

BJC BIT COMPONENT SET

User guide

1 Table of contents

2	1.	Overview.....	6
3	1.1	Styles.....	6
4	1.2	Setting styles at author time.....	6
5	1.3	Setting styles with ActionScript.....	7
6	1.4	Skinning.....	8
7	1.5	Focus management.....	9
8	1.6	Tab indexing and focus rectangle.....	10
9	1.7	Keyboard control.....	11
10	2.	Using the components.....	12
11	1.8	General.....	12
12	1.9	Accordion (com.bjc.controls.Accordion).....	13
13	1.9.1	About the Accordion.....	13
14	1.9.2	Using the Accordion.....	13
15	1.9.3	Styling the Accordion.....	14
16	1.9.4	Skinning the Accordion.....	14
17	1.10	Alert (com.bjc.controls.Alert).....	14
18	1.10.1	About the Alert.....	14
19	1.10.2	Using the Alert.....	14
20	1.10.3	Styling the Alert.....	15
21	1.10.4	Skinning the Alert.....	15
22	1.11	PushButton (com.bjc.controls.PushButton).....	15
23	1.11.1	About the PushButton.....	15
24	1.11.2	Using the PushButton.....	15
25	1.11.3	Styling the PushButton.....	16
26	1.11.4	Skinning the PushButton.....	16
27	1.12	Calendar (com.bjc.controls.Calendar).....	16
28	1.12.1	About the Calendar.....	16
29	1.12.2	Using the Calendar.....	16
30	1.12.3	Styling the Calendar.....	17
31	1.12.4	Skinning the Calendar.....	17
32	1.13	CheckBox (com.bjc.controls.CheckBox).....	17
33	1.13.1	About the CheckBox.....	17
34	1.13.2	Using the CheckBox.....	18
35	1.13.3	Styling the CheckBox.....	18
36	1.13.4	Skinning the CheckBox.....	18
37	1.14	ColorChooser (com.bjc.controls.ColorChooser).....	18
38	1.14.1	About the ColorChooser.....	18
39	1.14.2	Using the ColorChooser.....	18
40	1.14.3	Styling the ColorChooser.....	19
41	1.14.4	Skinning the ColorChooser.....	19
42	1.15	ComboBox (com.bjc.controls.ComboBox).....	19

43	1.15.1	About the ComboBox	19
44	1.15.2	Using the ComboBox	19
45	1.15.3	Styling the ComboBox.....	20
46	1.15.4	Skinning the ComboBox.....	20
47	1.16	GroupBox (com.bjc.controls.GroupBox)	20
48	1.16.1	About the GroupBox.....	20
49	1.16.2	Using the GroupBox.....	20
50	1.16.3	Styling the GroupBox	21
51	1.16.4	Skinning the GroupBox.....	21
52	1.17	IconButton (com.bjc.controls.IconButton)	21
53	1.17.1	About the IconButton.....	21
54	1.17.2	Using the IconButton	21
55	1.17.3	Styling the IconButton	21
56	1.17.4	Skinning the IconButton	21
57	1.18	Knob (com.bjc.controls.Knob).....	21
58	1.18.1	About the Knob	21
59	1.18.2	Using the Knob	21
60	1.18.3	Styling the Knob.....	22
61	1.18.4	Skinning the Knob	22
62	1.19	Label (com.bjc.controls.Label).....	22
63	1.19.1	About the Label.....	22
64	1.19.2	Using the Label.....	22
65	1.19.3	Styling the Label	22
66	1.19.4	Skinning the Label	22
67	1.20	BitList (com.bjc.controls.BitList).....	22
68	1.20.1	About the BitList.....	22
69	1.20.2	Using the BitList.....	22
70	1.20.3	Styling the BitList	23
71	1.20.4	Skinning the BitList.....	23
72	1.21	Loader (com.bjc.controls.Loader).....	23
73	1.21.1	About the Loader	23
74	1.21.2	Using the Loader	24
75	1.21.3	Styling the Loader.....	24
76	1.21.4	Skinning the Loader	24
77	1.22	NumericStepper (com.bjc.controls.NumericStepper).....	24
78	1.22.1	About the NumericStepper.....	24
79	1.22.2	Using the NumericStepper.....	24
80	1.22.3	Styling the NumericStepper.....	24
81	1.22.4	Skinning the NumericStepper.....	25
82	1.23	PlayerControl (com.bjc.controls.PlayerControl)	25
83	1.23.1	About the PlayerControl.....	25
84	1.23.2	Using the PlayerControl	25
85	1.23.3	Styling the PlayerControl.....	25
86	1.23.4	Skinning the PlayerControl	25

87	1.24	ProgressBar (com.bjc.controls.ProgressBar)	25
88	1.24.1	About the ProgressBar	25
89	1.24.2	Using the ProgressBar	25
90	1.24.3	Styling the ProgressBar	26
91	1.24.4	Skinning the ProgressBar	26
92	1.25	RadioButton (com.bjc.controls.RadioButton).....	26
93	1.25.1	About the RadioButton	26
94	1.25.2	Using the RadioButton	26
95	1.25.3	Styling the RadioButton.....	27
96	1.25.4	Skinning the RadioButton	27
97	1.26	RichTextArea (com.bjc.controls.RichTextArea)	27
98	1.26.1	About the RichTextArea	27
99	1.26.2	Using the RichTextArea	27
100	1.26.3	Styling the RichTextArea.....	28
101	1.26.4	Skinning the RichTextArea	29
102	1.27	ScrollBar (com.bjc.controls.ScrollBar, HorizScrollBar, VertScrollBar).....	29
103	1.27.1	About the ScrollBar.....	29
104	1.27.2	Using the ScrollBar.....	29
105	1.27.3	Styling the ScrollBar	29
106	1.27.4	Skinning the ScrollBar	30
107	1.28	ScrollPane (com.bjc.controls.ScrollPane).....	30
108	1.28.1	About the ScrollPane.....	30
109	1.28.2	Using the ScrollPane	30
110	1.28.3	Styling the ScrollPane	31
111	1.28.4	Skinning the ScrollPane	31
112	1.29	Slider (com.bjc.controls.HSlider, VSlider)	31
113	1.29.1	About the Slider	31
114	1.29.2	Using the Slider	31
115	1.29.3	Styling the Slider.....	31
116	1.29.4	Skinning the Slider.....	31
117	1.30	TabPane (com.bjc.controls.TabPane).....	32
118	1.30.1	About the TabPane.....	32
119	1.30.2	Using the TabPane.....	32
120	1.30.3	Styling the TabPane	32
121	1.30.4	Skinning the TabPane.....	32
122	1.31	TextArea (com.bjc.controls.TextArea).....	33
123	1.31.1	About the TextArea.....	33
124	1.31.2	Using the TextArea.....	33
125	1.31.3	Styling the TextArea	33
126	1.31.4	Skinning the TextArea.....	33
127	1.32	TextInput (com.bjc.controls.TextInput).....	33
128	1.32.1	About the TextInput.....	33
129	1.32.2	Using the TextInput	33
130	1.32.3	Styling the TextInput.....	33

131	1.32.4	Skinning the TextInput	34
132	1.33	Resizers (com.bjc.resizers.Resizer, VResizer, HResizer).....	34
133	1.33.1	About the Resizers	34
134	1.33.2	Using the Resizers	34
135	1.33.3	Styling the Resizers.....	34
136	1.33.4	Skinning the Resizers.....	34
137	1.34	Tree (com.bjc.controls.Tree).....	34
138	1.34.1	About the Tree	34
139	1.34.2	Using the Tree	34
140	1.34.3	Styling the Tree.....	35
141	1.34.4	Skinning the Tree	35
142	1.35	VideoPlayer (com.bjc.controls.VideoPlayer)	35
143	1.35.1	About the VideoPlayer	35
144	1.35.2	Using the VideoPlayer.....	35
145	1.35.3	Styling the VideoPlayer.....	35
146	1.35.4	Skinning the VideoPlayer.....	36
147	1.36	Window (com.bjc.controls.Window)	36
148	1.36.1	About the Window	36
149	1.36.2	Using the Window.....	36
150	1.36.3	Styling the Window.....	36
151	1.36.4	Skinning the Window.....	36
152		Version history	37
153			

154 **1. Overview**

155 The BJC Bit components were created after extensive experience with Macromedia's "V2"
156 components (referred to as MMV2 components in this document) and are the result of an effort to
157 overcome some of the drawbacks of those components while implementing most of their positive
158 features. The major advantages in using the BJC Bit components are file size and ease of
159 skinning and styling. These components weigh in at about 1/3 the size of their MMV2 equivalents,
160 and most of the style properties have been created as inspectable component parameters,
161 meaning that you can easily change the styles right from the Component Inspector Panel and
162 immediately see the result in live preview.

163

164 Although the BJC Bit components are not based on the UIComponent/UIObject framework, every
165 effort has been made to ensure that they are compatible and you should encounter no problems
166 using components of both types within a single movie.

167

168 For the most part, you will find that the API's for the BJC Bit components match their MMV2
169 equivalents. There are minor differences, so please refer to the help files to check any
170 differences. The components belong to packages com.bjc.resizers, com.bjc.controls and
171 com.bjc.core, you will find the their API descriptions in these packages.

172 **1.1 Styles**

173 The major difference you will notice will be in styles. MMV2 components use the "setStyle"
174 method to apply styles to specific components, classes of components or to all components.
175 While somewhat convenient, this method, as implemented has several drawbacks. First, it can
176 cause a huge performance hit on your movie when called. SetStyle will iterate through every
177 component on stage in your movie, even if only called on a single instance. In a large application
178 with many components, this can actually hang the movie up long enough to generate a timeout
179 error. Secondly, the cascading nature of the styles can be very confusing. For example, if you set
180 a global style for "backgroundColor", you would expect that every component in the movie would
181 get that background color. Not so. Several components have that style set internally on a class
182 level, which overrides the global level. So your text areas for instance will not inherit that style.
183 Also, some styles set on containers such as accordions or windows will cascade down to the
184 contents of those containers, i.e. any components contained in them. Others will not. It can take
185 many rounds of trial and error to get the right style on a component, and suddenly it may change
186 when you set a style for another component!

187

188 The BJC Bit components were designed with all of these issues in mind, and a key design criteria
189 was ease of styling.

190 **1.2 Setting styles at author time.**

191 When you drag a component from the components panel onto the stage, you will see its
192 component parameters listed in the Property Inspector. This may include some style properties.
193 For the most part though, the styles will only appear in the Component Inspector, as this has
194 more room available to list the styles.

195

196 Any component that has a text label (such as the text on a button, a window title, the label for a
197 check box or radio button, etc.) will have at least six property styles shown in the Component

198 Inspector panel (and these will only appear in this panel, not the Property Inspector). These are
199 align, embedFont, fontColor, fontFace, fontSize and html. These are pretty self explanatory and
200 are explained in more detail in the help files. In addition, there is a disabledColor property which
201 will not appear in any inspector, but is accessible with ActionScript.

202

203 For components with a single label, these properties will apply to that label. For components with
204 multiple text labels, such as accordions, lists, combo boxes, these properties will apply to all text
205 shown in the component itself.

206

207 Note that if these properties are assigned to containers such as accordions or windows, the style
208 will *not* cascade to components included in the container's contents.

209

210 In addition to the text styles, components may have other styles listed in the Property Inspector or
211 Component Inspector, such as highlightColor, selectedColor, etc. Again, these will be detailed in
212 the help.

213

214 When you set any of the styles in the inspectors, the results should immediately reflect in the
215 component live preview.

216 **1.3 Setting styles with ActionScript**

217 In addition to setting styles via the inspectors, styles can be set at run time via ActionScript. This
218 is as simple as setting the property to a value, such as:

219

```
220 myButton.align = "left";
```

221

222 This sets the text in the label in the button to be left aligned.

223

224 Unlike MMV2 components, there is not a method to set global or class level styles. Styles must
225 be set on each instance of the component. However, there are a few shortcuts that make it not as
226 bad as it may seem.

227

228 First of all, you can create a style object and assign it to the component when you create or
229 attach it. A style object is simply a generic object that has a property for each style you want to
230 set on the component. For example, we can make a "buttonStyle" that sets the text to be left
231 aligned and red as follows:

232

```
233 buttonStyle = new Object();  
234 buttonStyle.align = "left";  
235 buttonStyle.fontColor = 0xff0000;
```

236

237 Or, you could use a shortcut, such as:

238

```
239 buttonStyle = {align:"left", fontColor:0xff0000};
```

240

241 Now, you just use that object as the `initObject` of any button you are creating or attaching that you
242 want to have that style:

243

```
244 attachMovie("PushButton", "myButton", 0, buttonStyle);
```

245

246 or

247

```
248 import com.bjc.controls.PushButton;  
249 PushButton.create(_root, "myButton", 0, buttonStyle);
```

250

251 Any properties on that `initObject` will be passed to the component when it is created.

252

253 So, you could set up a number of different styles at the beginning of your movie, such as a global
254 style, style for a particular class, or an individual custom style, and just apply the specific one you
255 want as you create the component.

256

257 But what about components that are already on stage? How do you apply a style object to them?
258 Simply by using the `style` property of the component. Say you already had a button on stage and
259 wanted to apply the `buttonStyle` to it. It is as simple as:

260

```
261 myButton.style = buttonStyle;
```

262

263 Note, it is important that the style object *only* contain valid styles for the particular component you
264 are setting the styles on. Any property on the style object will be copied over to the component,
265 so if you set a property to some non-style name, it could potentially overwrite some internal
266 property in the component, having unanticipated effects.

267

268 The seven most common styles that are used in the BJC Bit Components are: `align`,
269 `disabledColor`, `embedFont`, `fontColor`, `fontFace`, `fontSize` and `html`. These are used in all
270 components that have text labels.

271 1.4 Skinning

272 Skinning the BJC Bit components is pretty straightforward. All you have to do is create a movie
273 clip symbol in the library with a graphic in it, and export it with the correct skin name. OK, so how
274 do you know what the skin names are, and what they are for? Well, in the how-to section for each
275 component, the skin names are listed, so you can create a new movie clip symbol with those
276 names.

277

278 But there is an easier way. Open up one of the skin flas included with the BJC Bit components. In
279 the library, you will see a folder for each component that can be skinned. Simply drag that folder
280 into your movie. All the skins for that component will now be in your library. Just edit them and
281 change them to look the way you want.

282

283 Some additional differences here between BJC Bit components and the MMV2 components. One
284 is that the MMV2 components generally implement background colors and borders as styles. In

285 the BJC Bit components, there is usually a main skin for the component, such as `textAreaSkin`,
286 that contains the background and the border for the component. In the example skins provided,
287 most of those skins contain an instance of another symbol, `backgroundSkin`. Simply changing
288 that skin will change the background and border for every component that uses it.

289

290 There are other examples like this. For instance, you will see in the Base Skins folder, a core
291 skins folder which contains a number of gradient and solid fills. These are reused throughout
292 most of the skins, so changing those six symbols is often all you have to do to create a whole
293 new theme for every component in your movie.

294

295 Also, some of the MMV2 component skins are not graphical elements at all, but are drawn with
296 code. The accordion header is an example. In order to change the look of that skin, you would
297 have to subclass the class responsible for drawing it, and change the code. A bit over the top for
298 skinning. All of the BJC Bit components are skinned with simple graphic elements.

299 1.5 Focus management

300 Most of the BJC Bit Components broadcast *focus*, *tabFocus* and *killFocus* events. The *tabFocus*
301 event is broadcasted when user uses the tab button of the keyboard to set the focus. The *focus*
302 event is also broadcasted when the *tabFocus* event is broadcasted. To take advantage of the
303 focus events, you need to assign a event listener or a direct event handler to the components.
304 The following code example shows how to use event listener with a button component:

305

```
306 var focusListener:Object = new Object();  
307 focusListener.focus = function()  
308 {  
309     trace( "Component got focus" );  
310 }  
311  
312 focusListener.killFocus = function()  
313 {  
314     trace( "Component lost focus" );  
315 }  
316  
317 focusListener.tabFocus = function()  
318 {  
319     trace( "Component got tab focus" );  
320 }  
321 btn.addEventListener( "focus", focusListener );  
322 btn.addEventListener( "killFocus", focusListener );  
323 btn.addEventListener( "tabFocus", focusListener );
```

324

325 A shorter way to assign the focus handlers, is to use direct event handlers. The following example
326 shows how to use the direct event handlers with a `PushButton` component:

327

```
328 btn.focusHandler = function()  
329 {  
330     trace( "Component got focus" );  
331 }  
332  
333 btn.killFocusHandler = function()  
334 {  
335     trace( "Component lost focus" );  
336 }
```

```
337  
338 btn.tabFocusHandler = function()  
339 {  
340     trace( "Component got tab focus" );  
341 }
```

342 The tabFocus event can be used to create custom focus rectangles. Please notice that the
343 components will not broadcast the focus event if you click on the content movieclip. This is the
344 case with all components that load external content, such as movieclips or images.

345

346 The following components broadcast the focus events: Accrodion, Alert, BitList, Calendar,
347 CheckBox, ColorChooser, ComboBox, HSlider, IconButton, Knob, NumericStepper, PushButton,
348 RadioButton, RichTextArea, ScrollPane, TabPane, TextArea, TextInput, Tree, VSlider, Window.

349 1.6 Tab indexing and focus rectangle

350 Most of the components in the BJC Bit Component Set have a tabIndex property. It can be used
351 to define the order in which the components will receive focus when the tab key of the keyboard
352 is used to navigate in the movie. The tabIndex can be set in the following way:

353

```
354 myPushButton.tabIndex = 0;  
355 myVSlider.tabIndex = 1;  
356 myCheckBox.tabIndex = 2;
```

357

358 When a component receives focus with tab button, a focus rectangle will be displayed for a short
359 period of time. The focus rectangle can be skinned, it located in the skin folder in the library. You
360 can also adjust the time how long the focus rectangle will be visible. The default value is 2s, you
361 can set the time globally by setting the `_global.bitFocusTime` variable:

362

```
363 _global.bitFocusTime = 500; // Set the time to 500 ms
```

364

365 In addition, you can also set the time for each component instance individually by setting the
366 `focusTime` property:

367

```
368 myButton.focusTime = 4000; // Set the time to 4 seconds  
369 myList.focusTime = 1000; // Set the time to one second
```

370

371 The `focusTime` property overrides the global value and the default value. Both, the `focusTime` and
372 the `_global.bitFocusTime` properties, can be set to zero if you do not want to display the focus
373 rectangle.

374

375 The following components have the `tabIndex` and the `focusTime` properties: Accrodion, Alert,
376 BitList, Calendar, CheckBox, ColorChooser, ComboBox, HSlider, IconButton, Knob,
377 NumericStepper, PushButton, RadioButton, RichTextArea, ScrollPane, TabPane, TextArea,
378 TextInput, Tree, VSlider, Window.

379

380 1.7 Keyboard control

381 Most of the components in the BJC Bit Component Set can be controlled with keyboard. The
382 control keys are the arrow keys and the enter key. PushButton and IconButton are toggled when
383 the enter key or the space key is pressed. You can disable the keyboard controls for a single
384 component instance by setting the keyEnabled property to false:

385

```
386 myButton.keyEnabled = false;
```

387

388

2. Using the components

389

In this section, we will cover each component, how to use it, style it and skin it. Some components come in pairs, such as the HSlider and VSlider. These will simply be covered one time under the heading, Slider.

390

391

392

1.8 General

393

In general, using the components consists of five steps.

394

395

1. Putting the component on stage, sizing and positioning it. This can be done by physically moving it to where you want it and sizing it at author time, or dynamically at run time.

396

397

398

Components may be dynamically created at run time through the attachMovie function, like any other movie clip or component. One problem you might run into with that is if you are using data typing, attachMovie returns type MovieClip, and you may want to assign the reference to your newly attached clip to a reference which is typed as the component type. To overcome this, you can use the create method of the movie clip. Using the button as an example, here is the code:

399

400

401

402

403

404

405

```
import com.bjc.controls.PushButton;
var myButton:PushButton = PushButton.create(_root, "btn1", 0);
```

406

407

408

Here you see the create method is called directly from the PushButton class. It takes at least three arguments: Where to attach the new component, its instance name, and depth. You can also include an init object as in attachMovie, for assigning styles or other properties.

409

410

411

412

The create method returns a reference to the new component, typed as that component type, so you will not get an error assigning it to a variable of that type.

413

414

415

The component can then be positioned and sized using its move and setSize methods.

416

Remember that no matter which method you use, attach or create, the component needs to be in the movie's library and set to export first frame, or otherwise placed on stage somewhere in the movie in order for it to be included in the swf.

417

418

419

420

2. Now that you have the component where you want it, you can assign its properties, or component parameters. This can be done via the Property Inspector or Component Inspector at author time, or via code at run time.

421

422

423

424

Examples of this would be setting the label of a button, or the dataProvider of a list or combo box. This could also include adding items to a list with addItem, or adding sections to an accordion with addSection.

425

426

427

428

3. Next is styling. In the BJC Bit components, this is really no different than setting properties. You either do it via inspectors or setting them via code.

429

430

- 431 4. Though you might not do it in this order, skinning the component is the next logical step. This
432 involves creating or altering movie clip symbols for each skin element.
433
- 434 5. Finally, you handle component events. This is usually done by writing functions that will
435 respond to the events that the component generates. For example, the button generates a
436 click event. You can either create a clickHandler function on the button itself:

437

```
438 myButton.clickHandler = function(){  
439 // do something  
440 }
```

441

442 or you can add an event listener to the button, and create a click function that will respond to
443 the event:

444

```
445 myButton.addEventListener("click", this);  
446 function click(){  
447 // do something  
448 }
```

449

450 The subject of handling events is covered extensively in Macromedia's documentation, and
451 many web sites. The BJC Bit components are not handled any different in that respect. Just
452 check the help files to ensure you are handling or listening for the correct event names.

453 **1.9 Accordion (com.bjc.controls.Accordion)**

454 **1.9.1 About the Accordion**

455 The accordion is a container component consisting of a number of sections layed out vertically.
456 Each section has a header with a label, and an area below that for content. The content is in the
457 form of a movie clip symbol that is attached from the library. When you click on the header for a
458 section, that section will open up, and all the others will close. You must ensure that you make
459 the component tall enough to have enough vertical space to display all of the headers, with
460 enough left over to view your content.

461 **1.9.2 Using the Accordion**

462 To add sections to the accordion, you use the `addSection` method. This method takes two
463 arguments. The first is a title for the section, the second is the linkage name for a movie clip in the
464 library.

465

466 Example:

467

- 468 1. Drag an accordion component to the stage and size it to 100 x 200. Name it "myAccordion".
- 469 2. Create three movie clips with some visual content in them. Name them "content1", "content2",
470 and "content3". Set them to export, and export on first frame.
- 471 3. Add the following code to frame one:

472

```
473 myAccordion.addSection("Section 1", "content1");
```

```
474 myAccordion.addSection("Section 2", "content2");
475 myAccordion.addSection("Section 3", "content3");
```

476

477 Test the movie. You should see your accordion with three sections. Click on any section and
478 you should see the movie clip you created for that section.

479

480 To access the various movie clips in each section, use the `content` property. This is an array,
481 with an element for each section. Thus, `myAccordion.content[0]` will return a reference to the
482 movie clip attached in the first section, in our example, that would be the “content1” clip.

483

484 To know which section is currently open, refer to the `selectedIndex` property, which will return
485 the number of the currently open section. Thus

486 `myAccordion.content[myAccordion.selectedIndex]` will always return a reference to the open
487 section's content.

488

489 To respond to the user action of clicking on one of the headers, thus changing the active section,
490 handle the “change” event.

491 **1.9.3 Styling the Accordion**

492 The accordion has the seven text styles as described above. These will control the style of the
493 text seen in each accordion header.

494 **1.9.4 Skinning the Accordion**

495 The accordion has three skin symbols. The *accordionSkin* symbol is the background and border
496 for the entire component. Then there are two skins for the header states: *accordionHeaderUpSkin*
497 and *accordionHeaderDownSkin*. The “up” skin will be used for the header of any section that is
498 not active. The “down” skin will be used on the currently selected section header.

499 **1.10 Alert (com.bjc.controls.Alert)**

500 **1.10.1 About the Alert**

501 The alert component is used to create a popup window with a message.

502 **1.10.2 Using the Alert**

503 Generally you would add the Alert component to your library by dragging it to the stage and
504 deleting it and then creating an alert at run time using the create method. As an alert is usually
505 used as a response to some action or event, it is rare that you would leave an instance on stage
506 at author time. Remember, though, that it needs to be in the library and exported in order to be
507 available to create. The Alert.create method is a bit different than the other component create
508 methods. It does not require an instance name, as that is generated automatically, and it takes
509 two additional arguments for the message to be displayed in the alert, and the title shown in the
510 alert window. Here is an example:

511

```
512 import com.bjc.controls.Alert;
513 Alert.create(_root, 100, "This is an alert message.", "Alert Title");
```

514

515 The first two arguments are where to place the Alert and at what depth. Then there is the alert
516 message and the title string. You can also pass an init object as a last argument if you wish.

517 **1.10.3 Styling the Alert**

518 As the alert is essentially a Window component with a text field in it, styling the Alert is basically
519 the same as styling a window. Of course this would usually be done with ActionScript after
520 creating the alert, rather than on an instance at author time via the Component Inspector.

521 **1.10.4 Skinning the Alert**

522 Again, the Alert component contains a Window component and will be affected by the skins for
523 that component. See the section *Skining the Window* for specifics.

524 **1.11 PushButton (com.bjc.controls.PushButton)**

525 **1.11.1 About the PushButton**

526 The PushButton is a basic pushbutton. The “Push” is added to maintain compatibility with the
527 MMV2 components which often attach buttons internally with the linkage name “Button”.

528 **1.11.2 Using the PushButton**

529 The PushButton is simply placed on stage at author time, or at run time with attachMovie or
530 PushButton.create. The component has the seven standard text styles which affect the text label
531 on the button. The label parameter specifies the text that will appear on the button. The toggle
532 parameter sets the behavior of the button to be a simple pushbutton (when you release it, it will
533 come back up) or a toggle button (pushing once makes the button “down” or selected, pushing
534 again makes it back “up” or not selected). With the selected parameter, you can read whether or
535 not the button is currently selected, or put it into a selected or not selected state. You can use the
536 “click” event to determine when the button is clicked, either by adding an event listener, or writing
537 a function for the button’s clickHandler.

538

539 Example:

540

```
541 import com.bjc.controls.PushButton;  
542 var b1:PushButton = PushButton.create(_root, "b1", 0);  
543 var b2:PushButton = PushButton.create(_root, "b2", 1);  
544 b1.toggle = true;  
545 b2.toggle = true;  
546 b1.clickHandler = function(){  
547     b2.selected = !this.selected;  
548 }  
549 b2.clickHandler = function(){  
550     b1.selected = !this.selected;  
551 }
```

552

553 This code creates two buttons, b1 and b2. It sets them to toggle, and then sets clickHandlers on
554 each one of them that put the other button into the opposite state when clicked.

555 1.11.3 Styling the PushButton

556 The PushButton supports the seven text styles described above.

557 1.11.4 Skinning the PushButton

558 The PushButton has four skins associated with it; one for each of the up, over and down states.
559 These are: buttonUpSkin, buttonOverSkin, buttonDownSkin and buttonShadow. As described
560 above in the section on skinning, simply make three symbols in your library, exported with those
561 linkage names. The button will use those symbols over its internal skins. If you do not want the
562 button to have a shadow, simply make an empty movie clip exported with the linkage name,
563 buttonShadow.

564 1.12 Calendar (com.bjc.controls.Calendar)

565 1.12.1 About the Calendar

566 The calendar is simply that, a highly customizable calendar that will highlight today's date, the
567 currently selected date, and any dates that you want to add to its data provider.

568 1.12.2 Using the Calendar

569 Put the calendar on stage at author time or via attachMovie or Calendar.create. You can adjust
570 the size however you want and the dates will lay themselves out accordingly. There are two sets
571 of font styles as covered below. Functionally, the calendar allows you to store information under
572 specific dates. This data is stored in the component's dataProvider property, which is an array.
573 You add items to the data provider via the calendar's addItem method. This method has two
574 forms with slightly different syntax, but bot essentially take a date to store the information, and the
575 information itself. The date can be specified in a single argument which is a Date object, or as
576 three separate arguments, year, month and day. The data to store can be of any type. Here are
577 two examples showing both forms:

578

```
579 myCalendar.addItem(2004, 11, 25, "Christmas");  
580 // note that the month argument is zero-indexed, meaning January = 0,  
581 December = 11  
582  
583 var bday:Date = new Date(2004, 9, 20);  
584 myCalendar.addItem(bday, "My Birthday");
```

585

586 Again, the data parameter does not have to be a string, but can be any data type, including an
587 object with other properties, an reference to a movie clip, etc. Currently, only one item may be
588 stored in a particular date. Storing a new item in that date will overwrite the previously save data.
589 To store multiple items under one date, you could use an array or object as the data, and add
590 other specific data to that.

591

592 Additionally, you could create a separate array, fill it with objects and assign that as your
593 calendar's data provider. These objects would need to have four properties: year, month, date
594 and data, as described above. Here's an example:

595

```
596 var myDP:Array = new Array();  
597 myDP[0] = {year:2004, month:11, date:25, data:"Christmas"};
```

598
599

```
myDP[1] = {year:2004, month:9, date:20, data:"My Birthday"};  
myCalendar.dataProvider = myDP;
```

600

601 Items are removed with `removeItem`, which takes the year, month and day of the item to remove,
602 or `removeAll`, which clears all stored data.

603

604 You can have the calendar display different months by clicking on the arrows next to the name of
605 the month. Or you can do this programmatically by using the `setDate` method. Like the `addItem`
606 method, this can take a `Date` object or separate parameters for year, month and date. This will
607 cause the calendar to display the specified month and year.

608

609 When the user clicks on any date in the calendar, a “click” event will be generated. At that point,
610 the `selectedItem` property will return an object containing currently selected year, month, date and
611 data if any for that date.

612 1.12.3 Styling the Calendar

613 The calendar is highly customizable through its many style properties. It has two sets of five font
614 styles, one for the headers (the month, year and the headers for the days of the week), and one
615 for the actual dates themselves. The styles for `html` and `align` are not applicable and not included.
616 Otherwise these are the same as the seven font styles, prefixed with “header” or “date”. Example:
617 `dateFontSize`, `headerEmbedFont`.

618

619 In addition to the text styles, there are four color properties, `todayColor` is the color of the current
620 day, `rollOverColor` is the color a date will turn when the mouse is over it, `selectedColor` is the
621 color of the currently selected date, and `markedColor` is the color a date will turn when there is
622 some data stored under that date.

623

624 Finally, there are two styles that allow you to customize how the month and year will appear.
625 `MonthNames` is an array of strings. This defaults to the full names of the months – “January”,
626 “February”, etc., but you can put any strings in here, such as abbreviations or other languages.
627 `yearFormat` is a number from 2 to 4 specifying how you want the year to be displayed. A value of
628 4 will display the full year: 2004. A value of 3 will display a two-digit year with an apostrophe: ‘04.
629 And a value of 2 will simply show the last two digits: 04.

630 1.12.4 Skinning the Calendar

631 The majority of the customizing of the calendar is done through the styles above. There are three
632 skins. `calendarSkin` contains the background and border. `calendarBackBtn` and
633 `calendarForwardBtn` are the two skins for the arrows used to change the month. As described in
634 the skinning section above, simply create symbols in your library, exported with those linkage
635 names and the calendar will use those symbols instead of its internal skins.

636 1.13 CheckBox (com.bjc.controls.CheckBox)

637 1.13.1 About the CheckBox

638 The `CheckBox` component displays a small box with a label that can be checked or unchecked by
639 clicking on it.

640 **1.13.2 Using the CheckBox**

641 Place the checkbox on stage at author time or via attachMovie or CheckBox.create. The label can
642 be set via the label parameter. You can read whether or not the checkbox is checked with the
643 selected property. You can also programmatically check or uncheck the box by setting this
644 property to true or false. Example:

645

```
646 cb1.clickHandler = function(){  
647   cb2.selected = !this.selected;  
648 }  
649 cb2.clickHandler = function(){  
650   cb1.selected = !this.selected;  
651 }
```

652

653 This simply takes two checkboxes and when either one is clicked, it will change the other to the
654 opposite state.

655 **1.13.3 Styling the CheckBox**

656 The checkbox supports the seven text styles as described above.

657 **1.13.4 Skinning the CheckBox**

658 The checkbox has two skins, one for each of its states. checkBoxTrueSkin is the graphic shown
659 when the checkbox is checked and checkBoxFalseSkin when it is not checked. As described in
660 the skinning section above, create symbols in your library, exported with those linkage names,
661 and the component will use those instead of its internal skins.

662 **1.14 ColorChooser (com.bjc.controls.ColorChooser)**

663 **1.14.1 About the ColorChooser**

664 The color chooser duplicates very closely the color picker found in the Macromedia Flash
665 authoring environment or any other program that allows you to pick a specific color. Initially, the
666 user will only see a small square. When this is clicked on, a window will open up with a full range
667 of colors to choose from, or the user can enter a hex value manually. Clicking on a color or
668 pressing enter will close the window and cause the selected color to be displayed in the small
669 square.

670 **1.14.2 Using the ColorChooser**

671 The component is placed on stage at author time or via attachMovie or ColorChooser.create.
672 When a user chooses a color, a "change" event will be fired. You can use an event listener or the
673 changeHandler function to respond to the event. The currently selected color can be read from
674 the value property. You can also set the selected color by assigning a color value to this property.

675

676 Example:

677

```
678 // assuming there is a movie clip on stage named "my_mc"  
679 var myCol:Color = new Color(my_mc);  
680 myColorChooser.changeHandler = function(){
```

681
682

```
myCol.setRGB(this.value);  
}
```

683

684 This will cause the movie clip to change color to whatever color the user chose.

685 **1.14.3 Styling the ColorChooser**

686 There are no style properties for the color chooser.

687 **1.14.4 Skinning the ColorChooser**

688 The color chooser has a single skin, colorChooserSkin which is the window background and
689 border on which the color choices are displayed. You can use your own graphic for this symbol,
690 but it should be roughly the same size as the existing graphic. As described above under
691 skinning, simply make a movie clip symbol in the library, exported with that linkage name and the
692 color chooser will use that instead of its internal skin.

693 **1.15 ComboBox (com.bjc.controls.ComboBox)**

694 **1.15.1 About the ComboBox**

695 The ComboBox displays a single text label, and when clicked on will display a drop down list with
696 additional items. When the user clicks on one of the items in the list, the list will close and the
697 selected item will then be shown in the component.

698 **1.15.2 Using the ComboBox**

699 The combobox is put on stage at author time or at run time via attachMovie or ComboBox.create.
700 Items are added to the list with the addItem method. This method can take two types of
701 arguments. One would be a simple string. This string is the text that will be displayed in the list.
702 More often though, you would supply the addItem method with an object that contains two
703 properties: label and data. The label would be the string to display, while data can contain any
704 data type that you want to store in that slot. A similar method is addItemAt. In addition to the
705 string or object, this takes a number which is the position in the list to add the item. Remember
706 that the list is zero-indexed, meaning 0 would be the first position. Additionally, you could create
707 an array containing strings or objects, and assign that to the dataProvider property of the combo
708 box. Finally, in addition to the above ActionScript methods of adding items to the list, you can add
709 items at author time through the Property or Component Inspector panels. The labels parameter
710 allows you to enter a list of labels and the data property allows you to enter a list of data items
711 that will correspond to those labels. Note however, that in this case you can only add simple data
712 types such as strings and numbers as data. To remove items, use the removeItemAt method,
713 giving it the index position of the item you want to remove, or removeAll to clear out all of the
714 items.

715

716 When the user chooses a new item, a "click" event will be generated. This can be handled with an
717 event listener or the clickHandler method. The currently selected item will be available at that
718 point via the selectedItem property. You can also find the list position of the selected item via the
719 selectedIndex property.

720

721 Example:

722

```
723 myCB.addItem({label:"Paris", data:" France"});
724 myCB.addItem({label:"Berlin", data:" Germany"});
725 myCB.addItem({label:"Madrid", data:"Spain"});
726 myCB.addItem({label:"Moscow", data:"Russia"});
727
728 myCB.clickHandler = function(){
729     trace(this.selectedItem.label + " is in " + this.selectedItem.data);
730 }
```

731

732 **1.15.3 Styling the ComboBox**

733 The combo box supports the seven text styles as described above. These styles will be used for
734 both the text in the list as well as the displayed selected item. In addition, there are two color
735 properties: `highlightColor` and `selectedColor`, which specify the color of items in the list when they
736 are rolled over or selected. The `numRows` property specifies how many rows of data will be
737 shown in the drop down list, and `rowHeight` is the height in pixels of each row. If there are more
738 items in the list than can be shown in the specified number of rows, a scroll bar will appear,
739 allowing the user to scroll and view the additional items. The scroll bar width can be set with the
740 `scrollBarWidth` property.

741 **1.15.4 Skinning the ComboBox**

742 The combo box itself has three skins, `comboBoxSkin`, which is the background of the box in its
743 closed position, and `comboBoxBtnUpSkin` and `comboBoxBtnDownSkin` which are the skins for
744 the button used to open and close the combo box. As described above in the section on skinning,
745 simply make a symbol for any of the items you want to skin, and export it with that linkage name.
746 The combo box will use that skin instead of its internal one.

747

748 Additionally, the combo box contains a full list component, which contains a scrollbar component.
749 See the entries for those components for information on skinning them.

750 **1.16 GroupBox (com.bjc.controls.GroupBox)**

751 **1.16.1 About the GroupBox**

752 The group box has two possible uses. One would be merely as a visual border you can place
753 around screen elements to signify they belong together. More significantly though, it is designed
754 to work with a set of `RadioButton` components. Radio buttons are mutually exclusive in terms of
755 being selected. Any radio buttons on the timeline of a particular movie clip will by default be in the
756 same group. You can set them to be part of different groups by `ActionScript`, or you can use a
757 group box to do this automatically.

758

759 Any radio buttons that are placed within the bounds of a group box on the same timeline will be
760 part of a separate group, and will function as a unit separate from any other radio buttons on that
761 timeline.

762 **1.16.2 Using the GroupBox**

763 To use the group box to separate radio buttons, simply place it on stage at author time, or at run
764 time via `attachMovie` or `GroupBox.create`. Size and position it and then add some radio buttons,
765 placing them within the bounds of the group box.

766

767 As a test, you can now add some more radio buttons, placing them outside the bounds of the
768 group box. Test your movie and see that the two sets of radio buttons function as separate
769 groups.

770 **1.16.3 Styling the GroupBox**

771 There are no styles for the group box.

772 **1.16.4 Skinning the GroupBox**

773 The group box has one skin, named groupBoxSkin. To skin it, make a symbol exported from the
774 library with that linkage name. The group box will use that skin instead of its own internal skin.

775 **1.17 IconButton (com.bjc.controls.IconButton)**

776 **1.17.1 About the IconButton**

777 The IconButton component is functionally equivalent to the PushButton. It differs mainly in
778 appearance. You can easily set separate skins for each instance of the icon button, and it does
779 not have a built in label.

780 **1.17.2 Using the IconButton**

781 Place a icon button on stage manually, or at run time through attachMovie or IconButton.create.
782 Set the skins for each state of the button – up, over and down. This can be done via the
783 Component or Property Inspector, or with ActionScript. The properties are downIcon, overIcon
784 and upIcon. Simply assign strings to each of these properties. The strings correspond to the
785 linkage names of exported symbols in the library. The selected and toggle properties, as well as
786 the click event work exactly the same as the PushButton.

787 **1.17.3 Styling the IconButton**

788 There are no styles associated with the IconButton.

789 **1.17.4 Skinning the IconButton**

790 Skinning the icon button is done by assigning the skin names to the three icon properties as
791 described above.

792 **1.18 Knob (com.bjc.controls.Knob)**

793 **1.18.1 About the Knob**

794 The knob is a rotary control that can be used to set values. The user clicks on the knob and
795 moves his mouse to change the value. You can set the knob to respond to vertical, horizontal or
796 angular mouse movement, and set the sensitivity as well.

797 **1.18.2 Using the Knob**

798 Place the knob on stage manually or with attachMovie or Knob.create. You can set a maximum
799 and minimum value for the knob as well as set a starting value. The mode property can be set to

800 the string "horizontal", "vertical" or "angular". This specifies what type of mouse movement will
801 cause the knob to rotate. The sensitivity property controls how much mouse movement is
802 required to rotate the knob a given distance. Higher numbers are more sensitive, meaning a small
803 mouse movement will cause the knob to rotate more.

804

805 As the knob is rotated, its value property will constantly update, and "change" events will be fired.
806 These can be handled with an event listener or the changeHandler function.

807 **1.18.3 Styling the Knob**

808 There are no styles for the knob.

809 **1.18.4 Skinning the Knob**

810 The knob has two skins, knobSkin, which is the overall background skin, and knobHandle, which
811 is the small shape that rotates around and acts as a pointer or indicator to the knob's current
812 position. To skin these elements, make symbols exported from the library with those linkage
813 names. The knob will use those skins instead of its own internal skins.

814 **1.19 Label (com.bjc.controls.Label)**

815 **1.19.1 About the Label**

816 The label is a simple component for displaying text.

817 **1.19.2 Using the Label**

818 Put the label on stage at author time, or with attachMovie or Label.create. Size and position it and
819 assign some text to display using the text property.

820 **1.19.3 Styling the Label**

821 The label supports the seven text styles described above.

822 **1.19.4 Skinning the Label**

823 The label does not have any skins.

824 **1.20 BitList (com.bjc.controls.BitList)**

825 **1.20.1 About the BitList**

826 The list is used for displaying many items and allowing the user to choose one.

827 **1.20.2 Using the BitList**

828 The list is put on stage at author time or at run time via attachMovie or BitList.create. Items are
829 added to the list with the addItem method. This method can take two types of arguments. One
830 would be a simple string. This string is the text that will be displayed in the list. More often though,
831 you would supply the addItem method with an object that contains two properties: label and data.
832 The label would be the string to display, while data can contain any data type that you want to

833 store in that slot. A similar method is `addItemAt`. In addition to the string or object, this takes a
834 number which is the position in the list to add the item. Remember that the list is zero-indexed,
835 meaning 0 would be the first position. Additionally, you could create an array containing strings or
836 objects, and assign that to the `dataProvider` property of the list. Finally, in addition to the above
837 ActionScript methods of adding items to the list, you can add items at author time through the
838 Property or Component Inspector panels. The `labels` parameter allows you to enter a list of labels
839 and the `data` property allows you to enter a list of data items that will correspond to those labels.
840 Note however, that in this case you can only add simple data types such as strings and numbers
841 as data. To remove items, use the `removeItemAt` method, giving it the index position of the item
842 you want to remove, or `removeAll` to clear out all of the items.

843

844 When the user chooses a new item, a “click” event will be generated. This can be handled with an
845 event listener or the `clickHandler` method. The currently selected item will be available at that
846 point via the `selectedItem` property. You can also find the list position of the selected item via the
847 `selectedIndex` property.

848

849 Example:

850

```
851 myList.addItem({label:"Paris", data:" France"});  
852 myList.addItem({label:"Berlin", data:" Germany"});  
853 myList.addItem({label:"Madrid", data:"Spain"});  
854 myList.addItem({label:"Moscow", data:"Russia"});  
855  
856 myList.clickHandler = function(){  
857     trace(this.selectedItem.label + " is in " + this.selectedItem.data);  
858 }
```

859

860 **1.20.3 Styling the BitList**

861 The list supports the seven text styles as described above. These styles will be used for the text
862 in the list. In addition, there are two color properties: `highlightColor` and `selectedColor`, which
863 specify the color of items in the list when they are rolled over or selected. `rowHeight` is the height
864 in pixels of each row. If there are more items in the list than can be shown in the given space, a
865 scroll bar will appear, allowing the user to scroll and view the additional items. The scroll bar
866 width can be set with the `scrollBarWidth` property.

867 **1.20.4 Skinning the BitList**

868 The list itself supports the `listSkin` which contains the background and border. It also contains a
869 vertical scroll bar, which can also be skinned. See the scroll bar entry for information on its
870 particular skin elements. Simply make a symbol with the new skin and export it with the specified
871 skin name as a linkage name and the list will use that skin instead of its internal skin.

872 **1.21 Loader (com.bjc.controls.Loader)**

873 **1.21.1 About the Loader**

874 The loader allows you to easily load in external content such as a non-progressive jpeg image or
875 a swf. You can have the loaded content scale to its original size or to the size of the component.

876 **1.21.2 Using the Loader**

877 Put the loader on stage at author time or with `attachMovie` or `Loader.create`. The `url` property can
878 be set via `ActionScript` or the `Property` or `Component Inspector` panels. Then you can call the
879 `load` method which will start loading the content. Alternately you can set the `autoLoad` property to
880 `true`. This will cause the content to start downloading as soon as the `url` property is set. As soon
881 as the content has finished downloading, it will be displayed. If `autoScale` is set to `true`, the
882 content will be scaled to fit inside the component (aspect ratio will be retained). Otherwise the
883 content will be displayed at its original size.

884

885 As the content is loading, the loader will fire “progress” events, and when it has completely loaded
886 in content, it will fire a “load” event. These can be handled with event listeners or the
887 `progressHandler` or `loadHandler` functions. Also, while the content is loading, you can check the
888 value of the `progress` property which will be a number from zero to one hundred, indicating the
889 percentage of content that has loaded in.

890 **1.21.3 Styling the Loader**

891 The loader has no styles.

892 **1.21.4 Skinning the Loader**

893 The loader has no skins.

894 **1.22 NumericStepper (com.bjc.controls.NumericStepper)**

895 **1.22.1 About the NumericStepper**

896 The `NumericStepper` is a component that allows users to select numeric values. It is possible to
897 set the minimum and maximum values and the step. Also, you can set the number of decimals
898 that are displayed in the `NumericStepper` text field. If you set the number of the decimals to zero,
899 it will show as many decimals as there are in the number.

900 **1.22.2 Using the NumericStepper**

901 Just drag and drop the component to the stage, assign it an instance name and it is ready to be
902 used. The component broadcasts a change event when the value has been changed by user.
903 The following example shows how to assign a handler function to a numeric stepper:

904

```
905 myNumericStepper.changeHandler = function()  
906 {  
907     trace( myNumericStepper.value );  
908 }
```

909 **1.22.3 Styling the NumericStepper**

910 The `NumericStepper` inherits the `LabelWrapper` class, so it has the seven default styles described
911 earlier in this document: `align`, `disabledColor`, `embedFont`, `fontColor`, `fontFace`, `fontSize` and `html`.

912 **1.22.4 Skinning the NumericStepper**

913 The NumericStepper has seven skins. All skins are located in the library in folder Skins ->
914 NumericStepper Skins.

915

916 **1.23 PlayerControl (com.bjc.controls.PlayerControl)**

917 **1.23.1 About the PlayerControl**

918 The PlayerControl component is designed as an optional control interface to the VideoPlayer, and
919 a possible future MP3Player.

920 **1.23.2 Using the PlayerControl**

921 Put the player control on stage either at author time or with attachMovie or PlayerControl.create.
922 At that point all you need to do is assign the target parameter. This would be a reference to a
923 VideoPlayer component.

924

925 The component will show two buttons – a play/pause button and a rewind button.

926 **1.23.3 Styling the PlayerControl**

927 The player control does not have any styles.

928 **1.23.4 Skinning the PlayerControl**

929 The player control has several skins for its buttons. They are pretty self-explanatory by their
930 names: playerControlOverSkin, playerControlPauseSkin, playerControlPlaySkin,
931 playerControlRewindDownSkin, playerControlRewindOverSkin, playerControlRewindUpSkin. You
932 can make symbols exported from the library with those linkage names and the component will
933 use those skins instead of its internal skins for the buttons.

934 **1.24 ProgressBar (com.bjc.controls.ProgressBar)**

935 **1.24.1 About the ProgressBar**

936 The progress bar is a way of showing a changing value, usually one that will increase up to a
937 certain point such as the amount of content loaded or the amount of a song played.

938 **1.24.2 Using the ProgressBar**

939 Put the progress bar on stage at author time or with attachMovie or ProgressBar.create. Next,
940 you probably want to set the maximum property. This defaults to 100, which is useful if the value
941 you are working with is a percent. If you are loading in a jpeg or swf, you might want to set
942 maximum to the value retrieved from getBytesTotal() on the clip doing the loading.

943

944 Next you usually want to update the value property on a regular basis while it is changing. This
945 would often be in an onEnterFrame handler or setInterval. Each time the value is changed, the
946 bar will update to show how much of the total has been achieved.

947 **1.24.3 Styling the ProgressBar**

948 The progress bar is very customizable in terms of styles. First of all, it supports the seven text
949 styles covered above. These affect the text shown below the bar. What this text shows is also
950 customizable. By default it shows the percent of the value / maximum followed by “%”, for
951 example: “98 %”.

952

953 You can change what number is displayed with the `displayType` property, this takes a string
954 which can be “percent”, “value” or “valueTotal”. The default is “percent”. If you change it to
955 “value”, the number shown will simply be the value supplied. “valueTotal” will show both value
956 and maximum, separated by “ / “, for example: “99 / 100”.

957

958 You can further customize the number by specifying how many digits will be shown. Particularly
959 in showing percentages, you might get a long trailing fraction, such as 99.9999999. The value of
960 the `digits` property specifies how many digits to show after the decimal point. For instance, a
961 value of 2 would display 99.99 in the example above.

962

963 What appears after the number can be specified by the `suffix` property. It defaults to “%”, which is
964 fine if you are showing a percent. If you are using one of the value formats you might want to
965 change that to reflect what units are being shown, such as “bytes loaded”.

966

967 Finally, in terms of the label, you can turn off the label all together by setting `displayLabel` to `false`.

968

969 The last style-type property is `autoHide`. If set to `true`, the progress bar will disappear as soon as
970 value reaches maximum. At that point the content has loaded in, or whatever progress you were
971 keeping track of has reached its goal, and there is no need to continue showing a progress bar.

972 **1.24.4 Skinning the ProgressBar**

973 The progress bar has two skins: `progressBarFalseSkin`, which is the background of the bar, and
974 `progressBarTrueSkin` which is the part of the bar which appears to “grow” and show the progress.
975 Simply make two symbols in your library exported with those linkage names. The component will
976 use those skins instead of its internal skins.

977 **1.25 RadioButton (com.bjc.controls.RadioButton)**

978 **1.25.1 About the RadioButton**

979 The `RadioButton` component allows the user to choose from several mutually exclusive options. It
980 can be used in conjunction with the `GroupBox` to set up separate groups of options.

981 **1.25.2 Using the RadioButton**

982 Place two or more radio buttons on the stage at author time or with `attachMovie` or
983 `RadioButton.create`. Arrange them and set their labels by assigning some text to the `label`
984 property. By default any radio buttons placed in the same timeline will act as a group, meaning
985 that if you select one, all others will be unselected. There are a couple of ways of setting up
986 separate groups. One is to use the `GroupBox` component. See the entry for that component for
987 more information on how to use it. The other method is to set the `radioButtonGroupName` to a
988 different value for each set of radio buttons you want to act as a group. Here’s an example:

989

990

```
radioButton1.radioButtonGroupName = "group1";
```

991

```
radioButton2.radioButtonGroupName = "group1";
```

992

```
radioButton3.radioButtonGroupName = "group1";
```

993

994

```
radioButton4.radioButtonGroupName = "group2";
```

995

```
radioButton5.radioButtonGroupName = "group2";
```

996

```
radioButton6.radioButtonGroupName = "group2";
```

997

998

Here you can see that radio buttons 1 – 3 are part of group one, and will function separately from 4 – 6, which are part of group two.

999

1000

1001

For any radio button, the radioButtonGroup property is an array that contains all the radio buttons in that group.

1002

1003

1004

When a radio button is clicked on, it will generate a “click” event, and at that point its selected property will return true. Any other radio buttons in the same group will now be unselected.

1005

1006

1.25.3 Styling the RadioButton

1007

The radio button supports the seven text styles described above. These will be applied to the label of the radio button

1008

1009

1.25.4 Skinning the RadioButton

1010

The radio button has two skins – radioButtonFalseSkin, which is how it looks when it is not selected, and radioButtonTrueSkin, which is how it looks when it is selected. You can make symbols exported from the library with those linkage names and the radio button will use those as skins rather than the internal skins.

1011

1012

1013

1014

1.26 RichTextArea (com.bjc.controls.RichTextArea)

1015

1.26.1 About the RichTextArea

1016

The RichTextArea component makes it possible to create rich text editors. The component allows developer to decide which features to use, for example it is possible to create an editor with only bold and italic buttons. The component uses the TextFormat object in the formatting.

1017

1018

1019

1020

The component is based on the TextArea component, so without any control buttons it behaves like a normal TextArea component.

1021

1022

1.26.2 Using the RichTextArea

1023

First you need to drag the component to the stage. Then you need to assign the component an instance name, for example ‘rta’. The RichTextArea component broadcasts a *selection* event every time user changes the text selection. You can assign an event handler like this:

1024

1025

1026

1027

```
rta.selectionHandler = function( evt:Object ):Void
```

1028

```
{  
    trace( "RichTextArea selection changed" );
```

1029

```
1030     trace( "Selection begins: " + evt.begin );
1031     trace( "Selection ends: " + evt.end );
1032     trace( "Caret index: " + evt.caret );
1033 }
```

1034

1035 The component does not contain any control buttons by default, so you need to use for example
1036 IconButton components to format the user's document. You could add an IconButton to the stage
1037 and give it an instance name 'bold_btn', it would be then used to assign bold formatting to the
1038 selected text. Assuming you have the IconButton called 'bold_btn' (*toggle* property has to be true)
1039 and a RichTextArea called 'rta' on the stage, the following code will make the components work
1040 together:

1041

```
1042 // Called each time the text selection changes in the RichTextArea
1043 rta.selectionHandler = function( evt:Object ):Void
1044 {
1045     // The evt.format object contains the text format for the selection
1046     bold_btn.selected = evt.format.bold;
1047 }
1048
1049 // Called when user clicks on the IconButton
1050 bold_btn.clickHandler = function():Void
1051 {
1052     // Applies the bold format to the selection
1053     rta.setFormatProperty( "bold", bold.selected );
1054 }
```

1055

1056 As it can be seen in the previous example, the format of the selected text is passed to the
1057 selectionHandler function as a TextFormat object. The TextFormat class is described in the Flash
1058 documentation, this is not different from it in any way.

1059

1060 All formatting can be applied by using the setFormatProperty method of the RichTextArea
1061 component. The first parameter is the name of one of the properties in the TextFormat class. The
1062 second parameter is the value that is set to the format. Following code listing shows some
1063 examples on how to set the text format:

1064

```
1065 rta.setFormatProperty( "bold", true ); // Bold
1066 rta.setFormatProperty( "italic", true ); // Italic
1067 rta.setFormatProperty( "font", "Arial" ); // Font
1068 rta.setFormatProperty( "size", 16 ); // Size of the font
1069 rta.setFormatProperty( "color", 0xff0000 ); // Font color
1070 rta.setFormatProperty( "underline", true ); // Underlined
1071 rta.setFormatProperty( "url", "http://www.flashloaded.com" ); // L
1072 rta.setFormatProperty( "target", "_blank" ); // The link target
```

1073

1074 **1.26.3 Styling the RichTextArea**

1075 The RichTextArea component extends the TextArea component, please see the TextArea
1076 component documentation for more information.

1077 **1.26.4 Skinning the RichTextArea**

1078 The RichTextArea component extends the TextArea component, please see the TextArea
1079 component documentation for more information.

1080 **1.27 ScrollBar (com.bjc.controls.ScrollBar, HorizScrollBar, 1081 VertScrollBar)**

1082 **1.27.1 About the ScrollBar**

1083 ScrollBars are mainly for use within other components and can be useful if you are building your
1084 own components, though you may find other uses for them within an application. If you simply
1085 want a sliding component for choosing a value, you would do better using one of the Slider
1086 components. If you want to scroll a text field, you should consider using the TextArea component.

1087 **1.27.2 Using the ScrollBar**

1088 The slider is placed on stage at author time or with attachMovie or ScrollBar.create. The scrollbar
1089 has three main properties: maximum, minimum and value. It is valid to make minimum a greater
1090 value than maximum in order to reverse the direction the scroll bar works. The other key property
1091 is thumbScale. This is a value from 0 to 1, and represents the size of the thumb button in the
1092 middle of the bar, as a percentage of the full space. So if you set thumbScale to .5, it will fill half
1093 the slot in the bar.

1094

1095 The tough part of programming the scroll bar manually is determining the values for maximum,
1096 minimum and thumbScale. Generally in something like a scroll pane or text area, the thumb's
1097 scale would be a representation of the size of the visible content compared to the total size of the
1098 content. For example, if you had 100 lines of text, and only 15 were visible in a text area, you
1099 would set the thumbScale to .15. If the thumbScale is greater or equal to 1.0, the thumb will
1100 become invisible, as the content is now able to fit within the viewable area and no scrolling is
1101 necessary.

1102

1103 From a user's perspective, there are three ways to change the value of the scroll bar:

1104

- 1105 1. Dragging the thumb.
- 1106 2. Clicking or clicking and holding the arrow buttons on the top/bottom or left/right of the bar.
- 1107 3. Clicking on the slot behind the thumb.

1108

1109 One, of course, allows for smooth selection of values. Two will scroll the value in either direction
1110 when clicked, or start a continuous scroll if the mouse is held down. The amount it will scroll on
1111 clicking or holding can be set with the lineScroll property. This defaults to 1, which is useful for
1112 text scrolling, where you would probably want to scroll the text one line at a time.

1113

1114 Finally, clicking the slot will move the thumb in the direction of the mouse one unit, which should
1115 result in scrolling one "page" of content if the thumbScale is set correctly.

1116 **1.27.3 Styling the ScrollBar**

1117 The scroll bar does not support any styles.

1118 **1.27.4 Skinning the ScrollBar**

1119 The scroll bar has many skins, and these are doubled, as there are separate skins for vertical and
1120 horizontal scroll bars. First we'll cover the skins for the horizontal type:

1121

1122 hScrollBarBackSkin – the graphic for the back or slot of the bar.

1123 hScrollBarLeftBtnDownSkin – the down state of the left button.

1124 hScrollBarLeftBtnUpSkin – the up state of the left button.

1125 hScrollBarRightBtnDownSkin – the down state of the right button.

1126 hScrollBarRightBtnUpSkin – the up state of the right button.

1127 hScrollBarThumbSkin the graphic for the thumb.

1128

1129 The vertical skins are essentially the same, substituting the initial h with v, and Left/Right with
1130 Top/Bottom.

1131

1132 You can make symbols exported from the library with those linkage names and the scroll bar will
1133 use those as skins instead of its internal skins.

1134 **1.28 ScrollPane (com.bjc.controls.ScrollPane)**

1135 **1.28.1 About the ScrollPane**

1136 The scroll pane allows you to display content either attached from the library, or loaded in from an
1137 external source. If the content is larger than the size of the component, a pair of scroll bars will
1138 appear allowing you to scroll around to view the rest of the content, or you can drag it with the
1139 mouse. It also allows you to set a zoom factor to scale the content.

1140 **1.28.2 Using the ScrollPane**

1141 Put the scroll pane on stage at author time or with attachMovie or ScrollPane.create. Assign a
1142 value to its contentPath property. This can be either the linkage name of a symbol exported from
1143 the library, or the url of an external jpeg or swf. If it is a linkage name, that symbol will be
1144 immediately attached inside the scroll pane. If you supply a url, that content will begin to
1145 download and the scroll pane will fire "progress" events while it loads. When it is finished loading,
1146 a "load" event will fire. You can check the progress property at any time during the loading to get
1147 how much content has downloaded so far. This will be in the form of a number from 0 to 100,
1148 representing the percent of loaded content to the whole.

1149

1150 To access the content, use the content property. This will return a reference to the movie clip
1151 holding that content. Note that if you change the content programmatically in any way, such as
1152 adding elements, resizing or moving things around, the scroll bars of the scroll pane will not
1153 immediately update, as they have no way of knowing that you have changed the content. After
1154 any change to the content, you should call invalidate() on the scroll pane, to update its scroll bars.

1155

1156 Although you could scale the content manually with _xscale and _yscale and then invalidate the
1157 scroll pane, the zoom method does this all in one step. Simply set the zoom value of the scroll
1158 pane itself to whatever scale value you want and it will resize the contents and update the scroll
1159 bars.

1160

1161 The scroll pane also has `zoomWidth`, `zoomHeight` and `zoomFull` methods, which will resize the
1162 content so that it exactly fits in the content area vertically, horizontally or both.

1163

1164 Finally, if the content is larger than the visible area, you can move it around with the scroll bars or
1165 by clicking on the content and dragging it. In some cases, you may not want to drag the content.
1166 You can turn this feature off by setting the `dragContent` property to `false`.

1167 **1.28.3 Styling the ScrollPane**

1168 The only style-type property on the scroll pane is `scrollBarWidth`. Note that this will set both the
1169 width of the vertical scroll bar and the height of the horizontal scroll bar.

1170 **1.28.4 Skinning the ScrollPane**

1171 The scroll pane has two skins. `scrollPaneSkin` contains the background and border.
1172 `scrollPaneCornerSkin` contains the small square that appears in the lower right corner if both
1173 scroll bars are visible. Simply make symbols and export them from your library with those linkage
1174 names and the scroll pane will use those as skins instead of its internal skins. Note also that as
1175 the scroll pane contains a vertical and horizontal scroll bar, you can skin those as described in the
1176 Scroll Bar entry.

1177 **1.29 Slider (com.bjc.controls.HSlider, VSlider)**

1178 **1.29.1 About the Slider**

1179 The slider components allow the user to choose a numerical value by a sliding button. There is a
1180 horizontal and vertical version.

1181 **1.29.2 Using the Slider**

1182 Put the slider on stage at author time or with `attachMovie` or `HSlider.create` or `VSlider.create`.
1183 There are only three properties to worry about – maximum, minimum and value, which are pretty
1184 self-explanatory. It is valid to set minimum greater than maximum to reverse the direction of the
1185 slider.

1186

1187 When the user moves the slider, it will fire a “change” event. You can read the current value of
1188 the slider at any time through the `value` property. You can also set the slider’s value with this
1189 property.

1190

1191 **1.29.3 Styling the Slider**

1192 There are no styles for the slider.

1193 **1.29.4 Skinning the Slider**

1194 The horizontal slider has `hSliderBackSkin` and `hSliderThumbSkin`, which are the slot and the
1195 button of the slider. The vertical slider has the same, but with an initial `v` instead of `h`. Simply
1196 make symbols exported from the library with those linkage names and the slider will use those
1197 skins instead of the internal ones.

1198 **1.30 TabPane (com.bjc.controls.TabPane)**

1199 **1.30.1 About the TabPane**

1200 The TabPane component is pretty much functionally equivalent to the Accordion component, with
1201 a horizontal layout rather than vertical. You can add new tabs and assign content to each tab in
1202 the form of a linked movie clip, or externally loaded swf or jpeg.

1203 **1.30.2 Using the TabPane**

1204 Put the TabPane component on stage at author time or with attachMovie or TabPane.create. You
1205 add new tabs to the pane with the addTab method. This takes two or three arguments. First is a
1206 string for the title of the tab. Next is the content path. This can be the linkage name of an exported
1207 symbol in the library, or the url of an external swf or jpeg. You can also enter an icon name. This
1208 would again be the linkage name of a symbol in the library. It would contain the image you
1209 wanted to show as an icon.

1210

1211 You can also add a tab at a particular position with addTabAt. This takes a third argument of a
1212 number to put the tab at. Zero would be the first position. The icon, if used, would be the fourth
1213 argument.

1214

1215 You can remove a tab with removeTabAt, specifying the number to remove, or removeAll.

1216

1217 As you add tabs, they will resize to fill the width of the component. Obviously, if you add too
1218 many, they could become too small to be useful. Thus you can set a minimum width with the
1219 minTabWidth property. If adding a new tab would cause the tabs to be smaller than the minimum
1220 width, the action will fail and the tab will not be added.

1221

1222 To access the content of the tabs, you go through the content property. This is an array
1223 containing a reference to each content clip. For instance, myTabPane.content[0] would contain
1224 the content of the first tab.

1225

1226 When the user clicks on a tab, that content will become visible, and a “change” event will fire. You
1227 can find the currently selected tab number with the selectedIndex property.

1228 **1.30.3 Styling the TabPane**

1229 In addition to the seven text styles, which will affect the tab labels, you can set the tab height with
1230 the tabHeight property.

1231 **1.30.4 Skinning the TabPane**

1232 The tab pane has three skins. tabPaneSkin contains the background and border. tabSkin is the
1233 skin for a tab in the selected state, and tabBackSkin is the skin for a tab in the unselected state.
1234 Simply make symbols and export them from the library with those linkage names and the tab
1235 pane will use those skins instead of its internal skins.

1236 **1.31 TextArea (com.bjc.controls.TextArea)**

1237 **1.31.1 About the TextArea**

1238 The text area is a convenient way of displaying text or allowing a user to enter large quantities of
1239 text. It features an automatic scroll bar that appears when the text takes up more space than can
1240 be shown in the component.

1241 **1.31.2 Using the TextArea**

1242 Put the text area on stage at author time or with attachMovie or TextArea.create. Size and
1243 position it. You can add text to the component by assigning a string to the text property. If you are
1244 just using the text area to display text, you can set the editable property to false, which will
1245 prevent the user from changing the text.

1246

1247 The text area largely acts as a wrapper to the text field, and the following properties function
1248 essentially the same as the text field object: condenseWhite, maxChars, maxScroll, password,
1249 restrict, scroll and textHeight.

1250 **1.31.3 Styling the TextArea**

1251 The text area supports all the text styles as described above. It also has a scrollBarWidth
1252 property which lets you adjust the width of the vertical scroll bar that appears if there is more text
1253 than can fit in the visible area.

1254 **1.31.4 Skinning the TextArea**

1255 The text area has a single skin, textAreaSkin which contains the border and background of the
1256 skin. It also contains a vertical scroll bar and this can be skinned as described in the scroll bar
1257 listing.

1258 **1.32 TextInput (com.bjc.controls.TextInput)**

1259 **1.32.1 About the TextInput**

1260 The text input is largely similar to the text area component, but is always editable by definition,
1261 and is limited to a single line, so does not contain a scroll bar.

1262 **1.32.2 Using the TextInput**

1263 Put the text input on stage at author time or with attachMovie or TextInput.create. You can set the
1264 text to be shown in it by assigning a string to the text property. And, when the user has entered
1265 some text, you can read that by checking the value of the text property.

1266

1267 As the text input is a wrapper for a text field, the following properties work essentially the same as
1268 they do on the text field: maxChars, password, restrict and textHeight.

1269 **1.32.3 Styling the TextInput**

1270 The text area supports the seven text styles as described above.

1271 **1.32.4 Skinning the TextInput**

1272 The text input has a single skin: `textInputBackgroundSkin`. This contains the border and
1273 background. Simply make a new symbol in the library, exported with that linkage name and the
1274 text input will use that skin instead of its internal skin.

1275 **1.33 Resizers (com.bjc.resizers.Resizer, VResizer, HResizer)**

1276 **1.33.1 About the Resizers**

1277 The Resizer components (`Resizer`, `VResizer`, `HResizer`) enable smooth resizing of a predefined
1278 graphic element, keeping the margins the same and stretching the middle.

1279 **1.33.2 Using the Resizers**

1280 To use the resizers, you simply need to set the skin parameter of the resizer component.

1281 **1.33.3 Styling the Resizers**

1282 The Resizer component has four style properties; `bottomMargin`, `leftMargin`, `rightMargin` and the
1283 `topMargin`. By adjusting these style parameters, you can change the way the graphic is resized.

1284 **1.33.4 Skinning the Resizers**

1285 The graphic element is acts as the skin for the component. The component has no visible parts,
1286 only the content is visible.

1287 **1.34 Tree (com.bjc.controls.Tree)**

1288 **1.34.1 About the Tree**

1289 The tree is a way of showing hierarchical data in a tree form. It is based on an xml object which
1290 you need to create or load and assign as the tree's data provider.

1291 **1.34.2 Using the Tree**

1292 Put the tree on stage at author time or with `attachMovie` or `Tree.create`. Size and position it and
1293 assign an xml object to its `dataProvider` property. The xml object can be created with `ActionScript`
1294 or can contain xml loaded in from an external xml document. Each node in the xml document
1295 should have an attribute called "label", which is what the tree will display as a leaf. You can also
1296 include an attribute called "open" and set it to true or false. This will determine the initial state of
1297 that leaf if it has subnodes. If open is true, that leaf will be expanded to show all the subnodes. If
1298 false, it will be closed.

1299

1300 There are not methods on the tree itself for adding or altering data. All changes to the data should
1301 be made to the xml itself, and then you can invalidate the tree to cause it to update and show the
1302 new xml structure.

1303

1304 When a user clicks on a particular row in the tree, a "click" event will be fired. You can find the
1305 index of the selected row with the `selectedIndex` property, or you can get the actual xml node of

1306 that branch of the tree with the `selectedNode` property. You can also set a particular row to be
1307 selected by assigning a value to the `selectedIndex` property.

1308 **1.34.3 Styling the Tree**

1309 The tree supports many different style properties. Of course it supports the seven text styles
1310 described above. In addition:

1311

1312 `highlightColor` – the color when a row is moused over.

1313 `indentSize` – each new branch of the tree will be indented from its parent. This value specifies
1314 how many pixels to indent.

1315 `rowHeight` – simply the height in pixels for each row.

1316 `scrollBarWidth` – how thick to make the vertical scroll bar which will appear if there are more rows
1317 than can be displayed in the visible area.

1318 `selectedColor` – the color of the row that has been clicked on or selected.

1319 **1.34.4 Skinning the Tree**

1320 The tree supports three skins for the icons shown next to the labels. They are
1321 `treeClosedFolderIcon`, `treeOpenFolderIcon` and `treePageIcon`. You can make symbols and export
1322 them from the library with those linkage names and the tree will use those skins instead of its
1323 internal skins. In addition, the tree contains a list component which in turn contains a vertical
1324 scroll bar. Those can be skinned as well as described in their respective listings.

1325 **1.35 VideoPlayer (com.bjc.controls.VideoPlayer)**

1326 **1.35.1 About the VideoPlayer**

1327 The video player is a simple playback component for viewing flv files. Unlike the rest of the BJC
1328 Bit Components, this requires the Flash 7 player to function.

1329 **1.35.2 Using the VideoPlayer**

1330 Put the video player on stage at author time of with `attachMovie` or `VideoPlayer.create`. Size and
1331 position it. Assign it the url of a flv file. You can also set a buffer size which controls how much of
1332 the video will be preloaded in memory.

1333

1334 There are then three methods you can use to control the playback: `play` starts the video playing,
1335 `pause` stops it, keeping the playhead where it is, and `rewind` sets the playhead back to the
1336 beginning of the video. You can create buttons or other navigation elements that call these
1337 methods, or you can choose to control the video with the player control component, which simply
1338 takes a reference to the video player component and handles everything from there.

1339 **1.35.3 Styling the VideoPlayer**

1340 The video player has no styles.

1341 **1.35.4 Skinning the VideoPlayer**

1342 The video player has a single skin: videoPlayerSkin. This contains the background and border.
1343 You can make a symbol exported from the library with that linkage name and the player will use
1344 that instead of the internal skin.

1345 **1.36 Window (com.bjc.controls.Window)**

1346 **1.36.1 About the Window**

1347 The window is a holder for other content that allows that content to be dragged and moved
1348 around the stage inside a skinnable window.

1349 **1.36.2 Using the Window**

1350 Place the window on stage at author time or with attachMovie or Window.create. Assign a string
1351 to the contentPath property. This would be the linkage name of an exported symbol in the library.
1352 The symbol will be immediately attached inside the window. You can always get a reference to
1353 the content with the content property.

1354

1355 You can set a title for the window title bar by assigning a string to the title property.

1356

1357 By default, the window is a fixed size and is not closeable. By setting the closeable property to
1358 true, you can allow the user to close the window by clicking on a small button that appears in the
1359 top right of the window. By setting resizable to true, the bottom right corner of the window
1360 becomes draggable, allowing the user to change the size of the window.

1361 **1.36.3 Styling the Window**

1362 The window supports the seven text styles as described above. These will affect only the text in
1363 the title bar, if any.

1364 **1.36.4 Skinning the Window**

1365 The window has several skins:

1366

1367 windowCloseBtnSkin – this is the graphic that will be used as the close button.

1368 windowResizerSkin – this is the graphic that appears in the lower right corner, indicating that the
1369 window is resizable and allowing the user to drag it to resize the window.

1370 windowShadow – this is the shadow shape behind the window.

1371 windowSkin – this is the entire window skin, including the background, border and title bar.

1372

1373 Just create symbols in the library exported with those linkage names, and the window will use
1374 those skins instead of the internal skins.

