Drag and Drop Basics[*]

R.G. (Dick) Baldwin

Abstract

Learn the basics of writing ActionScript 3 code to provide a drag and drop capability.

NOTE: **Click** DragAndDrop01 [1] to run this ActionScript program. *(Click the "Back" button in your browser to return to this page.)*

1 Table of Contents

2 Preface

2.1 General

NOTE: All references to ActionScript in this lesson are references to version 3 or later.

[*]Version 1.1: May 25, 2010 4:06 pm -0500
[†]http://creativecommons.org/licenses/by/3.0/
[1]http://cnx.org/content/m34479/latest/DragAndDrop01.html

This tutorial lesson is part of a series of lessons dedicated to object-oriented programming (OOP) with ActionScript.

Several ways to create and launch ActionScript programs

There are several ways to create and launch programs written in the ActionScript programming language. Many of the lessons in this series will use Adobe Flex as the launch pad for the sample ActionScript programs.

An earlier lesson titled **The Default Application Container** provided information on how to get started programming with Adobe's Flex Builder 3. *(See Baldwin's Flex programming website [2] .)* You should study that lesson before embarking on the lessons in this series.

Some understanding of Flex MXML will be required

I also recommend that you study all of the lessons on Baldwin's Flex programming website in parallel with your study of these ActionScript lessons. Eventually you will probably need to understand both ActionScript and Flex and the relationships that exist between them in order to become a successful ActionScript programmer.

Will emphasize ActionScript code

It is often possible to use either ActionScript code or Flex MXML code to achieve the same result. Insofar as this series of lessons is concerned, the emphasis will be on ActionScript code even in those cases where Flex MXML code may be a suitable alternative.

2.2 Viewing tip

I recommend that you open another copy of this document in a separate browser window and use the following links to easily find and view the figures and listings while you are reading about them.

2.2.1 Figures

2.2.2 Listings

[2] http://www.dickbaldwin.com/tocFlex.htm

2.3 Supplemental material

I recommend that you also study the other lessons in my extensive collection of online programming tutorials. You will find a consolidated index at www.DickBaldwin.com [3] .

3 General Background Information

For Adobe online documentation on this topic, see Using Drag and Drop [4] .

A drag and drop operation is carried out in three stages:

- Initiation
- Dragging
- Dropping

Initiation

As you are probably already aware, initiation consists of the user pointing to an item with the mouse and pressing the mouse button.

Dragging

During the drag operation, the user drags the item to another location on the screen without releasing the mouse button.

Dropping

When the item has been dragged to the new location, the user releases the mouse button causing the item to remain in the new location.

Copying

It is also possible to copy an item using the drag and drop gestures, but that capability won't be illustrated in this lesson. Instead, this lesson will concentrate on moving a Flex component from one location in its container to a different location in its container.

Classes and events

The sample program that I will explain in this lesson will use the following classes, methods, and events:

- MouseEvent class
    - mouseDown event
- DragEvent class
    - dragDrop event
    - dragEnter event
- DragManager class
    - acceptDragDrop method
    - doDrag method
- DragSource class
    - addData method
    - hasFormat method

[3]http://www.dickbaldwin.com/toc.htm
[4]http://livedocs.adobe.com/flex/3/html/help.html?content=dragdrop_1.html

4 Preview

I will explain a program named **DragAndDrop01** . This program illustrates the fundamentals of drag and drop in ActionScript 3. The program places three images in the upper-left corner of a **Canvas** object as shown in Figure 1.

Program output at startup.

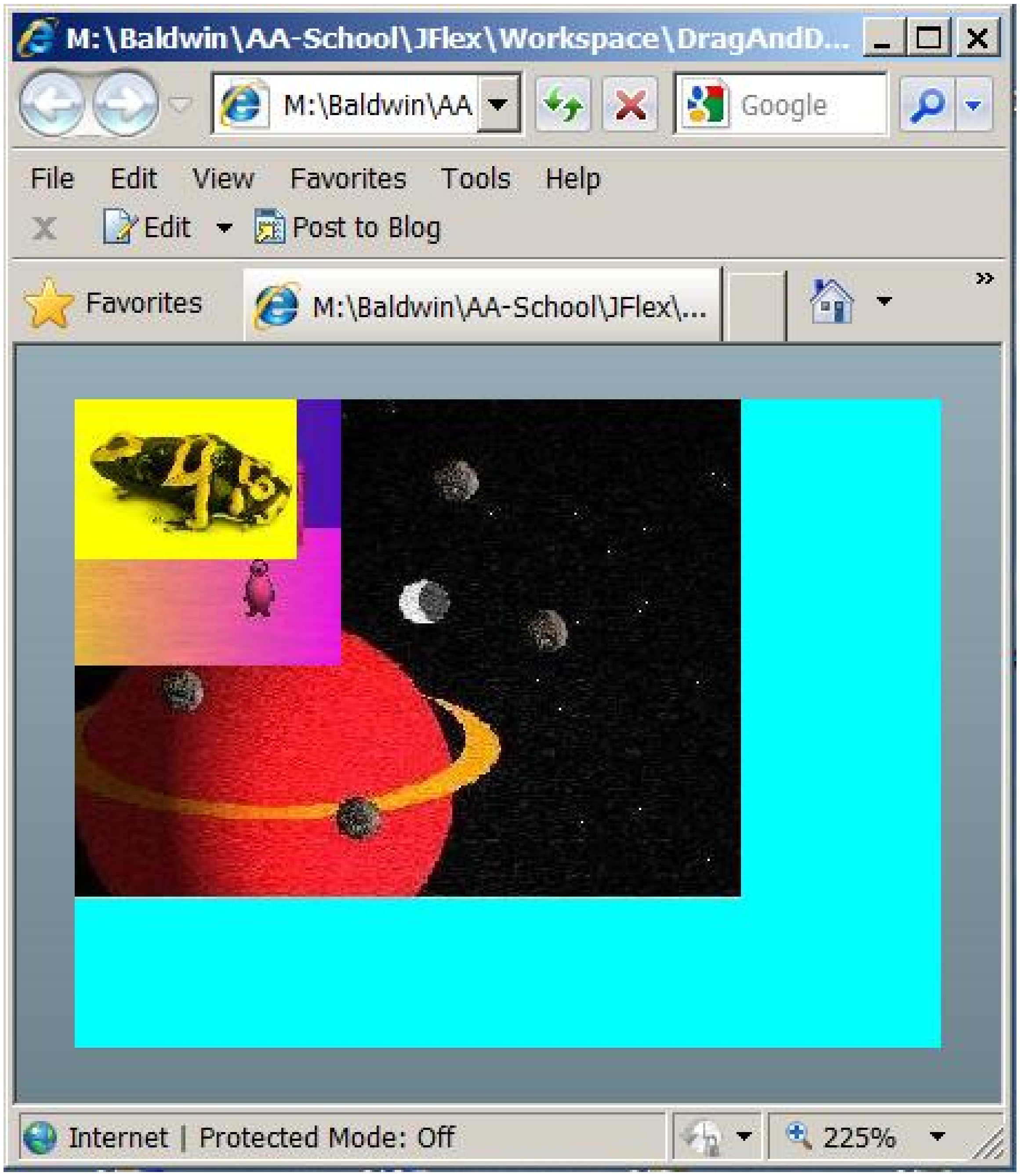
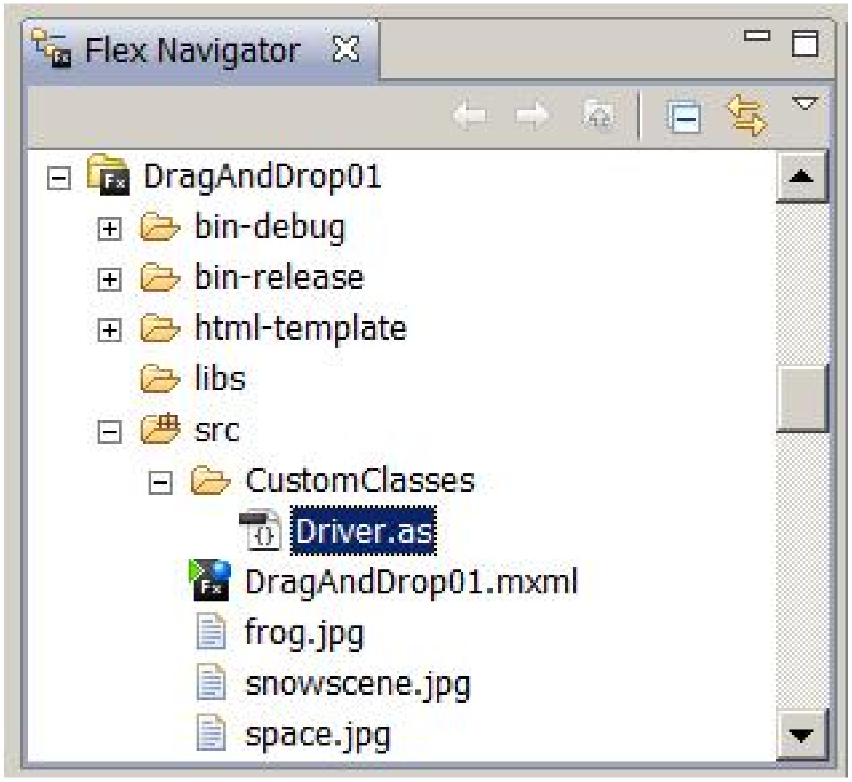

M:\Baldwin\AA-School\JFlex\Workspace\DragAndD...
M:\Baldwin\AA
Google
File Edit View Favorites Tools Help
Edit
Post to Blog
Favorites
M:\Baldwin\AA-School\JFlex\...
Internet | Protected Mode: Off
225%

Figure 1: Program output at startup.

The program file structure

The program file structure, taken from the Flex Builder 3 Navigator panel is as shown in Figure 2.

Program file structure.



```
DragAndDrop01
    bin-debug
    bin-release
    html-template
    libs
    src
        CustomClasses
            Driver.as
        DragAndDrop01.mxml
        frog.jpg
        snowscene.jpg
        space.jpg
```

Figure 2: Program file structure.

Three image files

As you can see in Figure 2, the program uses **the following image files** :

- 0 - space.jpg
- 1 - snowscene.jpg
- 2 - frog.jpg

The z-axes indices

The program sets the z-axis indices in the order shown in the above list (p. 6) on the **Image** objects produced using the image files. This causes the *space* image to be in the back (0), the *frog* image to be in the front (2), and the *snowscene* image to be in the middle (1).

The three images are initially placed in the upper-left corner of the canvas, which is shown as a cyan rectangle in Figure 1.

Any image can be dragged

If you run (p. 1) this program, you will see that any any of the images can be dragged and dropped anywhere within the canvas as long as the mouse pointer doesn't leave the canvas. However, if the edge of the dragged image goes outside the left edge or the top of the canvas, the drag and drop operation is aborted.

If the dragged image goes outside the right side or the bottom of the canvas, scroll bars automatically appear on the canvas as shown in Figure 3.

Program output after dragging the images

Figure 3 shows the program output after dragging the three images to different locations.

Program output after dragging the images.

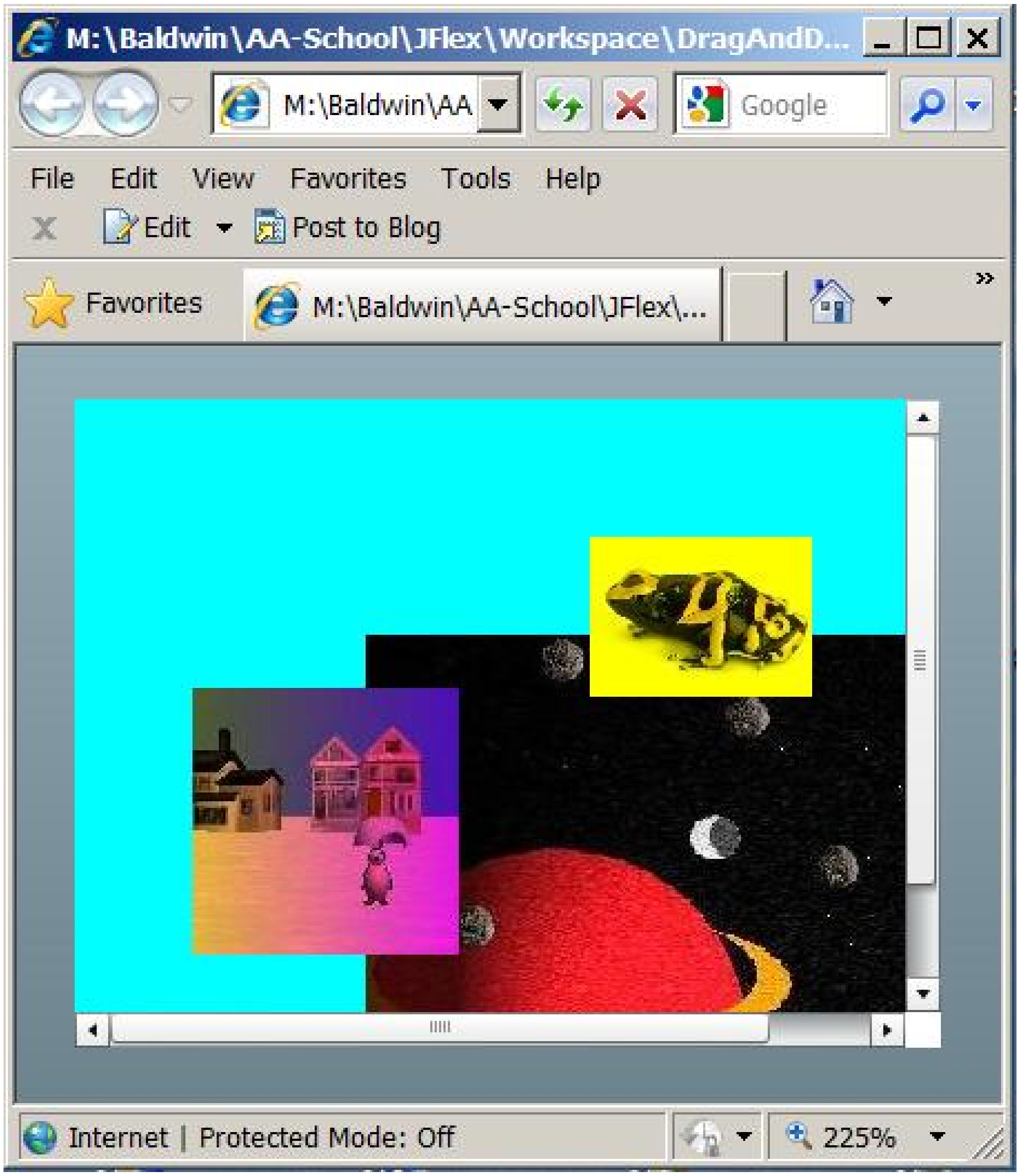

M:\Baldwin\AA-School\JFlex\Workspace\DragAndD...
M:\Baldwin\AA
Google
File
Edit
View
Favorites
Tools
Help
Edit
Post to Blog
Favorites
M:\Baldwin\AA-School\JFlex\...
Internet | Protected Mode: Off
225%

Figure 3: Program output after dragging the images.

5 Discussion and sample code

Will explain in fragments

I will explain the code for this program in fragments. Complete listings of the MXML code and the ActionScript code are provided in Listing 16 and Listing 17 near the end of the lesson.

5.1 The MXML file

The MXML file is shown in Listing 1 and also in Listing 16 for your convenience.

Listing 1: he MXML file for DragAndDrop01.

```
  <?xml version="1.0" encoding="utf-8"?>
<!--DragAndDrop01-->

<mx:Application
  xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:cc="CustomClasses.*">

  <cc:Driver/>

</mx:Application>
```

As you can see, the MXML file is very simple because the program was coded almost entirely in ActionScript. The MXML code simply instantiates an object of the **Driver** class. From that point forward, the behavior of the program is controlled by ActionScript code.

5.2 The ActionScript file

Beginning of the Driver class

The Driver class begins in Listing 2.

Listing 2: Beginning of the Driver class for DragAndDrop01.

```
    package CustomClasses{
  import flash.events.MouseEvent;

  import mx.containers.Canvas;
  import mx.controls.Image;
  import mx.core.DragSource;
  import mx.events.DragEvent;
  import mx.events.FlexEvent;
  import mx.managers.DragManager;

  //====================================================//

  public class Driver extends Canvas {
    private var imageA:Image = new Image();
    private var imageB:Image = new Image();
    private var imageC:Image = new Image();
    private var localX:Number;
    private var localY:Number;
```

Extends the Canvas class

As you can see in Listing 2, the **Driver** class extends the **Canvas** class. Therefore, an object of the **Driver** class is a **Canvas** object and has all of the attributes associated with a **Canvas** object. Among those attributes is the following, which was taken from the documentation [5] :

> *"A Canvas layout container defines a rectangular region in which you place child containers and controls. It is the only container that lets you explicitly specify the location of its children within the container by using the x and y properties of each child."*

As you will see, the new location of each image is explicitly specified each time it is dragged to a new location.

Instantiate three new Image objects

The code in Listing 2 instantiates three new **Image** objects, which will be loaded with the contents of the three image files listed earlier (p. 6) . The code in Listing 2 also declares two instance variables that will be used to store the position of the mouse pointer within an image when the drag operation is initiated.

Beginning of the constructor for the Driver class

The constructor for the **Driver** class begins in Listing 3.

Listing 3: Beginning of the constructor for the Driver class.

```
  public function Driver(){//constructor
setStyle("backgroundColor",0x00FFFF);
setStyle("backgroundAlpha",1.0);
```

Make the canvas visible

Normally a **Canvas** object is not visible. The code in Listing 3 sets the alpha value for the **Canvas** object to 1.0 making it opaque and visible. Listing 3 also sets the background color of the **Canvas** object to cyan as shown in Figure 1.

The size of the **Canvas** object will be set later when the **Canvas** object and all of its children have been constructed.

Prepare the three images

Listing 4 prepares the three images for use by the program.

Listing 4: Prepare the three images.

```
    //Embed the image files in the SWF file.
[Embed("snowscene.jpg")]
var imgA:Class;

[Embed("space.jpg")]
var imgB:Class;

[Embed("frog.jpg")]
var imgC:Class;

//Load the images from the embedded image files
// into the Image objects.
imageA.load(imgA);
imageB.load(imgB);
imageC.load(imgC);
```

[5] http://livedocs.adobe.com/flex/3/langref/mx/containers/Canvas.html

```
      // Set the z-axes indices such that the frog is
      // in front, the snowscene is in the middle and the
      // space image is at the back.
      addChildAt(imageB,0);//set index to 0
      addChildAt(imageA,1);//set index to 1
      addChildAt(imageC,2);//set index to 2
```

Listing 4 begins by embedding the three image files in the SWF file. Then it loads the contents of the image files into the **Image** objects instantiated in Listing 2. Finally Listing 3 adds the **Image** objects as children of the **Canvas** object.

Set the z-axis indices

The z-axis index of each **Image** object is set in Figure 4 so as to place the space image at the back, the frog image at the front, and the snowscene image between the other two.

Register a creationComplete event handler

Listing 5 registers a **creationComplete** event handler on the **Canvas** object. This event handler will be executed after the **Canvas** object and all of its children are fully constructed.

Listing 5: Register a creationComplete event handler.

```
        this.addEventListener(FlexEvent.CREATION_COMPLETE,
                                          completeHandler);
    } //end constructor
```

Beginning of the creationComplete event handler

The **creationComplete** event handler begins in Listing 6. This handler is executed once when the **Canvas** object and all of its children have been constructed.

Listing 6: Beginning of the creationComplete event handler.

```
      private function completeHandler(
                        event:mx.events.FlexEvent):void{
    //Set the width and height of the canvas based on
    // the dimensions of imageB.
    this.width = 1.3*imageB.width;
    this.height = 1.3*imageB.height;
```

Set the size of the Canvas object

Listing 6 sets the width and height of the **Canvas** object based on the dimensions of the **Image** object referred to by **imageB** . It was not possible to reliably execute this code in the constructor because the code might be executed before the contents of the image file were fully loaded into the **Image** object.

Register a mouseDown event handler on each Image object

A drag and drop operation is heavily dependent on the handling of different types of events. The remaining code in the **creationComplete** event handler registers appropriate event handlers on the images and on the **Canvas** object to support the drag and drop operation with the **Canvas** object as the drag target.

As you will see later, a drag operation is initialized when an image dispatches a **mouseDown** event. Listing 7 registers the same **mouseDown** event handler on all three **Image** objects.

Listing 7: Register a mouseDown event handler on each Image object.

```
    imageA.addEventListener(MouseEvent.MOUSE_DOWN,
                                mouseDownHandler);
  imageB.addEventListener(MouseEvent.MOUSE_DOWN,
                                mouseDownHandler);
  imageC.addEventListener(MouseEvent.MOUSE_DOWN,
                                mouseDownHandler);
```

Register dragDrop and dragEnter event handlers on the Canvas object

Two different event handlers must be registered on the drag target, which is the **Canvas** object in this case. The registration of those event handlers on the **Canvas** object is accomplished in Listing 8.

Listing 8: Register dragDrop and dragEnter event handlers on the Canvas object.

```
    this.addEventListener(DragEvent.DRAG_DROP,
                                     dropHandler);
  this.addEventListener(DragEvent.DRAG_ENTER,
                                enterHandler);
 } //end completeHandler
```

Beginning of the mouseDown event handler

The **mouseDown** event handler that was registered on the **Image** objects in Listing 7 begins in Listing 9. This event handler initiates the drag and drop operation on the **Image** object that dispatches the event.

Listing 9: Beginning of the mouseDown event handler.

```
   private function mouseDownHandler(
                           event:MouseEvent):void{

  //Save the location of the mouse within the image
  // being dragged. This information will be used
  // later to properly position the dropped image in
  // the drop target.
  this.localX = event.localX;
  this.localY = event.localY;
```

Positioning the dropped object

The easiest approach simply drops the image with its upper-left corner at the position of the mouse pointer when the mouse button is released. However, in my opinion, that is somewhat less than satisfactory from a visual viewpoint.

The drag proxy

When you drag an image, there is a default drag proxy that moves along with the mouse. *(It is possible to replace the default drag proxy with a drag proxy of your choice.)* The default drag proxy is a partially transparent rectangle that is the same size as the image.

Adjust the position of the upper-left corner

My preference is to manually adjust the drop location of the image based on the upper-left corner of the drag proxy and not based on the location of the mouse pointer. The code in Listing 9 gets and saves the coordinates of the mouse pointer within the image when the event is dispatched. As you will see later, I use these coordinates later to set the drop location on the basis of the upper-left corner of the proxy.

Get and save the drag initiator

The documentation refers to the object being dragged as the *drag initiator* .

Listing 10: Get and save the drag initiator.

```
        //Get the drag initiator component from the event
    // object and cast it to the correct type.
    var dragInitiator:Image = Image(
                                   event.currentTarget);
```

In this program, the drag initiator could be any of the three images shown in Figure 1 and Figure 3.
The code in Listing 10

- Gets a reference to the **Image** object that dispatched the **mouseDown** event from the incoming method parameter
- Casts it to type **Image** , and
- Saves it in the variable named **dragInitiator** .

Populate a DragSource object with a copy of the image being dragged
Here is part of what the documentation [6] has to say about the **DragSource** class.

"The DragSource class contains the data being dragged. The data can be in multiple formats, depending on the type of control that initiated the drag.

Each format of data is identified with a string. ... Data can be added directly using the addData() method, or indirectly using the addHandler() method."

Listing 11 adds the image being dragged to a new **DragSource** object and provides an identifier for the format as a string. You will see later how this string is used to establish the drop target.

Listing 11: Populate a DragSource object with a copy of the image being dragged.

```
        var dragSource:DragSource = new DragSource();

    dragSource.addData(dragInitiator,"imageObject");
```

Initiate the drag and drop operation by calling the doDrag method
Listing 12 initiates the drag and drop operation by calling the static **doDrag** method of the **DragManager** class.

Listing 12: Initiate the drag and drop operation by calling the doDrag method.

```
        DragManager.doDrag(dragInitiator,dragSource,event);

    }//end mouseDownHandler
```

What does the documentation have to say about the DragManager class?
Here is part of what the documentation [7] has to say about the **DragManager** class.

*"The **DragManager** class manages drag and drop operations, which let you move data from one place to another in a Flex application. For example, you can select an object, such as an item in a List control or a Flex control, such as an Image control, and then drag it over another component to add it to that component.*

[6] http://livedocs.adobe.com/flex/3/langref/mx/core/DragSource.html
[7] http://livedocs.adobe.com/flex/3/langref/mx/managers/DragManager.html

All methods and properties of the DragManager are static, so you do not need to create an instance of it. ...

When the user selects an item with the mouse, the selected component is called the drag initiator. The image displayed during the drag operation is called the drag proxy.

When the user moves the drag proxy over another component, the ***dragEnter*** *event is sent to that component. If the component accepts the drag, it becomes the drop target and receives dragOver, dragExit, and dragDrop events.*

When the drag is complete, a dragComplete event is sent to the drag initiator."

What about the doDrag method?

The documentation states simply that the **doDrag** method
"Initiates a drag and drop operation."

The doDrag method parameters

The **doDrag** method has several parameters with default values in addition to the three shown in Listing 12. Here is part of what the documentation has to say about the three parameters passed to the **doDrag** method in Listing 12.

- **dragInitiator :IUIComponent** - IUIComponent that specifies the component initiating the drag.
- **dragSource :DragSource** - DragSource object that contains the data being dragged.
- **event:MouseEvent** - The MouseEvent that contains the mouse information from the start of the drag.

The end of the mouseDown event handler

Listing 12 signals the end of the **mouseDown** event handler. This leaves two more event handlers to be discussed. The remaining two event handlers were registered on the drop target *(* ***Canvas*** *object)* by the code in Listing 8.

The dragEnter event handler

As you learned earlier (p. 14) , when the user moves the drag proxy over another component, that component dispatches a **dragEnter** event.

If a **dragEnter** event handler has been registered on that component, the handler method is executed. If the code in the event handler accepts the drag, it becomes the drop target and receives **dragOver** , **dragExit** , and **dragDrop** events.

In this case the intended drop target is the **Canvas** object and the event handler shown in Listing 13 is registered on that object.

Listing 13: The dragEnter event handler.

```
    private function enterHandler(event:DragEvent):void{
  if (event.dragSource.hasFormat("imageObject")){
      DragManager.acceptDragDrop(
                        Canvas(event.currentTarget));
  } //end if
} //end enterHandler
```

Confirm the correct format string

The code in Listing 13 checks to confirm that the format string in the **DragSource** object matches "imageObject" (see Listing 11). If so, it calls the static **acceptDragDrop** method on the **DragManager** class, passing a reference to itself as a parameter in the method call.

Accept the dragged object

The call to the **acceptDragDrop** method notifies the **DragManager** that the **Canvas** object is willing to accept the contents of the **DragSource** object being dropped onto itself.

Beginning of the dragDrop event handler

The **dragDrop** event handler was registered on the **Canvas** object in Listing 8. This method is executed after the **Canvas** object accepts the drag and the user releases the mouse button while the drag proxy is over the **Canvas** .

Correct the drop position for the image

The code in Listing 14 uses the current location of the mouse pointer along with the values stored in **localX** and **localY** to compute the new location for the upper-left corner of the image when it is dropped on the canvas.

> *(Recall that* ***localX*** *and* ***localY*** *contain the coordinates of the mouse pointer relative to the upper-left corner of the image when the* ***mouseDown*** *event was dispatched by the image and the drag and drop operation began.)*

I explained the need for this position adjustment earlier (p. 12) .

Listing 14: Beginning of the dragDrop event handler.

```
    private function dropHandler(event:DragEvent):void{

  var cornerX:Number = (Canvas(event.currentTarget).
                                  mouseX) - localX;
  var cornerY:Number = (Canvas(event.currentTarget).
                                  mouseY) - localY;
```

Do the drop

Listing 15 checks to confirm that the location at which the upper-left corner of the image will be placed is within the bounds of the canvas on the left side and the top. If so, it sets the coordinates of the **Image** object that dispatched the original **mouseDown** event to the coordinates that were computed in Listing 14. This causes that **Image** object to move to the new position on the **Canvas** object.

Listing 15: Do the drop.

```
      if((cornerX > 0.0) && (cornerY > 0.0)){
    Image(event.dragInitiator).x = cornerX;
    Image(event.dragInitiator).y = cornerY
  } //end if
} //end dropHandler
//--------------------------------------------------//

 } //end class
} //end package
```

The end of the program

Listing 15 also signals the end of the **dragDrop** event handler, the end of the **Driver** class, and the end of the program.

6 Run the program

I encourage you to run (p. 1) this program from the web. Then copy the code from Listing 16 and Listing 17. Use that code to create a Flex project. Compile and run the project. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

7 Resources

I will publish a list containing links to ActionScript resources as a separate document. Search for ActionScript Resources in the Connexions search box.

8 Complete program listings

Complete listings for the MXML and ActionScript code discussed in this lesson are provided in Listing 16 and Listing 17 below.

Listing 16: The MXML file for DragAndDrop01.

```
<?xml version="1.0" encoding="utf-8"?>
<!--DragAndDrop01
-->

<mx:Application
  xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:cc="CustomClasses.*">

  <cc:Driver/>

</mx:Application>
```

Listing 17: The ActionScript file for DragAndDrop01.

```
/*DragAndDrop01
Illustrates the fundamentals of drag and drop in
ActionScript 3.

Places three images in a Canvas object:

0 - space.jpg - largest
1 - snowscene.jpg - midsize
2 - frog.jpg - smallest

Sets the z-axis indices as shown above. This causes the
space image to be in the back, the frog image to be in the
front, and the snowscene image to be in the middle.

Any of the images can be dragged and dropped anywhere
within the canvas so long as the mouse pointer doesn't
leave the canvas. However, If the edge of the dragged
```

```
image goes outside the left edge or the top of the canvas,
the drag and drop operation is aborted. If the dragged
image goes outside the right side or the bottom of the
canvas, scroll bars automatically appear on the canvas.

The size of the canvas is based on the size of the space
image so that other images can be substituted for mine
when the program is recompiled so long as the file names
and paths are the same.
*****************************************************/
package CustomClasses{
  import flash.events.MouseEvent;

  import mx.containers.Canvas;
  import mx.controls.Image;
  import mx.core.DragSource;
  import mx.events.DragEvent;
  import mx.events.FlexEvent;
  import mx.managers.DragManager;

  //====================================================//

  public class Driver extends Canvas {
    private var imageA:Image = new Image();
    private var imageB:Image = new Image();
    private var imageC:Image = new Image();
    private var localX:Number;
    private var localY:Number;

    public function Driver(){//constructor
      //Make the Canvas visible.
      setStyle("backgroundColor",0x00FFFF);
      setStyle("backgroundAlpha",1.0);

      //Embed the image files in the SWF file.
      [Embed("snowscene.jpg")]
      var imgA:Class;

      [Embed("space.jpg")]
      var imgB:Class;

      [Embed("frog.jpg")]
      var imgC:Class;

      //Load the images from the embedded image files
      // into the Image objects.
      imageA.load(imgA);
      imageB.load(imgB);
      imageC.load(imgC);

      // Set the z-axes indices such that the frog is
      // in front, the snowscene is in the middle and the
      // space image is at the back.
      addChildAt(imageB,0);//set index to 0
      addChildAt(imageA,1);//set index to 1
      addChildAt(imageC,2);//set index to 2

      //Register an event handler that will be executed
      // whcn the canvas and its children are fully
      // constructed.
      this.addEventListener(FlexEvent.CREATION_COMPLETE,
                                        completeHandler);
    } //end constructor
    //--------------------------------------------------//

    //This handler method is executed when the Canvas and
    // its children have been fully created.
    private function completeHandler(
                      event:mx.events.FlexEvent):void{
      //Set the width and height of the canvas based on
      // the dimensions of imageB.
      this.width = 1.3*imageB.width;
      this.height = 1.3*imageB.height;

      //Register event listeners to support drag and drop
      // operations on all three images with the canvas
      // as the drag target.
      imageA.addEventListener(MouseEvent.MOUSE_DOWN,
                                       mouseDownHandler);
      imageB.addEventListener(MouseEvent.MOUSE_DOWN,
                                       mouseDownHandler);
      imageC.addEventListener(MouseEvent.MOUSE_DOWN,
                                       mouseDownHandler);

      this.addEventListener(DragEvent.DRAG_DROP,
                                            dropHandler);
      this.addEventListener(DragEvent.DRAG_ENTER,
                                       enterHandler);
    } //end completeHandler
    //--------------------------------------------------//

    // This event handler initiates the drag-and-drop \
    // operation for the image that dispatches the
    // mouseDown event.
    private function mouseDownHandler(
                               event:MouseEvent):void{

      //Save the location of the mouse within the image
      // being dragged. This information will be used
      // later to properly position the dropped image in
      // the drop target.
      this.localX = event.localX;
      this.localY = event.localY;

      //Get the drag initiator component from the event
      // object and cast it to the correct type.
      var dragInitiator:Image = Image(
                                      event.currentTarget);

      //Add the image being dragged to a DragSource
      // object and define an identifier as a string.
      var dragSource:DragSource = new DragSource();
      dragSource.addData(dragInitiator,"imageObject");

      //Call the static doDrag method on the DragManager
      // class to manage the overall drag and drop
      // operation.
      DragManager.doDrag(dragInitiator,dragSource,event);
    }//end mouseDownHandler
    //----------------------------------------------------//

    //This dragEnter event handler causes the canvas to
    // be a suitable drop target.
    private function enterHandler(event:DragEvent):void{
      if (event.dragSource.hasFormat("imageObject")){
          DragManager.acceptDragDrop(
                                Canvas(event.currentTarget));
      } //end if
    } //end enterHandler
    //----------------------------------------------------//

    //Execute the dragDrop event handler to drop the image
    // in its new location. Compensate for the fact that
    // the mouse pointer is not at the upper-left corner
    // of the image. Also don't allow the image to be
    // dragged off the left side of the canvas or off the
    // top of the canvas.
    private function dropHandler(event:DragEvent):void{

      //Compute the position of the upper-left corner of
      // the dropped image.
      var cornerX:Number = (Canvas(event.currentTarget).
                                          mouseX) - localX;
      var cornerY:Number = (Canvas(event.currentTarget).
                                          mouseY) - localY;
      if((cornerX > 0.0) && (cornerY > 0.0)){
        Image(event.dragInitiator).x = cornerX;
        Image(event.dragInitiator).y = cornerY
      } //end if
    } //end dropHandler
    //----------------------------------------------------//

  } //end class
```

```
} //end package
```

9 Miscellaneous

This section contains a variety of miscellaneous materials.

NOTE: **Housekeeping material**

- Module name: Drag and Drop Basics
- Files:
  - ActionScript0140\ActionScript0140.htm
  - ActionScript0140\Connexions\ActionScriptXhtml0140.htm

NOTE: **PDF disclaimer:** Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

-end-