

# Creating ActionScript 3.0 components in Flash CS3 Professional – Part 7: Focus management

**Jeff Kameron**

Adobe

In this part of the article series on creating components using ActionScript 3.0, we'll discuss how to control focus in the MenuBar component. If you skipped the articles leading up to this section of the series, you might find it helpful to review the previous sections. In Part 1 of the series you can download the sample files for the entire series of articles all at once. Or if you prefer to just follow along with Part 7, you can download the sample files for this part of the series below.

Before I begin discussing how to support and control focus in the MenuBar component, let's take a look at our progress so far and see why this is important. If you've been following along with this article series, you'll remember that we added invalidation to our test.fla file. Since we've made many changes to the project, now is a good time to try Control > Test Movie to evaluate the way the menu bar is working. As you Test Movie, make sure that the option to Disable Keyboard Shortcuts is checked under the Control menu, then start hitting the tab key to change the focus. As you navigate through the top level of menu items, it will look as though everything is working at first. But now try clicking on the MenuBar to open one of the drop-down menus and hit the tab key several more times. You'll see the focus going to the TileList for the menu bar and then to the List for the drop-down menu separately, which is not the desired behavior. After making some tests, we've diagnosed the issue: the subcomponents are getting focus, but the MenuBar component is not. In this article, we'll discover how to resolve this, and how to control focus events using the FocusManger.

## Controlling and debugging focus issues using FocusManager

In order to ensure that our MenuBar component functions as users would expect it to, it's necessary to add focus support via ActionScript. Not only is this important for creating intuitive interactions with users – but it is also critical for ensuring that the drop-down menus will be accessible to specially-abled users who rely on using the keyboard, rather than the mouse, to select items in MenuBar component's drop-down menus.

### IFocusManagerComponent

Communicating to the User Interface Component Infrastructure that your component wants focus is very easy. Any class that can have focus needs to implement the interface `fl.managers.IFocusManagerComponent`. The changes to the ActionScript to support this in `MenuBar` were very straightforward.

Here's the code I used to achieve this:

```
import fl.managers.IFocusManagerComponent;

public class MenuBar extends UIComponent implements IFocusManagerComponent {
```

All of the classes for components implement this interface. Classes that are for display objects that are only used within components, like `fl.controls.listClasses.CellRenderer`, do not.

Implementing `IFocusManagerComponent` registers a component with the `FocusManager`, which is implemented by `fl.managers.FocusManager`, as a component that wants focus. If a display object does not inherit from `UIComponent`, then the `FocusManager` treats it using the normal rules for display objects. Buttons and movie clips do not have to do anything special to get the focus.

## focusEnabled

Implementing `IFocusManagerComponent` is all that is necessary to get focus working for a component that does not have any subcomponents. However, in our sample project the `MenuBar` component has components within it that also require focus. You can easily disable focus by setting the `focusEnabled` property to `false`. In `MenuBar`, I changed `configUI()` to set the `focusEnabled` property on `myMenuBar`. I also changed `createMenu()` to customize the drop-down menus, as shown in the code below:

```
override protected function configUI():void {
    // always call super.configUI() in your implementation
    super.configUI();

    // dynamically create myMenuBar
    myMenuBar = new MenuBarTileList();

    // keeps TileList from accepting focus, must be done before addChild
    myMenuBar.focusEnabled = false;

    // turning off selectable makes the TileList instance behave more like a
    menu bar
    myMenuBar.selectable = false;

    // need to listen for a mouseDown event on the menu bar TileList
    // to start the menu handling. Other mouse event handlers will
    // be added after the initial mouseDown on the TileList menu bar
    // and removed when the menus are closed.
    myMenuBar.addEventListener(MouseEvent.CLICK, menuBarMouseHandler);

    // finally, add myMenuBar
    addChild(myMenuBar);
}

private function createMenu():List {
    var theMenu:List = new MenuList();
```

```

    theMenu.visible = false;
    theMenu.selectable = false;
    // keeps TileList from accepting focus
    theMenu.focusEnabled = false;
    myMenus.push(theMenu);
    addChildAt(theMenu, 0);
    return theMenu;
}

```

The order of operations is very important here. The `focusEnabled` property must be set immediately after the component instances are created and before they are added to the display list. If you want to turn off focus for a component instance that is not dynamically created, then after setting the `focusEnabled` property you'll need to remove the object from the display list and add it again, like this:

```

myButton.focusEnabled = false;
removeChild(myButton);
addChild(myButton);

```

## Debugging Focus Problems

Initially I had some difficulties making this change to the code, because originally I had set `focusEnabled` after calling `addChild()` for `myMenuBar`. This caused the tab focus to stall on the `MenuBar` instance. After doing some troubleshooting and testing, I eventually discovered this was because `myMenuBar` was in the `FocusManager`'s list for focusable display objects, right after my `MenuBar` instance. When I hit the tab key, the focus would try to shift to `myMenuBar`, but it would fail since `focusEnabled` was set to `false`.

The first thing I did as I was debugging this issue was to add stage listeners for the focus events. Stage listeners can be a very helpful tool to help you identify where focus events, keyboard events and mouse events are going. The event's `currentTarget` property will always be the Stage, but the `target` property will indicate who received the event. To facilitate debugging, I temporarily added the following code to the frame script in `test.fla`:

```

stage.addEventListener(FocusEvent.FOCUS_IN, focusInHandler);
function focusInHandler(e:FocusEvent):void {
    trace("focus into " + e.target);
}

stage.addEventListener(FocusEvent.FOCUS_OUT, focusOutHandler);
function focusOutHandler(e:FocusEvent):void {
    trace("focus out of " + e.target);
}

stage.addEventListener(KeyboardEvent.KEY_DOWN, keyDownHandler);
function keyDownHandler(e:KeyboardEvent):void {
    trace("key down to " + e.target);
}

```

The traces output from these listeners all seemed normal, except I realized that once focus hit the MenuBar component, it never received another focusIn or focusOut event. My next step in this process was to use the debugger. I added the User Interface Component source to the classpath of test.fla and set some break points in `FocusManager`. After experimenting with adding break points in different methods, I had luck with a break point I added in `keyFocusChangeHandler()`. I stepped through the code into `setFocusToNextObject()` and from there into `getNextFocusManagerComponent()`, and it was while stepping through that method that I realized the `focusableCandidates` array had a `MenuBarTileList` instance in it, which was wrong. This made me realize that something must be going wrong when I set `focusEnabled` on the MenuBar component instance — which lead me to consider whether the timing of the call was at the root of the problem. In the next section of this article, we'll take a look at the steps I took to fix the focus issues.

## Resolving focus event display issues

At this point, the menu was almost working as expected, but I needed to make just a few more changes to the code in order to resolve issues with the way the drop-down menus were behaving. By adding some code to handle the focus events and control the order of display objects, I was able to make the MenuBar component retain the focus and work in an intuitive way.

### Handling focusOut Event

Focus was now tabbing in and out of the MenuBar component correctly, and the simple steps I had taken thus far would be enough for many components that you may develop. But the MenuBar component contains subcomponents. If I had a drop-down menu showing and I hit the tab key to move focus to the next component, the drop-down menu remained open. To fix this issue, I added code to handle the focusOut event.

If your component needs to handle either the focusIn or focusOut event, you do not need to add your own listener because `UIComponent` registers `focusInHandler()` and `focusOutHandler()` and you can override these methods. The `UIComponent` implementations handle drawing the `focusRectSkin`, so you should always call the `super` implementation.

For MenuBar, I overrode `focusOutHandler()` to close any open menus, as shown in the code example below:

```
override protected function focusOutHandler(e:FocusEvent):void {
    closeMenuBar();
    super.focusOutHandler(e);
}
```

## Customizing drawFocus

When the focusRectSkin was drawn around the MenuBar component and a drop-down menu was open, the drop-down menu would cover the focusRectSkin, and I did not like that behavior. Fortunately, I discovered I could change it by overriding `drawFocus()` and adding a few lines of code. I found this by reading the ActionScript source and by stepping through each line of code in the debugger. You will find there are many places you can tweak the behavior of the User Interface Component Infrastructure this way, but documenting them all is outside the scope of this article series. As you develop components, you will find that you'll need to do some research and analyze the code to determine the best way to update it.

The `UIComponent` implementation of `drawFocus()` adds the focusRectSkin at the bottom of the display list, so at this point I just needed to move it to the top.

Check out the code below to see how this was accomplished:

```
override public function drawFocus(focused:Boolean):void {
    super.drawFocus(focused);
    if (focused && uiFocusRect != null) {
        setChildIndex(uiFocusRect, numChildren - 1);
    }
}
```

## Where to go from here

In Part 8 of this article series we'll add keyboard support to the MenuBar component so that users can navigate through the drop-down menus using key presses (up and down arrows, the space bar, the escape and return keys) to make the menu bar easier to negotiate and more accessible.

If this article has made you curious about working with focus events and you'd like to learn more about how to control them with ActionScript 3.0, be sure to check out the [ActionScript 3.0 Language and Components Reference](#) sections that are listed below:

[ActionScript 3.0 documentation on FocusManager](#)

[ActionScript 3.0 documentation on IFocusManager](#)

[ActionScript 3.0 documentation on IFocusManagerComponent](#)

## About the author

Jeff Kameron is a computer scientist at Adobe Systems who has worked on the Flash authoring team since 2002.