

# Using and Animating Filters

---

## Import the Project

1. If not already created, create a directory named `adobeFlexTraining` on your **C** drive.
2. In Flex Builder, delete the **ReservationSystem** project if created previously. If you don't delete the contents the files will be overwritten with the new project files.
3. In Flex Builder, select **File > Import > Flex Project**.
4. In the dialog window, select **Archive File** and browse to where `Ex20_Starter.zip` is located in your local file system.  
**Note:** This starter file has the code you created through the end of Day 4 exercises. This exercise will only delve into a portion of that code. For more information, review Day 4 exercises.
5. Uncheck **Use default location**.
6. Enter `C:\adobeFlexTraining\ReservationSystem`.
7. Click **Finish**.

## Use a filter on the Submit button

8. Using the **Flex Navigator** view, open the `RequestTicket.xml` located in the **ReservationSystem > src > components** folder.  
Using the **Outline** view locate the `<mx:Button>` component. Change the `id` property value to `glowingButton`.

Your code should appear as follows.

```
<mx:FormItem>

    <mx:Button id="glowingButton"
        label="Submit" />

</mx:FormItem>
```

9. Locate the `Script` block.
10. In the `Script` block, after the creation of the last variable, create a private variable named `glow` data typed as `GlowFilter`.
11. Instantiate `glow` and pass the following parameters:
  - `color: 0x80ff00`

- alpha: 1
- blurX: 20
- blurry: 20

Your code should appear as follows:

```
private var glow:GlowFilter = new GlowFilter(0x80ff00, 1, 20, 20);
```

12. Before the existing function in the Script block, create a private function named `toggleGlow` that takes no parameters and returns `void`. Within the `toggleGlow` function, create a conditional statement that will test the `filters.length` of the `glowingButton` button to see if the value is equivalent to zero. This will be used to toggle the glow filter on the Button component. If there are no filters applied to the `glowingButton` button (referenced by the button id), using array bracket notation the glow is applied. If there is a glow filter applied to the button, remove the filter value within the brackets to turn it off.

Your code should appear as follows:

```
private function toggleGlow():void
{
    if (glowingButton.filters.length == 0)
    {
        glowingButton.filters = [glow];
    }
    else
    {
        glowingButton.filters = [];
    }
}
```

Your complete code should appear as follows:

```
<mx:Script>
  <![CDATA[
    [Bindable] public var selectedRoom:Object;
    private var eventList:String;
    private var glow:GlowFilter = new GlowFilter(0x80ff00,
1, 20, 20);

    private function toggleGlow():void{
      if (glowingButton.filters.length == 0)
```

```

        {
            glowingButton.filters = [glow];
        }
        else
        {
            glowingButton.filters = [];
        }
    }

    private function createlist(eData:XMLList):String{
        var a:Array = new Array();
        for each (var sData:String in
eData.child('event'))
            a.push(sData);

        eventList = a.toString();
        return eventList;
    }

]]>
</mx:Script>

```

13. Using the **Outline** view locate the `<mx:Button>` component.

14. Add the `click` property to the button and assign the `toggleGlow` function to it.

Your code should appear as follows.

```

<mx:FormItem>

    <mx:Button id="glowingButton"
        label="Submit"
        click = "toggleGlow()"/>

</mx:FormItem>

```

15. Save and run the `ReservationSystem.mxml` file.

16. Click the log in button.

17. Look at the Request Ticket form on the right hand side.

18. Click on the Submit button, a glow will be visible around the button.

19. Click on the button again to switch off the glow.

## Animating the filter on the Submit button

20. At the top of the `Script` block, import the `mx.effects.Tween` class. The **Tween** class defines a tween, a property animation performed on a target object over a period of time.
21. After the last variable creation, create a `private` variable named `tween` data typed to `Tween`.
22. Create one more `private` variable named `forward` data typed to `Boolean`.

Your code should appear as follows:

```
import mx.effects.Tween;

[Bindable] public var selectedRoom:Object;
private var eventList:String;
private var glow:GlowFilter = new GlowFilter(0x80ff00, 1, 20,
20);
private var tween:Tween;
private var forward:Boolean;
```

23. In the `toggleGlow()` function, if the conditional statement tests true, instantiate the `Tween`. The `Tween` instance accepts the `startValue`, `endValue` and `duration` properties. The keyword `this` is used for the listener object. Use these parameters: `this`, `0`, `1`, `500`.
24. Set the `forward` flag to `true` after the tween is instantiated.

Your code should appear as follows:

```
tween = new Tween(this, 0, 1, 500);
forward = true;
```

25. If the conditional statement tests false, use the `stop` method of the tween.

Your code should appear as follows:

```
tween.stop();
```

Your complete code should appear as follows:

```
import mx.effects.Tween;

[Bindable] public var selectedRoom:Object;
```

```

private var eventList:String;
private var glow:GlowFilter = new GlowFilter(0x80ff00, 1, 20,
20);
private var tween:Tween;
private var forward:Boolean;

private function toggleGlow():void
{
    if (glowingButton.filters.length == 0)
    {
        glowingButton.filters = [glow];
        tween = new Tween(this, 0, 1, 500);
        forward = true;
    }
    else
    {
        glowingButton.filters = [];
        tween.stop();
    }
}
. . .

```

The Tween object invokes a callback function at regular intervals for the duration of the effect, passing to the `onTweenUpdate` method an interpolated value between the `startValue` and `endValue` causing the object to animate. When the effect ends, the Tween object invokes the `onTweenEnd` callback function.

26. Create a private function named `onTweenUpdate` that takes a parameter named `value` datatype as `Number` and returns `void`.
27. Within the function, assign the variable `value` to the `alpha` property of the variable `glow`.
28. Using the array bracket notation, assign the `glow` to the `filters` property on the `glowingButton` button.

Your code should appear as follows:

```

public function onTweenUpdate(value:Number):void
{
    glow.alpha = value;
    glowingButton.filters = [glow];
}

```

29. Create a private function named `onTweenEnd` that accepts a parameter named `value` datatype as `Number` and returns `void`.

30. Within the function, assign the variable `value` to the `alpha` property of the variable `glow`.

Your code should appear as follows.

```
glow.alpha = value;
```

31. Using the array bracket notation, assign the `glow` to the filters on the `glowingButton` button.

Your code should appear as follows:

```
glowingButton.filters = [glow];
```

32. Using a conditional, test the forward **Boolean** flag. If `forward` tests true, instantiate the `tween` variable using these parameters in this order: `this`, `1`, `0`, `500`.

33. If `forward` tests false, instantiate the `tween` variable using these parameters in this order: `this`, `0`, `1`, `500`.

34. Assign `!forward` to `forward` to change the value.

Your code should appear as follows:

```
if (forward)
    tween = new Tween(this, 1, 0, 500);
else
    tween = new Tween(this, 0, 1, 500);
    forward = !forward;
```

Your code should appear as follows.

```
public function onTweenEnd(value:Number):void
{
    glow.alpha = value;
    glowingButton.filters = [glow];
    if (forward)
        tween = new Tween(this, 1, 0, 500);
    else
        tween = new Tween(this, 0, 1, 500);
    forward = !forward;
}
```

35. Save and run the `ReservationSystem.mxml` file.

After the login page, look at the Request Ticket form. If you click on the Submit button, the glow will automatically fade in and out on the button.