此代码放在 **SeekBar** 影片剪辑时间轴上，也可以将其与其它 **ActionScript** 代码一起放在主时间轴上。如果通过使用代码而非修改布局来自定义，则舞台上不需要有填充。它只需要在库中，并使用正确的类名设置为在第 1 帧上为 **ActionScript** 导出。

与 FLV 回放自定义用户界面 **SeekBar** 组件一样，可以为搜索栏创建完成程度影片剪辑。如果您的搜索栏不需要缩放，或者它缩放但不具有 9 切片缩放，则您可以使用用于 FLV 回放自定义用户界面组件的任何方法来设置 `progress_mc` 或 `fullness_mc`。有关详细信息，请参阅第 198 页的“**进度和完成程度影片剪辑**”。

因为 Flash 外观的音量栏不缩放，所以采用与 **VolumeBar** FLV 回放自定义用户界面组件相同的方法建立其结构。有关详细信息，请参阅第 197 页的“**SeekBar 和 VolumeBar 组件**”。例外情况就是手柄的实现方式不同。有关这一点的详细信息，请参阅以下部分。

手柄

SeekBar 和 **VolumeBar** 手柄都放置于布局图层上该栏的旁边。默认情况下，手柄的左边距、右边距和 y 轴值按其相对于栏影片剪辑的位置设置。左边距按手柄的 x （水平）位置和栏的 x （水平）位置之间的差值设置，而右边距等于左边距。您可以在 **SeekBar** 或 **VolumeBar** 影片剪辑中通过 **ActionScript** 自定义这些值。以下示例所采用的 **ActionScript** 代码就是用于 **FLV** 回放自定义用户界面组件的代码：

```
this.seekBar_mc.handleLeftMargin = 2;
this.seekBar_mc.handleRightMargin = 2;
this.seekBar_mc.handleY = 11;
```

可以将此代码放在 **SeekBar** 影片剪辑时间轴上，也可以将其与其它 **ActionScript** 代码一起放在主时间轴上。如果通过使用代码而非修改布局来自定义，则舞台上不需要有手柄。它只需要在库中，并使用正确的类名设置为在第 1 帧上为 **ActionScript** 导出。

除了这些属性之外，手柄就是简单的影片剪辑，其设置方式与在 **FLV** 回放自定义用户界面组件中的设置方式相同。两者都具有 `alpha` 属性设置为 0 的矩形背景。提供它们只是为了增加点击区域，它们并不是必需的。

背景剪辑和前景剪辑

影片剪辑 **chrome_mc** 和 **forwardBackBorder_mc** 是作为背景剪辑实现的。

在舞台和占位符“快进”和“快退”按钮上的 **ForwardBackBorder**、**ForwardBorder** 和 **BackBorder** 影片剪辑中，唯一不在引导图层上的影片剪辑是 **ForwardBackBorder**。它只位于实际使用“快进”和“快退”按钮的外观中。

对于这些剪辑的唯一要求是需要库的第 1 帧上将它们为 **ActionScript** 导出。

修改外观行为

使用 `bufferingBarHidesAndDisablesOthers` 属性和 `skinAutoHide` 属性可以自定义 **FLVPlayback** 外观的行为。

通过将 `bufferingBarHidesAndDisablesOthers` 属性设置为 `true`，可使 **FLVPlayback** 组件隐藏 **SeekBar** 及其手柄，并且在组件进入缓冲状态时禁用“播放”和“暂停”按钮。在 **FLV** 文件通过慢速连接（`bufferTime` 属性设置得较高，例如设置为 10）从 **FMS** 流式加载时，上述功能特别有用。在此情况下，不耐心的用户可能会尝试通过单击“播放”和“暂停”按钮启动搜索，这可能会导致文件播放时间延迟得更长。您可以通过将 `bufferingBarHidesAndDisablesOthers` 设置为 `true`，并在组件处于缓冲状态时禁用 **SeekBar** 组件以及“暂停”和“播放”按钮来防止上述情况发生。

`skinAutoHide` 属性只影响预先设计的外观 **SWF** 文件，并不影响通过 **FLV** 回放自定义用户界面组件创建的控件。如果设置为 `true`，**FLVPlayback** 组件将在鼠标不在查看区域上时隐藏外观。此属性的默认值是 `false`。

使用 SMIL 文件

为了为多个带宽处理多个流，`VideoPlayer` 类使用支持 SMIL 的一个子集的辅助类 (`NCManager`)。SMIL 用于标识视频流的位置、FLV 文件的布局（宽和高）以及对应于不同带宽的源 FLV 文件。它还可以用于指定 FLV 文件的比特率和持续时间。

使用 `source` 参数或 `FLVPlayback.source` 属性 (ActionScript) 指定 SMIL 文件的位置。有关详细信息，请参阅第 180 页的“源”和《ActionScript 3.0 语言和组件参考》中的 `FLVPlayback.source` 属性。

以下示例显示一个 SMIL 文件，该文件使用 RTMP 从 FMS 流式加载多个带宽 FLV 文件：

```
<smil>
  <head>
    <meta base="rtmp://myserver/myapp/" />
    <layout>
      <root-layout width="240" height="180" />
    </layout>
  </head>
  <body>
    <switch>
      <ref src="myvideo_cable.flv" dur="3:00.1"/>
      <video src="myvideo_isdn.flv" system-bitrate="128000"
dur="3:00.1"/>
      <video src="myvideo_mdm.flv" system-bitrate="56000"
dur="3:00.1"/>
    </switch>
  </body>
</smil>
```

`<head>` 标签可以包含 `<meta>` 和 `<layout>` 标签。`<meta>` 标签仅支持 `base` 属性，该属性用于指定视频流（来自 FMS 的 RTMP）的 URL。

`<layout>` 标签仅支持 `root-layout` 元素，该元素用于设置 `height` 和 `width` 属性，因此可确定用来呈现 FLV 文件的窗口的大小。这两个属性仅接受像素值，而不接受百分比。

在 SMIL 文件的正文内，您可以添加指向 FLV 源文件的单个链接；或者，如果您正在从 FMS 流式加载多个带宽的多个文件（如前面的示例中所示），则可以使用 `<switch>` 标签列出这些源文件。

`<switch>` 标签内的 `video` 和 `ref` 标签意义是相同的；也就是说，它们都可以使用 `src` 属性指定 FLV 文件。进一步讲，每种标签都可以使用 `region`、`system-bitrate` 和 `dur` 属性指定 FLV 文件的区域、所需的最小带宽和持续时间。

在 `<body>` 标签内，只允许出现 `<video>`、`<src>` 或 `<switch>` 标签中的一个。

以下示例显示不使用带宽检测的单个 FLV 文件的渐进式下载：

```
<smil>
  <head>
    <layout>
      <root-layout width="240" height="180" />
    </layout>
  </head>
  <body>
    <video src="myvideo.flv" />
  </body>
</smil>
```

<smil>

可用性

Flash Professional 8。

用法

```
<smil>
...
  child tags
...
</smil>
```

属性

无。

子标签

<head>, <body>

父标签

无。

说明

顶级标签，用于标识 SMIL 文件。

示例

以下示例显示指定了三个 FLV 文件的一个 SMIL 文件：

```
<smil>
  <head>
    <meta base="rtmp://myserver/myapp/" />
    <layout>
      <root-layout width="240" height="180" />
    </layout>
  </head>
```



```

    <body>
      <switch>
        <ref src="myvideo_cable.flv" dur="3:00.1"/>
        <video src="myvideo_isdn.flv" system-bitrate="128000" dur="3:00.1"/
      >
        <video src="myvideo_mdm.flv" system-bitrate="56000" dur="3:00.1"/>
      </switch>
    </body>
  </smil>

```

<head>

可用性

Flash Professional 8。

用法

```

<head>
...
child tags
...
</head>

```

属性

无。

子标签

<meta>, <layout>

父标签

<smil>

说明

支持 <meta> 和 <layout> 标签，用于指定源 FLV 文件的位置和默认布局（高度和宽度）。

示例

以下示例将根布局设置为 240 x 180 像素：

```

<head>
  <meta base="rtmp://myserver/myapp/" />
  <layout>
    <root-layout width="240" height="180" />
  </layout>
</head>

```

<meta>

可用性

Flash Professional 8。

用法

```
<meta/>
```

属性

base

子标签

```
<layout>
```

父标签

无。

说明

包含 base 属性，该属性指定源 FLV 文件的位置 (RTMP URL)。

示例

以下示例显示 myserver 基地址的 meta 标签：

```
<meta base="rtmp://myserver/myapp/" />
```

<layout>

可用性

Flash Professional 8。

用法

```
<layout>
...
  child tags
...
</layout>
```

属性

无。

子标签

```
<root-layout>
```

父标签

<meta>

说明

指定 FLV 文件的宽度和高度。

示例

下面的示例指定一个 240 x 180 像素的布局：

```
<layout>
  <root-layout width="240" height="180" />
</layout>
```

<root-layout>

可用性

Flash Professional 8。

用法

```
<root-layout...attributes.../>
```

属性

宽度、高度

子标签

无。

父标签

<layout>

说明

指定 FLV 文件的宽度和高度。

示例

下面的示例指定一个 240 x 180 像素的布局：

```
<root-layout width="240" height="180" />
```

<body>

可用性

Flash Professional 8。

用法

```
<body>
...
child tags
...
</body>
```

属性

无。

子标签

<video>, <ref>, <switch>

父标签

<smil>

说明

包含 <video>、<ref> 和 <switch> 标签，用于分别指定源 FLV 文件的名称、最小带宽以及 FLV 文件的持续时间。仅在使用 <switch> 标签时才支持 system-bitrate 属性。在 <body> 标签内，只允许存在 <switch>、<video> 或 <ref> 标签中的一个实例。

示例

下面的示例指定三个 FLV 文件，其中两个是使用 video 标签指定的，另一个则是使用 ref 标签指定的：

```
<body>
  <switch>
    <ref src="myvideo_cable.flv" dur="3:00.1"/>
    <video src="myvideo_isdn.flv" system-bitrate="128000" dur="3:00.1"/>
    <video src="myvideo_mdm.flv" system-bitrate="56000" dur="3:00.1"/>
  </switch>
</body>
```

<video>

可用性

Flash Professional 8。

用法

```
<video...attributes.../>
```

属性

src, system-bitrate, dur

子标签

无。

父标签

<body>

说明

与 <ref> 标签的用途相当。支持 src 和 dur 属性，这两个属性用于指定源 FLV 文件的名称及其持续时间。dur 属性支持完整时间格式 (00:03:00:01) 和不完整时间格式 (03:00:01)。

示例

以下示例将设置视频的来源和持续时间：

```
<video src="myvideo_mdm.flv" dur="3:00.1"/>
```

<ref>

可用性

Flash Professional 8。

用法

```
<ref...attributes.../>
```

属性

src, system-bitrate, dur

子标签

无。

父标签

<body>

说明

与 <video> 标签的用途相当。支持 src 和 dur 属性，这两个属性用于指定源 FLV 文件的名称及其持续时间。dur 属性支持完整时间格式 (00:03:00:01) 和不完整时间格式 (03:00:01)。

示例

以下示例将设置视频的来源和持续时间：

```
<ref src="myvideo_cable.flv" dur="3:00.1"/>
```

<switch>

可用性

Flash Professional 8。

用法

```
<switch>
...
  child tags
...
</switch/>
```

属性

无。

子标签

<video>, <ref>

父标签

<body>

说明

与 <video> 或 <ref> 子标签一起使用，用于为多个带宽视频流列出 FLV 文件。<switch> 标签支持 system-bitrate 属性，该属性用于指定最小带宽以及 src 和 dur 属性。

示例

下面的示例指定三个 FLV 文件，其中两个是使用 video 标签指定的，另一个则是使用 ref 标签指定的：

```
<switch>
  <ref src="myvideo_cable.flv" dur="3:00.1"/>
  <video src="myvideo_isdn.flv" system-bitrate="128000" dur="3:00.1"/>
  <video src="myvideo_mdm.flv" system-bitrate="56000" dur="3:00.1" />
</switch>
```

使用 FLVPlayback 字幕组件

FLVPlayback 组件允许您在 Adobe Flash CS3 Professional 应用程序中包括一个视频播放器，以便播放下载的 Adobe Flash 视频 (FLV) 文件和 FLV 流文件。有关 FLVPlayback 的详细信息，请参阅第 175 页的第 5 章“使用 FLVPlayback 组件”。

FLVPlaybackCaptioning 组件允许您为视频提供紧密的字幕支持。字幕组件支持 W3C 标准 XML 格式的 Timed Text 并包括以下功能：

- “利用嵌入的事件提示点显示字幕” — 将 FLV 文件中嵌入的事件提示点与 XML 相关联，以提供字幕显示的功能，而不是使用 Timed Text XML 文件。
- “多个 FLVPlayback 字幕” — 创建用于多个 FLVPlayback 实例的多个 FLVPlayback 字幕实例。
- “切换按钮控件” — 通过字幕切换按钮来实现用户与字幕之间的交互。

使用 FLVPlaybackCaptioning 组件

您可以将 FLVPlaybackCaptioning 组件与一个或多个 FLVPlayback 组件一起使用。在最简单的情况下，您可将一个 FLVPlayback 组件拖到舞台中，将一个 FLVPlaybackCaptioning 组件拖到同一个舞台中，标识您的字幕 URL 并将字幕设置为显示。另外，您也可以设置各种参数来自定义您的 FLVPlayback 字幕。

将字幕添加到 FLVPlayback 组件

您可以将 FLVPlaybackCaptioning 组件添加到任何 FLVPlayback 组件中。有关向您的应用程序添加 FLVPlayback 组件的信息，请参阅第 177 页的“创建具有 FLVPlayback 组件的应用程序”。

从“组件”面板中添加 FLVPlaybackCaptioning 组件：

1. 在“组件”面板中，打开“视频”文件夹。
2. 拖动（或双击）FLVPlaybackCaptioning 组件并将其添加到与将要为其添加字幕的 FLVPlayback 组件所处的舞台上。

提示

Adobe 提供了两个范例来帮助您快速了解 FLVPlaybackCaptioning 组件：caption_video.flv（FLVPlayback 范例）和 caption_video.xml（字幕范例）。这两个文件可从以下位置获得：
<http://www.helpexamples.com/flash/video>。

3. （可选）将 CaptionButton 组件拖动到 FLVPlayback 和 FLVPlaybackCaptioning 组件所在的舞台上。CaptionButton 组件使用户能够打开或关闭字幕。

提示

若要启用 CaptionButton 组件，您必须将其拖动到 FLVPlayback 和 FLVPlaybackCaptioning 组件所在的舞台上。

4. 在舞台中已选中了 FLVPlaybackCaptioning 组件的情况下，在“属性”检查器的“参数”选项卡上指定以下所需信息：

- 将 showCaptions 设置为 true。
- 将 Timed Text XML 文件的 source 指定为 download。

提示

在 Flash 中测试字幕时，应该将 showCaptions 属性设置为 true。但是，如果您包括了 CaptionButton 组件以允许用户打开或关闭字幕，应该将 showCaptions 属性设置为 false。

还可以使用其它参数自定义 FLVPlaybackCaptioning 组件。有关详细信息，请参阅第 223 页的“自定义 FLVPlaybackCaptioning 组件”和《ActionScript 3.0 语言和组件参考》。

5. 选择“控制”>“测试影片”以启动视频。

使用 ActionScript 动态创建实例：

1. 将 FLVPlayback 组件从“组件”面板拖到“库”面板（“窗口”>“库”）中。
2. 将 FLVPlaybackCaptioning 组件从“组件”面板拖到“库”面板中。
3. 将以下代码添加到时间轴第一帧的“动作”面板上。

```
import fl.video.*;
```

提示

以下示例适用于 Adobe Flash CS3 for Windows。FLVPlayback Skins 在 Macintosh 中的位置为 Macintosh HD/Applications/Adobe Flash CS3/ Configuration/FLVPlayback Skins/ActionScript 3.0/SkinUnderPlaySeekCaption.swf。

```
var my_FLVPlaybk = new FLVPlayback();  
my_FLVPlaybk.x = 100;  
my_FLVPlaybk.y = 100;  
addChild(my_FLVPlaybk);
```



```

my_FLVPlayback.skin = "install_drive:/Program Files/Adobe/Adobe Flash CS3/
en/Configuration/FLVPlayback Skins/ActionScript 3.0/
SkinUnderPlaySeekCaption.swf";
my_FLVPlayback.source = "http://www.helpexamples.com/flash/video/
caption_video.flv";
var my_FLVPlaybackcap = new FLVPlaybackCaptioning();
addChild (my_FLVPlaybackcap);
my_FLVPlaybackcap.source = "http://www.helpexamples.com/flash/video/
caption_video.xml";
my_FLVPlaybackcap.showCaptions = true;

```

4. 将 *install_drive* 更改为安装了 Flash 的驱动器，并修改路径以反映您的系统中 Skins 文件夹的位置：



如果您用 ActionScript 创建 FLVPlayback 实例，则还必须用 ActionScript 设置 skin 属性以便为其动态指定一个外观。如果使用 ActionScript 应用外观，则外观不会自动随 SWF 文件一同发布 SWF 文件。请将外观 SWF 文件及应用程序 SWF 文件复制到您的服务器，否则在用户执行程序时该外观 SWF 文件将不可用。

设置 FLVPlaybackCaptioning 组件参数

对于每个 FLVPlaybackCaptioning 组件实例，都可以在“属性”检查器或“组件”检查器中设置以下参数来进一步自定义组件。以下列表标识和简要介绍了这些属性：

- **autoLayout**。确定 FLVPlaybackCaptioning 组件是否控制字幕区域的大小。默认值为 **true**。
- **captionTargetName**。标识包含字幕的 TextField 或 MovieClip 实例的名称。默认值为 **auto**。
- **flvPlaybackName**。标识要显示字幕的 FLVPlayback 实例的名称。默认值为 **auto**。
- **simpleFormatting**。如果设置为 **true**，将对 Timed Text XML 文件的格式设置指令进行限制。默认值为 **false**。
- **showCaptions**。确定是否显示字幕。默认值为 **true**。
- **source**。标识 Timed Text XML 文件的位置。

有关所有 FLVPlaybackCaptioning 参数的详细信息，请参阅 [《ActionScript 3.0 语言和组件参考》](#)。

指定 source 参数

使用 source 参数指定包含影片字幕的 Timed Text XML 文件的名称和位置。请在“组件”检查器的 source 单元格中直接输入 URL 路径。

显示字幕

若要查看字幕，请将 `showCaptions` 参数设置为 `true`。

有关所有 `FLVPlaybackCaptioning` 组件参数的详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》。

在先前的示例中，您已经学习了如何创建和启用 `FLVPlaybackCaptioning` 组件来显示字幕。字幕有两个来源：(1) 包含字幕的 `Timed Text XML` 文件或 (2) 包含与嵌入的事件提示点相关联的字幕文本的 `XML` 文件。

使用 Timed Text 字幕

`FLVPlaybackCaptioning` 组件通过下载 `Timed Text (TT) XML` 文件来为关联的 `FLVPlayback` 组件实现字幕显示功能。有关 `Timed Text` 格式的详细信息，请查看 <http://www.w3.org> 上的 `AudioVideo Timed Text` 信息。

本节简要介绍了受支持的 `Timed Text` 标签和必需的字幕文件标签，并且提供了一个 `Timed Text XML` 文件范例。有关所有受支持 `Timed Text` 标签的详细信息，请参阅[附录 A “Timed Text 标签”](#)。

`FLVPlaybackCaptioning` 组件支持以下 `Timed Text` 标签：

- 段落格式支持
 - 右对齐、左对齐或居中对齐某个段落
- 文本格式支持
 - 以绝对像素大小或 `delta` 样式设置文本的大小（例如 `+2`、`-4`）
 - 设置文本颜色和字体
 - 将文本设为粗体和斜体
 - 设置文本对齐
- 其它格式支持
 - 设置字幕的 `TextField` 的背景颜色
 - 将字幕的 `TextField` 的背景颜色设置为透明 (`alpha 0`)
 - 设置字幕的 `TextField` 的自动换行（打开或关闭）

`FLVPlaybackCaptioning` 组件与 `FLV` 文件的时间代码相匹配。每个字幕必须有一个 `begin` 属性，以确定字幕应该在什么时间显示。如果字幕没有 `dur` 或 `end` 属性，则显示下一个字幕或 `FLV` 文件结束时该字幕将消失。

以下是一个 `Timed Text XML` 文件的示例。该文件 (`caption_video.xml`) 为 `caption_video.flv` 文件提供字幕。这两个文件可从以下位置获得：<http://www.helpexamples.com/flash/video/>。

```

<?xml version="1.0" encoding="UTF-8"?>
<tt xml:lang="en" xmlns="http://www.w3.org/2006/04/ttaf1"
  xmlns:tts="http://www.w3.org/2006/04/ttaf1#styling">
  <head>
    <styling>
      <style id="1" tts:textAlign="right"/>
      <style id="2" tts:color="transparent"/>
      <style id="3" style="2" tts:backgroundColor="white"/>
      <style id="4" style="2 3" tts:fontSize="20"/>
    </styling>
  </head>
  <body>
    <div xml:lang="en">
      <p begin="00:00:00.00" dur="00:00:03.07">I had just joined <span
tts:fontFamily="monospaceSansSerif,proportionalSerif,TheOther"tts:fontSiz
e="+2">Macromedia</span> in 1996,</p>
      <p begin="00:00:03.07" dur="00:00:03.35">and we were trying to figure out
what to do about the internet.</p>
      <p begin="00:00:06.42" dur="00:00:03.15">And the company was in dire
straights at the time.</p>
      <p begin="00:00:09.57" dur="00:00:01.45">We were a CD-ROM authoring
company,</p>
      <p begin="00:00:11.42" dur="00:00:02.00">and the CD-ROM business was
going away.</p>
      <p begin="00:00:13.57" dur="00:00:02.50">One of the technologies I
remember seeing was Flash.</p>
      <p begin="00:00:16.47" dur="00:00:02.00">At the time, it was called <span
tts:fontWeight="bold" tts:color="#ccc333">FutureSplash</span>.</p>
      <p begin="00:00:18.50" dur="00:00:01.20">So this is where Flash got its
start.</p>
      <p begin="00:00:20.10" dur="00:00:03.00">This is smart sketch running on
the <span tts:fontStyle="italic">EU-pin computer</span>,</p>
      <p begin="00:00:23.52" dur="00:00:02.00">which was the first product that
FutureWave did.</p>
      <p begin="00:00:25.52" dur="00:00:02.00">So our vision for this product
was to</p>
      <p begin="00:00:27.52" dur="00:00:01.10">make drawing on the computer</p>
      <p begin="00:00:29.02" dur="00:00:01.30" style="1">as <span
tts:color="#ccc333">easy</span> as drawing on paper.</p>
    </div>
  </body>
</tt>

```

将提示点用于字幕

提示点允许您与视频进行交互：例如，您可以影响 FLV 文件的播放或在视频播放到特定时间时显示文本。如果没有与 FLV 文件一起使用的 **Timed Text XML** 文件，可以在 FLV 文件中嵌入事件提示点，然后将这些提示点与文本相关联。本节介绍有关 **FLVPlaybackCaptioning** 组件提示点标准的信息，以及如何将这些提示点与字幕文本相关联。有关如何通过视频导入向导或 **Flash Video Encoder** 嵌入事件提示点的详细信息，请参阅[《使用 Flash》](#)中的第 16 章，“使用视频”。

了解 FLVPlaybackCaptioning 提示点标准

在 FLV 文件的元数据中，提示点表示为带有以下属性的对象：`name`、`time`、`type` 和 `parameters`。**FLVPlaybackCaptioning** **ActionScript** 提示点具有以下属性：

- **“name”** — `name` 属性是一个字符串，其中包含为提示点分配的名称。`name` 属性必须以 `“fl.video.caption.2.0.”` 为前缀并且后跟一个字符串。该字符串是一系列正整数，每次都会增加以使每个名称保持唯一。前缀包括也与 **FLVPlayback** 版本号匹配的版本号。对于 **Adobe Flash CS3**，您必须将版本号设置为 2.0。
- **“time”** — `time` 属性是应显示字幕的时间。
- **“type”** — `type` 属性是一个值为 `“event”` 的字符串。
- **“parameters”** — `parameters` 属性是一个支持以下名称 - 值对的数组：
 - `text:String`。用于字幕的 **HTML** 格式的文本。此文本直接传递给 `TextField.htmlText` 属性。**FLVPlaybackCaptioning** 组件支持可选的 `text:n` 属性，该属性支持使用多语言轨道。有关详细信息，请参阅第 222 页的[“支持带有嵌入的事件提示点的多语言轨道”](#)。
 - `endTime:Number`。字幕应消失的时间。如果您不指定此属性，**FLVPlaybackCaptioning** 组件将假定该属性不是一个数字 (`NaN`)，并且字幕将始终显示，直到 FLV 文件结束 (**FLVPlayback** 实例调度 `VideoEvent.COMPLETE` 事件)。以秒为单位指定 `endTime:Number` 属性。`backgroundColor:uint`。此参数设置 `TextField.backgroundColor`。此属性为可选属性。
 - `backgroundColorAlpha:Boolean`。如果 `backgroundColor` 的 `alpha` 值为 0%，则此参数设置 `TextField.background = !backgroundColor`。此属性为可选属性。
 - `wrapOption:Boolean`。此参数将设置 `TextField.wordWrap`。此属性为可选属性。

了解如何为嵌入的事件提示点创建字幕

如果您没有包含 FLV 文件字幕的 Timed Text XML 文件，可以通过将包含字幕的 XML 文件与嵌入的事件提示点进行关联来创建字幕。该 XML 范例假定您已执行以下步骤在您的视频中创建了嵌入的事件提示点：

- 添加事件提示点（遵循 FLVPlaybackCaptioning 标准），然后对视频进行编码。
- 在 Flash 中，将一个 FLVPlayback 组件和一个 FLVPlaybackCaptioning 组件拖到舞台中。
- 设置 FLVPlayback 和 FLVPlaybackCaptioning 组件的 source 属性（即您的 FLV 文件和 XML 文件的位置）。
- 发布。

以下范例将 XML 导入到编码器中。

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<FLVCoreCuePoints>
```

```
  <CuePoint>
    <Time>9136</Time>
    <Type>event</Type>
    <Name>fl.video.caption.2.0.index1</Name>
    <Parameters>
      <Parameter>
        <Name>text</Name>
        <Value><![CDATA[Captioning text for the first cue point]]></Value>
      </Parameter>
    </Parameters>
  </CuePoint>
```

```
  <CuePoint>
    <Time>19327</Time>
    <Type>event</Type>
    <Name>fl.video.caption.2.0.index2</Name>
    <Parameters>
      <Parameter>
        <Name>text</Name>
        <Value><![CDATA[Captioning text for the second cue point]]></Value>
      </Parameter>
    </Parameters>
  </CuePoint>
```

```
  <CuePoint>
    <Time>24247</Time>
    <Type>event</Type>
    <Name>fl.video.caption.2.0.index3</Name>
    <Parameters>
      <Parameter>
```

```

        <Name>text</Name>
        <Value><![CDATA[Captioning text for the third cue point]]></Value>
    </Parameter>
</Parameters>
</CuePoint>

<CuePoint>
    <Time>36546</Time>
    <Type>event</Type>
    <Name>fl.video.caption.2.0.index4</Name>
    <Parameters>
        <Parameter>
            <Name>text</Name>
            <Value><![CDATA[Captioning text for the fourth cue point]]></Value>
        </Parameter>
    </Parameters>
</CuePoint>

</FLVCoreCuePoints>

```

此外，`FLVPlaybackCaptioning` 组件还支持带有嵌入提示点的多语言轨道。有关详细信息，请参阅第 222 页的“支持带有嵌入的事件提示点的多语言轨道”。

支持带有嵌入的事件提示点的多语言轨道

只要 Timed Text XML 文件符合 `FLVPlaybackCaptioning` 提示点标准，`FLVPlaybackCaptioning` 的 `track` 属性还可支持带有嵌入提示点的多语言轨道。（有关详细信息，请参阅第 220 页的“了解 `FLVPlaybackCaptioning` 提示点标准”。）但是，`FLVPlaybackCaptioning` 组件不支持单独的 XML 文件中的多语言轨道。若要使用 `track` 属性，请将该属性设为不等于 0 的值。例如，如果将 `track` 属性设置为 1 (`track == 1`)，则 `FLVPlaybackCaptioning` 组件将搜索提示点参数。如果没有找到匹配项，将使用提示点参数中的 `text` 属性。有关详细信息，请参阅《[ActionScript 3.0 语言和组件参考](#)》中的 `track` 属性。

播放多个带有字幕的 FLV 文件

您可以在 `FLVPlayback` 组件的单个实例内打开多个视频播放器，以播放多个视频并在这些视频播放时在它们之间切换。您也可以将字幕与 `FLVPlayback` 组件中的每个视频播放器相关联。有关如何打开多个视频播放器的详细信息，请参阅第 190 页的“使用多个视频播放器”。若要在多个视频播放器中使用字幕，请为每个 `VideoPlayer` 创建一个 `FLVPlaybackCaptioning` 组件的实例，并将 `FLVPlaybackCaptioning` `videoPlayerIndex` 设为对应的索引。如果仅存在一个 `VideoPlayer`，`VideoPlayer` 索引默认为 0。

以下代码示例将唯一的字幕分配给唯一的视频。若要运行此示例，必须将该示例中虚构的 URL 替换为工作 URL。

```
captioner0.videoPlayerIndex = 0;
captioner0.captionURL = "http://www.[yourDomain].com/mytimedtext0.xml";
flvPlayback.play("http://www.[yourDomain].com/myvideo0.flv");
captioner1.videoPlayerIndex = 1;
captioner1.captionURL = "http://www.[yourDomain].com/mytimedtext1.xml";
flvPlayback.activeVideoIndex = 1;
flvPlayback.play ("http://www.[yourDomain].com/myvideo1.flv");
```

自定义 FLVPlaybackCaptioning 组件

若要迅速开始使用 FLVPlaybackCaptioning 组件，可以选择使用将字幕直接放到 FLVPlayback 组件上的 FLVPlaybackCaptioning 默认值。您可能希望自定义 FLVPlaybackCaptioning 组件以将字幕从视频中移出。

以下代码演示如何通过切换字幕按钮动态创建 FLVPlayback 对象。

通过切换字幕按钮动态创建 FLVPlayback 对象

1. 将 FLVPlayback 组件放在舞台的 (0,0) 处，然后输入实例名称 **player**。
2. 在舞台的 (0,0) 坐标上放置 FLVPlaybackCaptioning 组件，并提供实例名称 **captioning**。
3. 将 CaptionButton 组件放在舞台上。
4. 在下面的代码示例中，将 `testVideoPath:String` 变量设置为 FLV 文件（使用绝对或相对路径）。



该代码示例将 `testVideoPath` 变量设置为 Flash 视频范例 `caption_video.flv`。将此变量更改为字幕视频组件（您将向其添加字幕按钮组件）的路径。

5. 在下面的代码示例中，将 `testCaptioningPath:String` 变量设置为适当的 Timed Text XML 文件（使用绝对或相对路径）。



该代码示例将 `testCaptioningPath` 变量设置为 Timed Text XML 文件 `caption_video.xml`。将此变量更改为包含视频字幕的 Timed Text XML 文件的路径。

6. 将 FLVPlayback 和 FLVPlaybackCaptioning 组件添加到库中。
7. 将下面的代码保存在与 FLA 文件相同的目录中，命名为 `FLVPlaybackCaptioningExample.as`。

8. 将 FLA 文件中的 DocumentClass 设置为 FLVPlaybackCaptioningExample。

```
package
{
    import flash.display.Sprite;
    import flash.text.TextField;
    import fl.video.FLVPlayback;
    import fl.video.FLVPlaybackCaptioning;

    public class FLVPlaybackCaptioningExample extends Sprite {

        private var testVideoPath:String = "http://www.helpexamples.com/flash/
video/caption_video.flv";
        private var testCaptioningPath:String = "http://www.helpexamples.com/
flash/video/caption_video.xml";

        public function FLVPlaybackCaptioningExample() {
            player.source = testVideoPath;
            player.skin = "SkinOverAllNoCaption.swf";
            player.skinBackgroundColor = 0x666666;
            player.skinBackgroundAlpha = 0.5;

            captioning.flvPlayback = player;
            captioning.source = testCaptioningPath;
            captioning.autoLayout = false;
            captioning.addEventListener("captionChange",onCaptionChange);
        }
        private function onCaptionChange(e:*):void {
            var tf:* = e.target.captionTarget;
            var player:FLVPlayback = e.target.flvPlayback;

            // move the caption below the video
            tf.y = 210;
        }
    }
}
```

有关所有 **FLVPlaybackCaptioning** 参数的详细信息，请参阅 [《ActionScript 3.0 语言和组件参考》](#)。

Timed Text 标签

FLVPlaybackCaptioning 组件支持用于字幕 XML 文件的 Timed Text 标签。有关音视频 Timed Text 标签的详细信息，请查看 <http://www.w3.org> 上的信息。下表列出了支持和不支持的标签。

功能	标签 / 值	用法 / 描述	示例
忽略的标签	metadata	在文档的所有级别上忽略或允许此标签	
	set	在文档的所有级别上忽略或允许此标签	
	xml:lang	忽略此标签	
	xml:space	忽略此标签 / 将行为覆盖为 xml:space="default"	
	layout	忽略此标签 / 包括 layout 标签部分中的所有 region 标签	
	br 标签	忽略所有属性和内容。	
字幕的媒体定时	begin 属性	仅允许在 p 标签中使用。部署字幕媒体时间时需要此属性。	<p begin="3s">
	dur 属性	仅允许在 p 标签中使用。建议使用。如果未包括此属性，则字幕将在 FLV 文件结束或另一字幕开始时结束。	
	end 属性	仅允许在 p 标签中使用。建议使用。如果未包括此属性，则字幕将在 FLV 文件结束或另一字幕开始时结束。	

功能	标签 / 值	用法 / 描述	示例
字幕的时钟 定时	00:03:00.1	完整时钟格式	
	03:00.1	部分时钟格式	
	10	不带单位的偏移时间。偏移值以秒为单位。	
	00:03:00:05	不支持。不支持含有帧或刻度的	
	00:03:00:05.1 30f 30t	时间格式。	
Body 标签	body	必需 / 只支持一个 body 标签。	<body><div>...</div></body>
Content 标签	div 标签	可以没有，也可以有一个或多个。使用第一个标签。	
	p 标签	可以没有，也可以有一个或多个。	
	span 标签	一系列文本内容单元的逻辑容器。不支持嵌套的 span。支持属性 style 标签。	
	br 标签	表示显式换行符。	
样式标签 (所有 style 标签均在 p 标 签内使用)	style	引用一个或多个 style 元素。可以用作标签和属性。用作标签时，需要 ID 属性（可在文档内重复使用该样式）。支持在 style 标签内部有一个或多个 style 标签。	
	tts:background Color	指定用于定义区域的背景颜色的样式属性。如果 Alpha 值未设置为零 (Alpha 0)，则忽略 Alpha 值，以便使背景透明。颜色格式为 #RRGGBBAA	

功能	标签 / 值	用法 / 描述	示例
	tts:color	指定用于定义前景颜色的样式属性。对于任何颜色，均不支持使用 Alpha。值 transparent 对应黑色。	<pre><style id="3" style="2" tts:backgroundColor="white"/> "transparent" = #00000000 "black" = #000000FF "silver" = #C0C0C0FF "grey" = #808080FF "white" = #FFFFFFFF "maroon" = #800000FF "red" = #FF0000FF "purple" = #800080FF "fuchsia"("magenta") = #FF00FFFF "green" = #008000FF "lime" = #00FF00FF "olive" = #808000FF "yellow" = #FFFF00FF "navy" = #000080FF "blue" = #0000FFFF "teal" = #008080FF "aqua"("cyan") = #00FFFFFF</pre>
	tts:fontFamily	指定用于定义字体系列的样式属性。	<pre>"default" = _serif "monospace" = _typewriter "sansSerif" = _sans "serif" = _serif "monospaceSansSerif" = _typewriter "monospaceSerif" = _typewriter "proportionalSansSerif" = _sans</pre>
	tts:fontSize	指定用于定义字体大小的样式属性。如果提供两个值，则仅使用第一个（垂直）值。忽略百分比值和单位。支持绝对像素大小（例如 12）和相对样式大小（例如 +2）。	

功能	标签 / 值	用法 / 描述	示例
	tts:fontStyle	指定用于定义字体样式的样式属性。	"normal" "italic" "inherit"* * 默认行为；从封套标签继承此样式。
	tts:fontWeight	指定用于定义字体粗细的样式属性。	"normal" "bold" "inherit"* * 默认行为；从封套标签继承此样式。
	tts:textAlign	指定样式属性，该属性用于定义在包含区块区域内如何对齐内嵌区域。	"left" "right" "center" "start" (= "left") "end" (= "right") "inherit"* * 从封套标签继承此样式。如果未设置 textAlign 标签，则默认值为 "left"。
	tts:wrapOption	指定样式属性，该属性用于定义在受影响元素的上下文中是否应用自动换行。此设置对字幕元素中的所有段落都起作用。	"wrap" "noWrap" "inherit"* * 从封套标签继承此样式。如果未设置 wrapOption 标签，则默认值为 "wrap"。

功能	标签 / 值	用法 / 描述	示例
不支持的属性	tts:direction tts: display tts: displayAlign tts: dynamicFlow tts: extent tts: lineHeight tts: opacity tts: origin tts: overflow tts: padding tts: showBackgro und tts: textOutline tts: unicodeBidi tts:visibility tts: writingMode tts: zIndex		

索引

英文

ActionScript

- 添加 FLVPlayback 组件 178
- 添加组件使用 20
- 组件 API 34

ActionScript 提示点

- FLVPlayback 184
- 禁用和启用 185
- 删除 185
- 添加 185

ActionScript, 创建

- Button 72
- CheckBox 75
- ColorPicker 77
- ComboBox 81
- DataGrid 86
- DataProvider 52
- Label 89
- List 94
- NumericStepper 96
- ProgressBar 103
- RadioButton 106
- ScrollPane 109
- Slider 112
- TextArea 115
- TextInput 119
- TileList 122
- UILoader 124
- UIScrollBar 126

addChild() 方法 20, 46

addChildAt() 方法 46

addEventListener() 方法, 事件处理中 22

addItem() 方法 54, 55

addItemAt() 方法 55

API, 组件 15

Button 组件

- 参数 70
- 创建 71
- 进行交互 70
- 使用 69
- 使用 ActionScript 创建 72
- 使用外观 139
- 使用样式 138
- 自定义 137

CellRenderer

- 设置单元格格式 59
- 实现 ICellRenderer 61
- 使用 59
- 使用 SWF 66
- 使用库元件 62
- 使用图像 66
- 使用影片剪辑 66
- 属性 65
- 为可编辑单元格 66

CheckBox 组件

- 参数 74
- 创建 74
- 进行交互 73
- 使用 73
- 使用 ActionScript 创建 75
- 使用外观 141
- 使用样式 140
- 自定义 140

ColorPicker 组件

- 参数 76
- 创建 77
- 进行交互 76
- 使用 76
- 使用 ActionScript 创建 77
- 使用外观 143
- 使用样式 142

- 在 Greetings 应用程序中 24
- 在 Greetings2 应用程序中 27
- 自定义 142
- ComboBox 组件
 - 参数 79
 - 创建 80
 - 进行交互 79
 - 使用 78
 - 使用 ActionScript 创建 81
 - 使用外观 146
 - 使用样式 145
 - 在 Greetings 应用程序中 25
 - 在 Greetings2 应用程序中 27
 - 自定义 144
- Component Assets 文件夹 41
- DataGrid 对象, 应用 CellRenderer 65
- DataGrid 组件
 - 参数 84
 - 创建 84
 - 进行交互 82
 - 使用 82
 - 使用 ActionScript 创建 86
 - 使用外观 151
 - 使用样式 147
 - 用 XML 填充 87
 - 自定义 147
- DataProvider
 - merge() 57
 - sort() 57
 - sortOn() 57
 - 操作 55
 - 创建 50
 - 删除项目 56
 - 使用 50
 - 使用 ActionScript 创建 52
 - 使用 Array 52
 - 填充 List 92
 - 显示数据字段 54
- dataProvider 参数 50
- DataProvider 对象, 使用 XML 55
- defaultPushButton 属性 49
- fl/accessibility/package-detail.html 67
- Flash Media Service 193
- Flash 视频提示点对话框 184
- FLV 文件
 - 播放 175
 - 播放多个 190
 - 切换 191
 - 选项 181

- FLVPlayback 组件
 - 播放多个 FLV 190
 - 参数 179
 - 查找版本 22
 - 创建外观 202
 - 创建应用程序 177, 215
 - 流式加载 FLV 文件 193
 - 使用 175
 - 使用 ActionScript 添加 178
 - 使用 SMIL 文件 207
 - 使用视频播放器 190
 - 使用视频导入向导添加 178
 - 使用提示点 183
 - 说明 175
 - 预先设计的外观 194
 - 指定 source 参数 180
 - 自定义 193
 - 组件参数 217
- FLVPlaybackCaptioning 组件
 - 查找版本 22
 - 从组件面板中添加 216
 - 多语言轨道支持 222
 - 将嵌入的事件提示点用于字幕 221
 - 使用 215
 - 使用 ActionScript 添加 216
 - 使用 Timed Text 字幕 218
 - 提示点标准 220
- FocusManager, 使用 48
- getChild() 方法 46
- getChildAt() 方法 46
- getChildByName() 方法 46
- getStyle() 方法 131
- Greetings 应用程序
 - ColorPicker 在 24
 - ComboBox 在 25
 - RadioButton 在 24
 - TextArea 在 24
 - 在 FLA 文件中创建 24
 - 在外部类文件中 26
- ICellRenderer 接口, 实现 61
- Label 组件
 - 参数 88
 - 创建 88
 - 进行交互 88
 - 使用 88
 - 使用 ActionScript 创建 89
 - 使用外观 153
 - 使用样式 153
 - 自定义 153

List 组件

- 参数 91
- 创建 92
- 进行交互 90
- 使用 90
- 使用 `ActionScript` 创建 94
- 使用外观 155
- 使用样式 154
- 用 `DataProvider` 填充 92
- 与 `MovieClip` 进行交互 93
- 自定义 154

`numChildren` 属性 46, 47

NumericStepper 组件

- 参数 96
- 创建应用程序 96
- 进行交互 95
- 使用 95
- 使用 `ActionScript` 创建 96
- 使用外观 158
- 使用样式 157
- 自定义 157

`on(event)` 22

ProgressBar 组件

- 参数 99
- 创建 99
- 创建应用程序 99
- 进行交互 98
- 使用 98
- 使用 `ActionScript` 创建 103
- 使用外观 160
- 使用样式 159
- 在轮询模式下 100
- 在事件模式下 99
- 在手动模式下 101
- 自定义 159

RadioButton 组件

- 参数 105
- 创建 105
- 创建应用程序 105
- 进行交互 104
- 使用 104
- 使用 `ActionScript` 创建 106
- 使用外观 162
- 使用样式 161
- 在 `Greetings` 应用程序中 24
- 在 `Greetings2` 应用程序中 27
- 自定义 161

ScrollPane 组件

- 参数 108
- 创建 109
- 创建应用程序 109
- 进行交互 108
- 使用 107
- 使用 `ActionScript` 创建 109
- 使用外观 164
- 使用样式 163
- 自定义 163

`setFocus()` 方法 48

`setSize()` 方法 42

Slider 组件

- 参数 111
- 创建 111
- 进行交互 111
- 使用 110
- 使用 `ActionScript` 创建 112
- 使用外观 165
- 使用样式 164
- 自定义 164

SMIL 文件, 指定位置 180

SWC

- 导出 33
- 和 `FLVPlayback` 33
- 和 `FLVPlaybackCaptioning` 33
- 组件作为 33

SWC, 在基于 FLA 的组件中 32

TextArea 组件

- 参数 114
- 创建 115
- 进行交互 114
- 使用 113
- 使用 `ActionScript` 创建 115
- 使用外观 167
- 使用样式 166
- 在 `Greetings` 应用程序中 24
- 在 `Greetings2` 应用程序中 27
- 自定义 166

`TextFormat`, 设置文本属性 132

TextInput 组件

- 参数 117
- 创建 117
- 进行交互 117
- 使用 116
- 使用 `ActionScript` 创建 119
- 使用外观 169
- 使用样式 168
- 自定义 168

TileList 组件

- 参数 121
- 创建 121
- 进行交互 120
- 使用 120
- 使用 ActionScript 创建 122
- 使用外观 171
- 使用样式 170
- 自定义 170

Timed Text 标签 225

UI 组件

- 查找版本 21
- 类型 17

UIComponent 类和组件继承 34

UILoader 组件

- 参数 123
- 创建 124
- 进行交互 123
- 使用 123
- 使用 ActionScript 创建 124
- 自定义 172

UIScrollBar 组件

- 参数 125
- 创建 125
- 进行交互 125
- 使用 125
- 使用 ActionScript 创建 126
- 使用外观 173
- 使用样式 173
- 自定义 172

A

安装组件 17, 19

B

版本

- 为 FLVPlayback 查找 22
- 为 FLVPlaybackCaptioning 查找 22
- 为 UI 组件查找 21

包 34

本节简要介绍了受支持的 218

本文档的目标读者 11

编译

- 和嵌入的 SWC 32
- 影片剪辑 33

C

参数

- Button 组件 70
- CheckBox 组件 74
- ColorPicker 组件 76
- ComboBox 组件 79
- DataGrid 组件 84
- FLVPlayback 组件 179
- Label 组件 88
- List 组件 91
- NumericStepper 组件 96
- ProgressBar 组件 99
- RadioButton 组件 105
- ScrollPane 组件 108
- Slider 组件 111
- TextArea 组件 114
- TextInput 组件 117
- TileList 组件 121
- UILoader 组件 123
- UIScrollBar 组件 125
- 输入 38

重新加载组件 36

处理事件 43

创作, 添加组件 19

D

单元格

- 可编辑, CellRenderer 66
- 在基于 List 的组件中 49

F

访问默认样式 131

辅助功能, 组件 66

J

基于 FLA 的组件 31

基于 List 的组件

- 和单元格 49
- 和单元格渲染器 50
- 和数据提供者 49
- 使用 49

继承, 在组件中 34

简单的应用程序 23

K

可编辑单元格 66

库

编译剪辑 40

库面板 40

库面板中的编译剪辑 40

L

类和组件继承 34

类路径 36

P

屏幕阅读器 66

S

删除组件 21

设置单元格格式 59

设置外观

Button 组件 139

CheckBox 组件 141

ColorPicker 组件 143

ComboBox 组件 146

DataGrid 组件 151

FLVPlayback 组件 176

Label 组件 153

List 组件 155

NumericStepper 组件 158

ProgressBar 组件 160

RadioButton 组件 162

ScrollPane 组件 164

Slider 组件 165

TextArea 组件 167

TextInput 组件 169

TileList 组件 171

UIScrollBar 组件 173

定义 130

设置文本属性 132

实例, 设置样式 131

实时预览 43

示例, 运行 30

事件

处理 43

和侦听器 44

事件对象 44

事件处理

addEventListener() 方法 22

与 ActionScript 2.0 的区别 22

视频播放器, 使用 190

视频导入向导 178

属性

CellRenderer 65

设置 39

属性检查器 20

T

提示点, FLVPlayback 183

Flash 视频提示点对话框 184

查找 187

导航, 搜索 188

启用和禁用嵌入式提示点 188

删除 189

使用 183

侦听 186

体系结构, 组件 31

调试组件应用程序 37

调整组件大小 42

W

外部类文件 26

外观

创建 136

定义 134

关于 133

预先设计, FLVPlayback 194

在库中 41

在库中访问 135

在舞台上访问 134

为所有组件设置样式 133

文档

Adobe 开发人员中心和 Adobe 设计中心 13

关于 12

术语指南 12

文档中的术语 12

X

下载组件, Adobe Exchange 19

显示列表

删除组件 47

添加到 46

移动组件 46

显示列表, 使用 45

Y

样式

- 访问默认 131
- 了解设置 131
- 设置 130
- 在组件实例上设置 131

样式, 用于

- Button 138
- CheckBox 140
- ColorPicker 142
- ComboBox 145
- DataGrid 147
- Label 153
- List 154
- NumericStepper 157
- ProgressBar 159
- RadioButton 161
- ScrollPane 163
- Slider 164
- TextArea 166
- TextInput 168
- TileList 170
- UIScrollBar 173

印刷惯例 12

影片剪辑

- 编译 33

优点 16

源文件

- 和类路径 36
- 修改 36

源文件, 位置 35

运行示例 30

Z

侦听器, 事件 44

资源, 其它 Adobe 13

自定义, 关于 130

自定义组件 15

组件

- ActionScript API 15
- UI 类型 17
- 请参阅各个组件名称
- 安装 17, 19
- 查看 18
- 处理事件 43

创建外观 136

调试 37

调整大小 42

功能 16

关于 15

和 addChild() 方法 20

和实时预览 43

和显示列表 45

基于 FLA 的 31

基于 SWC 的 33

基于用户的位置 35

继承 34

类路径 36

嵌入的 SWC 32

删除 21

设置参数 20

设置属性 39

设置样式 132

使具有辅助功能 66

使用 ActionScript 添加 20

体系结构 31

添加到文档 19

添加和删除 19

外观, 基于 FLA 的 32

为所有组件设置样式 133

文件夹位置 34

下载 19

修改文件 36

一个简单的应用程序 23

优点 16

源文件, 位置 35

在创作时添加 19

在容器中的深度 45

在运行时添加 20

重新加载 36

自定义 15

组件参数

请参阅各个组件名称

查看 38

设置 38

组件的系统要求 12

组件面板 18

组件实例

获取样式 131

设置样式 131

在所有组件实例上设置样式 132