


## Summary of Comments on Adobe Flex & AIR ACE Study Guide

Page: 33

Author: Jonnie    Subject: Sticky Note    Date: 2/22/09 4:37:58 PM -08'00'  
Need to complete

Add	Specifies to play the effect on all children added during the change of view state.
Hide	Specifies to play the effect on all children whose visible property changes from true to false during the change of view state.
Move	Specifies to play the effect on all children whose x or y properties change during the change of view state.
Remove	Specifies to play the effect on all children removed during the change of view state.
Resize	Specifies to play the effect on all children whose width or height properties change during the change of view state.
Show	Specifies to play the effect on all children whose visible property changes from false to true during the change of view state.

 Using Effects

### 1.5 - Position UI elements by using constraint-based layout.

#### a) Absolute positioning

Three containers support `absolute` positioning:


Application and Panel controls use `absolute` positioning if you specify the layout property as "`absolute`" (`ContainerLayout.ABSOLUTE`).

The Canvas container always uses `absolute` positioning.

With absolute positioning, you specify the child control position by using its `x` and `y` properties, or you specify a constraint-based layout; otherwise, Flex places the child at position 0, 0 of the parent container.

When you specify the `x` and `y` coordinates, Flex repositions the controls only when you change the property values.

When you use absolute positioning, you have full control over the locations of the container's children. This lets you overlap components.

1  Position UI elements by using enhanced constraints.

## 1.7 - Implement application navigation by using navigator containers.

### a. ViewStack

- `selectedIndex`: The index of the currently active container if one or more child containers are defined. The value of this property is -1 if no child containers are defined.  
  
The index is numbered from 0 to `numChildren - 1`, where `numChildren` is the number of child containers in the `ViewStack` container. Set this property to the index of the container that you want active.  
  
You can use the `selectedIndex` property of the `<mx:ViewStack>` tag to set the default active container when your application starts.
  
- `selectedChild`: The currently active container if one or more child containers are defined. The value of this property is null if no child containers are defined.  
  
Set this property in ActionScript to the identifier of the container that you want active  
  
You can set this property only in an ActionScript statement, not in MXML.  
  
The following example uses ActionScript to set the `selectedChild` property so that the active child container is the child container with an identifier of search:
  
- `numChildren`: Contains the number of child containers in the `ViewStack` container.  
  
The following uses the `numChildren` property in an ActionScript statement to set the active child container to the last container in the stack:  
  

```
myViewStack.selectedIndex = myViewStack.numChildren - 1;
```

You use the `dispatchEvent()` method to dispatch an event. The `dispatchEvent()` method has the following signature:

```
public dispatchEvent(event:Event):Boolean dispatchEvent(new Event("click"));
```

This method requires an argument of the `Event` type, which is the event object. The `dispatchEvent()` method initializes the `target` property of the event object with a reference to the component dispatching the event.

You can create an event object and dispatch the event in a single statement, as the following example shows:

```
dispatchEvent(new Event("click"));
```

You can also create an event object, initialize it, and then dispatch it, as the following example shows:

```
var eventObj:EnableChangeEvent = new EnableChangeEvent("enableChange");
    eventObj.isEnabled = true;
dispatchEvent(eventObj);
```

#### a. Creating static constants for the `Event.type` property

The constructor of an event class typically takes a single required argument that specifies the value of the event object's type property. In the previous section, you passed the string `enableChange` to the constructor, as the following example shows:

```
// Define event object, initialize it, then dispatch it.
var eventObj:EnableChangeEvent = new EnableChangeEvent("enableChange");
    dispatchEvent(eventObj);
```

## 2.6 Handle framework events.

### 2.7 List and describe the differences between model, view, and controller code in a Flex application.

MVC is a design pattern or software architecture that separates the applications data, user interface, and control logic into three distinct groupings. The goal is to implement the logic so changes can be made to one portion of the application with a minimal impact to the others. Short definitions of the key terms are as follows:

- **Model:** The data the application uses. It manages data elements, responds to queries about its state, and instructions to change the data.
- **View:** The user interface. It is responsible for presenting model data to the user and gathering information from the user.
- **Controller:** Responds to events, typically user events, but also system events. The events are interpreted and the controller invokes changes on the model and view.

## Page: 41

Author: jonniespratley	Subject: Sticky Note	Date: 1/14/09 2:57:50 AM -08'00'
Question on exam, asks how and what do you do to dispatch a custom event.		
Author: jonniespratley	Subject: Highlight	Date: 1/14/09 2:56:49 AM -08'00'
Author: jonniespratley	Subject: Highlight	Date: 1/14/09 2:56:54 AM -08'00'
Author: jonniespratley	Subject: Highlight	Date: 1/14/09 2:56:57 AM -08'00'

### c. Class attributes

ActionScript 3.0 allows you to modify class definitions using one of the following four attributes:

Attribute	Definition
dynamic	Allow properties to be added to instances at run time.
final	Must not be extended by another class.
internal ( default )	Visible to references inside the current package
public	Visible to references everywhere

Page: 43

Author: jonniespratley Subject: Highlight Date: 1/14/09 2:59:40 AM -08'00'  
on exam, know these attributes

Author: jonniespratley Subject: Highlight Date: 1/14/09 2:59:21 AM -08'00'

For each of these attributes, except for internal, you must explicitly include the attribute to get the associated behavior.

For example, if you do not include the dynamic attribute when defining a class, you will not be able to add properties to a class instance at run time.

You explicitly assign an attribute by placing it at the beginning of the class definition, as the following code demonstrates:

```
dynamic class Shape
{
    //...
}
```

Notice that the list does not include an attribute named `abstract`. This is because abstract classes are not supported in ActionScript 3.0.

Notice also that the list does not include attributes named `private` and `protected`. These attributes have meaning only inside a class definition, and cannot be applied to classes themselves. If you do not want a class to be publicly visible outside a package, place the class inside a package and mark the class with the internal attribute.


Alternatively, you can omit both the internal and public attributes, and the compiler will automatically add the internal attribute for you. If you do not want a class to be visible outside the source file in which it is defined, place the class at the bottom of your source file, below the closing curly brace of the package definition.

### d. Class body

The class body, which is enclosed by curly braces, is used to define the variables, constants, and methods of your class. The following example shows the declaration for the Accessibility class in the Adobe Flash Player API:

```
public final class Accessibility
{
    public static function get active():Boolean;
    public static function updateProperties():void;
}
```

Although the number of parameters and the data type of each parameter in the implemented method must match that of the interface method, the parameter names do not need to match.

 Use access modifiers with classes and class members.

### 3.4 Under the purpose of and implement data transfer objects.

#### a. Using class-based models

As with MXML-based models, you cannot type the properties of a script-based model. To type properties, you must use a class-based model.

Using an ActionScript class as a model is a good option when you have to store complex data structures with typed properties, or when you want to execute client-side business logic by using application data. Also, the type information in a class-based model is retained on the server when the model is passed to a server-side data service.

The following example shows the employee model defined in an ActionScript class called `EmployeeModel`. The `EmployeeModel` model class contains various typed properties as well as accessor methods (getter/setters), a simple custom validation method, and a method that returns a Transfer Object.

*Note: A Transfer Object (also known as Value Object) is a simple object that contains just the data required to carry out a certain unit of business logic. In the following example, the Employee transfer object (EmployeeTO) contains only typed properties that describe an employee.*

It does not have methods like the model class. Transfer Objects are useful when moving data between the tiers by using technologies such as Flash Remoting (the transportation mechanism that is also used in Flex Data Services) and JSON that automatically serialize and deserialize complex objects across tiers while maintaining type information.

```
package
{
    public class EmployeeTO
    {
        public var firstName:String;
        public var lastName:String;
    }
}
```

```
    public var department:uint;  
    public var email:String;  
}  
}
```

### 3.5 Implement accessor methods in ActionScript.

### 3.6 Use an ArrayCollection to sort, filter, and provide data.

#### a. Using an ArrayCollection

A data collection is an ordered list of data objects stored in client application memory. Flex provides an ActionScript class named `ArrayCollection` that's designed for this purpose.

More than a simple `Array`, the `ArrayCollection` class has these advantages:

Reliably executes binding expressions that refer to its stored data.

Implements a set of interfaces that provide client-side data filtering, sorting, bookmarking, and traversal.

An `ArrayCollection` can be serialized for transport over the Web in requests to Web services, remoting services, and messaging services.

#### a) Declaring an ArrayCollection

To declare an `ArrayCollection` in MXML, use the `<mx:ArrayCollection/>` tag and assign it an `id` property.

To declare an `ArrayCollection` in ActionScript, you declare and instantiate the `ArrayCollection` variable:

```
import mx.collections.ArrayCollection;  
  
private var myData:ArrayCollection;
```

#### b) Setting an ArrayCollection object's source property

The `ArrayCollection` class has a `source` property that refers to a raw `Array` containing its data. You can set an

`ArrayCollection` object's source in a number of ways:

- By passing the `Array` into the `ArrayCollection` object's constructor method.  
`myData = new ArrayCollection( myArray );`

With an ActionScript statement after the `ArrayCollection` has been instantiated:

```
myData.source = ['red', 'green', 'blue'];
```

In an MXML declaration, nested in `<mx:source>` tags:

```
<mx:ArrayCollection id="myData">
  <mx:source>
    <mx:String>Red</mx:String>
    <mx:String>Blue</mx:String>
    <mx:String>Green</mx:String>
  </mx:source>
</mx:ArrayCollection>
```

### c) Accessing data at runtime

After an `ArrayCollection` object has been created, you can dynamically get, add, and remove data at runtime with the `ArrayCollection` class interface.

The following `ArrayCollection` methods and properties are designed for this purpose:

- `addItem( item:Object )` appends a data item to the end of the collection.
- `addItemAt( item:Object, index:int )` adds a data item in the collection at the declared index position. Existing data items are shifted downward to make room for the new data item.
- `getItemAt( index:int, prefetch:int = 0 )` returns a data item at the declared index position. The optional `prefetch` argument is used when an `ArrayCollection` contains managed data to indicate how many rows of data should be fetched from the server.
- `removeItemAt( index:int )` removes a data item from the `ArrayCollection` object
- `removeAll()` clears all items from the collection.
- `setItemAt( item:Object, index:int )` replaces a data item in the declared index position
- `length:int` returns the number of items in the `ArrayCollection`

### d) Managing data at runtime

The `ArrayCollection` class implements a number of interfaces to allow you to dynamically manage data in client application memory at runtime. These interfaces include:

- `ICollectionView`, with methods for filtering and sorting data at runtime
- `IList`, with the methods described previously for adding, removing, and accessing data at runtime

### e) Filtering Data

The `ArrayCollection` class executes filtering through its `filterFunction` property. This property is designed to reference an ActionScript function that you create and customize. A function designed for filtering always has this signature:

```
private function functionName( item:Object ):Boolean
```

The `item` argument can be either a generic ActionScript Object variable or a strongly typed value object. When you execute a filter, the `ArrayCollection` class loops through its source data and executes the filter function once for each data item.

If the filtering function returns `true`, the current data item is included in the resulting filtered view; if it returns `false`, the data item is hidden and won't be visible to the user unless and until the filter is removed.

The following filtering function examines a property of a data item and compares it to a value provided by the user through a visual component.

If the data item property and the user-provided value match, the function returns `true`, indicating that the data item should be included in the filtered view:

```
private function filterTables( item:Object ):Boolean
{
    return ( item.id == dg_tableData.selectedItem.id );
}
```

To use the filter function, first assign the function to the `ArrayCollection` class's filter function property by its name.

Then call the `ArrayCollection` object's `refresh()` method to cause the filtering to happen:

```
tableDataCollection.filterFunction = filterTables;
tableDataCollection.refresh();
```

## f) Sorting Data

The `ArrayCollection` sorts data through use of its `sort` property. The `sort` property references an instance of the `mx.collections.Sort` class.

This class in turn has `field`'s property that references an Array containing instances of `mx.collections.SortField`.

The `SortField` class supports the `Boolean` properties that determine which named property of an `ArrayCollection` object's data items to sort on and how to execute the sort operation:

- `caseInsensitive` defaults to `false`, meaning that sort operations are case-sensitive by default.
- `descending` defaults to `false`, meaning that sort operations are ascending by default
- `numeric` defaults to `false`, meaning that sort operations are text-based by default

You can instantiate a `SortField` object and set all of its `Boolean` properties in the constructor method call, using this syntax

```
var mySortField:SortField = new SortField( 'propName', caseInsensitive, descending, numeric );
```

```
<firstname>Jonnie</firstname>
<lastname>Spratley</lastname>
</customer>
<customer>
  <firstname>Foo</firstname>
  <lastname>Bar</lastname>
</customer>
```

#### d) Filtering XML data with predicate expressions

A predicate expression lets you filter data in an XML object, accomplishing tasks similar to the WHERE clause in an SQL statement. The predicate expression itself is an ActionScript comparison expression wrapped in parentheses. You appended the predicate to the part of the E4X expression that indicates what data you want to return, separated with a dot operator.

This ActionScript extracts all <customer> elements that have a <lastname> matching the String value Jones:

```
xRetrun = xInvoices..customer.(lastname=='Jones')
```

The resulting XML node looks like this:

```
<customer>
  <firstname>Foo</firstname>
  <lastname>Jones</lastname>
</customer>
```

## 4. Interacting with data sources and servers

### 4.1 Implement simple LiveCycle Data Services (LCDS) messaging and data management.

#### a. About LiveCycle Data Services

LiveCycle Data Services provides a messaging infrastructure for building data-rich Flex applications. This infrastructure underlies the LiveCycle Data Services features that are designed for moving data to and from applications: RPC services, the Data Management Service, and the Message Service.

The following table describes the types of services provided in LiveCycle Data Services:

Service	Description
RPC services	Provides a call and response model for accessing external data. Lets you create applications that make asynchronous requests to remote services that process the requests, and then return data to your Flex application. For more information, see About RPC services.

#### f) Creating a Producer component in ActionScript

You can create a Producer component in an ActionScript method. The following code shows a Producer component that is created in a method in an `<mx:Script>` tag. The import statements import the classes required to create a Producer component, create Message objects, add event listeners, and create message handlers.

```
<mx:Script>
  <![CDATA[
    import mx.messaging.*;
    import mx.messaging.messages.*;
    import mx.messaging.events.*;
    private var producer:Producer;
    private function acknowledgeHandler(event:MessageAckEvent):void{
      // Handle message event.
    }
    private function faultHandler(event:MessageFaultEvent):void{
      // Handle fault event.
    }
    private function logon():void {
      producer = new Producer();
      producer.destination = "chatTopicJMS";
      producer.addEventListener(MessageAckEvent.ACKNOWLEDGE, acknowledgeHandler);
      producer.addEventListener(MessageFaultEvent.FAULT, faultHandler);
    }
  ]]>
</mx:Script>
```

Page: 68

Author: jonniespratley Subject: Highlight Date: 1/14/09 3:01:16 AM -08'00'  
On exam, what do you use to send a message to Icds or blazed.

#### g) Sending a message to a destination

To send a message from a Producer component to a destination, you create an `mx.messaging.messages.AsyncMessage` object, populate the body of the `AsyncMessage` object, and then call the component's `send()` method. You can create text messages and messages that contain objects.

The following code creates a message, populates the body of the message with text, and sends the message by calling a Producer component's `send()` method:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
    <![CDATA[
      import mx.messaging.*;
      import mx.messaging.messages.*;
      import mx.messaging.events.*;
      private var producer:Producer;

      private function sendMessage():void {
        var message:AsyncMessage = new AsyncMessage();
        message.body = userName.text + " " + input.text;
        producer.send(message);
      }
    ]]>
  </mx:Script>
  <mx:TextInput id="userName"/>
  <mx:TextInput id="input"/>
  <mx:Button label="Send" click="sendMessage();"/>
</mx:Application>
```

## a. Creating a database

To create a new database file, you first create an `SQLConnection` instance. You call its `open()` method to open it in synchronous execution mode, or its `openAsync()` method to open it in asynchronous execution mode.

The `open()` and `openAsync()` methods are used to open a connection to a database. If you pass a `File` instance that refers to a non-existent file location for the reference parameter (the first parameter), the `open()` or `openAsync()` method will create a database file at that file location and open a connection to the newly created database.

Whether you call the `open()` method or the `openAsync()` method to create a new database, the database file's name can be any valid file name, with any file extension. If you call the `open()` or `openAsync()` method with null for the reference parameter, a new in-memory database is created rather than a database file on disk.

The following code listing shows the process of creating a database file (a new database) using asynchronous execution mode. In this case, the database file is saved in the application's storage directory, with the file name "DBSample.db".

```
import flash.data.SQLConnection;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();
conn.addEventListener(SQLEvent.OPEN, openHandler);
conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);

var dbFile:File = File.applicationStorageDirectory.resolvePath("DBSample.db");

conn.openAsync(dbFile);

function openHandler(event:SQLEvent):void
{
    trace("the database was created successfully");
}

function errorHandler(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

To execute operations synchronously, when you open a database connection with the `SQLConnection` instance, call the `open()` method.

The following example shows how to create and open an `SQLConnection` instance that executes its operations synchronously:

```
import flash.data.SQLConnection;
import flash.events.SQLErrorEvent;
```

```
}  
  
public function faultHandler (event:FaultEvent):void  
{  
    // Deal with event.fault.faultString, etc.  
}
```

#### 4.6 Upload files to a server.

##### a. The following is working with file upload.

###### a) Working with file upload

The `FileReference` class lets you add the ability to upload and download files between a client and a server. Users are prompted to select a file to upload or a location for download from a dialog box (such as the Open dialog box on the Windows operating system).

Each `FileReference` object that you create with ActionScript refers to a single file on the user's hard disk. The object has properties that contain information about the file's size, type, name, creation date, and modification date.

Note: The `creator` property is supported on Mac OS only. All other platforms return null.

You can create an instance of the `FileReference` class in two ways. You can use the new operator, as the following code shows:

```
import flash.net.FileReference;  
  
private var myFileReference:FileReference = new FileReference();
```

Or you can call the `FileReferenceList.browse()` method, which opens a dialog box on the user's system to prompt the user to select one or more files to upload and then creates an array of `FileReference` objects if the user selects one or more files successfully.

Each `FileReference` object represents a file selected by the user from the dialog box. A `FileReference` object does not contain any data in the `FileReference` properties (such as `name`, `size`, or `modificationDate`) until one of the following happens:

The `FileReference.browse()` method or `FileReferenceList.browse()` method has been called, and the user has selected a file from the file picker.

The `FileReference.download()` method has been called, and the user has selected a file from the file picker.

Note: When performing a download, only the `FileReference.name` property is populated before the download is complete. After the file has been downloaded, all properties are available.

You can send data to the server with the `FileReference.upload()` method by using the `URLRequest` method and `URLRequest.data` properties to send variables using the POST or GET methods.

When you attempt to upload a file using the `FileReference.upload()` method, any of the following events may be dispatched:

`Event.OPEN`: Dispatched when an upload operation starts.

`ProgressEvent.PROGRESS`: Dispatched periodically during the file upload operation.

`Event.COMPLETE`: Dispatched when the file upload operation completes successfully.

`SecurityErrorEvent.SECURITY_ERROR`: Dispatched when an upload fails because of a security violation.

`HTTPStatusEvent.HTTP_STATUS`: Dispatched when an upload fails because of an HTTP error.

`IOErrorEvent.IO_ERROR`: Dispatched if the upload fails because of any of the following reasons:

An input/output error occurred while Flash Player is reading, writing, or transmitting the file.

The SWF tried to upload a file to a server that requires authentication (such as a user name and password). During upload, Flash Player does not provide a means for users to enter passwords.

The `url` parameter contains an invalid protocol. The `FileReference.upload()` method must use either HTTP or HTTPS.

You can pass additional variables to the upload script using either the POST or GET request method. To send additional POST variables to your upload script, you can use the following code:

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
fileRef.browse();
private function selectHandler(event:Event):void
{
    var params:URLVariables = new URLVariables();
    params.date = new Date();
    params.ssid = "94103-1394-2345";
    var request:URLRequest = new URLRequest("http://yourdomain.com/fileupload.cfm");
    request.method = URLRequestMethod.POST;
    request.data = params;

    fileRef.upload(request, "Custom1");
}

private function completeHandler(event:Event):void
{
    trace("uploaded");
}
```

Property	Description
nativePath	<p>Specifies the platform-specific path to a file.</p> <p>For example:</p> <p>Windows a path might be "c:\Sample directory\test.txt"</p> <p>On Mac OS it could be "/Sample directory/test.txt"</p> <p>A nativePath property uses the backslash (\) character as the directory separator character on Windows, and it uses the forward slash (/) character on Mac OS.</p>
url	<p>This may use the file URL scheme to point to a file.</p> <p>For example:</p> <p>Windows a path might be "file:///c:/Sample%20directory/test.txt"</p> <p>Mac OS it could be "file:///Sample%20directory/test.txt"</p>

b) Pointing a File object to a directory

There are different ways to set a File object to point to a directory.

c) Pointing to the user's home directory

You can point a File object to the user's home directory.

On Windows, the home directory is the parent of the "My Documents" directory

```
"C:\Documents and Settings\userName\My Documents"
```

On Mac OS, it is the Users/userName directory.

The following code sets a File object to point to an AIR Test subdirectory of the home directory:

```
var file:File = File.userDirectory.resolvePath("AIR Test");
```

d) Pointing to the user's documents directory

You can point a File object to the user's documents directory.

On Windows, the "My Documents" directory

```
"C:\Documents and Settings\userName\My Documents"
```

The AIR window types combine chrome and visibility attributes of the native operating system to create three functional types of window. Use the constants defined in the `NativeWindowType` class to reference the type names in code. AIR provides the following window types:

Type	Description
normal	A typical window. Normal windows use the full-size style of chrome and appear on the Windows task bar and the Mac OS X window menu.
utility	A tool palette. Utility windows use a slimmer version of the system chrome and do not appear on the Windows task bar and the Mac OS-X window menu.
lightweight	Lightweight windows have no chrome and do not appear on the Windows task bar or the Mac OS X window menu.  In addition, lightweight windows do not have the System (Alt+Space) menu on Windows.  Lightweight windows are suitable for notification bubbles and controls such as combo-boxes that open a short-lived display area.  When the lightweight type is used, <code>systemChrome</code> must be set to <code>none</code> .

**b) Window chrome**

Window chrome is the set of controls that allow users to manipulate a window in the desktop environment.

Chrome elements include the title bar, title bar buttons, border, and resize grippers.

**c) System chrome**

You can set the `systemChrome` property to `standard` or `none`.

Choose `standard` system chrome to give your window the set of standard controls created and styled by the user's operating system.

Choose `none` to provide your own chrome for the window (or to use Flex chrome). Use the constants defined in the `NativeWindowSystemChrome` class to reference the system chrome settings in code.

System chrome is managed by the system. Your application has no direct access to the controls themselves, but can react to the

events dispatched when the controls are used. When you use standard chrome for a window, the `transparent` property must be set to `false` and the `type` property must be `normal` or `utility`.

#### d) Flex chrome

When you use the Flex `mx:WindowedApplication` or `mx:Window` components, the window can use either system chrome or chrome provided by the Flex framework. To use the Flex chrome, set the `systemChrome` property used to create the window to `none`.

#### e) Custom chrome

When you create a window with no system chrome and you do not use the Flex `mx:WindowedApplication` or `mx:Window` components, then you must add your own chrome controls to handle the interactions between a user and the window.

You are also free to make transparent, non-rectangular windows.

#### f) Window transparency

To allow alpha blending of a window with the desktop or other windows, set the `windowTransparent` property to `true`.

The `transparent` property must be set before the window is created and cannot be changed.

A transparent window has no default background. Any window area not occupied by a display object will be invisible.

If a display object has an alpha setting of less than one, then anything below the object will show through, including other display objects in the same window, other windows, and the desktop.

Rendering large alpha-blended areas can be slow, so the effect should be used conservatively.

Transparent windows are useful when you want to create applications with borders that are irregular in shape or that "fade out" or appear to be invisible.

Transparency cannot be used with windows that have system chrome.

#### c. Creating windows

AIR automatically creates the first window for an application, but you can create any additional windows you need.

To create a native window, use the `NativeWindow` constructor method.

To create a Flex window, use the `mx:Window` class.

To create an HTML window, either use the `HTMLLoader.createRootWindow()` method or, from an HTML document, call the `JavaScriptWindow.open()` method.

## j) Creating a menu separator line

To create a separator line, create a `NativeMenuItem`, setting the `isSeparator` parameter to true in the constructor. Then add the separator item to the menu in the correct location:

```
var separatorA:NativeMenuItem = new NativeMenuItem("A", true);
    editMenu.addItem(separatorA);
```

Note: that the label specified for the separator, if any, will not be displayed.

## 5.4 - Adding drag-and-drop functionality to and from the desktop.

## a. Drag-and-drop gesture stages

The drag-and-drop gesture has three stages:

- **Initiation:** A user initiates a drag-and-drop operation by dragging from a component, or an item in a component, while holding down the mouse button.

The component that is the source of the dragged item is typically designated as the **drag initiator and dispatches `nativeDragStart` and `nativeDragComplete` events.**

An Adobe AIR application starts a drag operation by calling the `NativeDragManager.doDrag()` method in response to a `mouseDown` or `mouseMove` event.

- **Dragging:** While holding down the mouse button, the user moves the mouse cursor to another component, application, or to the desktop. AIR optionally displays a proxy image during the drag. As long as the drag is underway, the initiator **object dispatches `nativeDragUpdate` events.**

When the user moves the mouse over a possible drop target in an AIR application, the drop target dispatches a **`nativeDragEnter` event.**

The event handler can inspect the event object to determine whether the dragged data is available in a format that the target accepts and, if so, let the user drop the data onto it by calling the `NativeDragManager.acceptDragDrop()` method.

As long as the drag gesture remains over an interactive object, that **object dispatches `nativeDragOver` events.** When the drag gesture leaves the interactive object, it **dispatches a `nativeDragExit` event.**

- **Drop:** The user releases the mouse over an eligible drop target. If the target is an AIR application or component, then the **component dispatches a `nativeDragDrop` event.**

The event handler can access the transferred data from the event object. If the target is outside AIR, the operating system or another application handles the drop.

In both cases, the initiating object dispatches a `nativeDragComplete` event (if the drag started from within AIR). The `NativeDragManager` class controls both drag-in and drag-out gestures. All the members of the `NativeDragManager` class are static; do not create an instance of this class.

Author: jonniespratley	Subject: Highlight	Date: 1/14/09 3:19:18 AM -08'00'
Understand all of these events when and why they get dispatched, that is on the exam as well		
Author: jonniespratley	Subject: Highlight	Date: 1/14/09 3:16:58 AM -08'00'
Author: jonniespratley	Subject: Highlight	Date: 1/14/09 3:17:08 AM -08'00'
Author: jonniespratley	Subject: Highlight	Date: 1/14/09 3:17:04 AM -08'00'
Author: jonniespratley	Subject: Highlight	Date: 1/14/09 3:17:39 AM -08'00'
Author: jonniespratley	Subject: Highlight	Date: 1/14/09 3:17:21 AM -08'00'
Author: jonniespratley	Subject: Highlight	Date: 1/14/09 3:17:52 AM -08'00'
Author: jonniespratley	Subject: Highlight	Date: 1/14/09 3:18:08 AM -08'00'
Author: jonniespratley	Subject: Highlight	Date: 1/14/09 3:18:23 AM -08'00'
Author: jonniespratley	Subject: Highlight	Date: 1/14/09 3:18:32 AM -08'00'
Author: jonniespratley	Subject: Highlight	Date: 1/14/09 3:18:37 AM -08'00'
Author: jonniespratley	Subject: Highlight	Date: 1/14/09 3:18:42 AM -08'00'

The drag manager sets the `dropAction` property or an abandoned gesture to `NativeDragAction.NONE`.

#### e) Supporting the drag-in gesture

To support the drag-in gesture, your application (or, more typically, a visual component of your application) must respond to `nativeDragEnter` or `nativeDragOver` events. Steps in a typical drop operation

The following sequence of events is typical for a drop operation:

- The user drags a clipboard object over a component.
- The component dispatches a `nativeDragEnter` event.
- The `nativeDragEnter` event handler examines the event object to check the available data formats and allowed actions. If the component can handle the drop, it calls `NativeDragManager.acceptDragDrop()`.
- The `NativeDragManager` changes the mouse cursor to indicate that the object can be dropped.
- The user drops the object over the component.
- The receiving component dispatches a `nativeDragDrop` event.
- The receiving component reads the data in the desired format from the `Clipboard` object within the event object.
- If the drag gesture originated within an AIR application, then the initiating interactive object dispatches a `nativeDragComplete` event. If the gesture originated outside AIR, no feedback is sent.

### 5.5 - Install, uninstall, and update an AIR application.

#### a. Installing, Updating and Uninstalling Air Applications

##### a) Installing

AIR applications are distributed via AIR installer files, which use the `air` extension. When Adobe AIR is installed and an AIR file is opened, the runtime administers and manages the application installation process.

*Note: Developers can specify a version, and application name, and a publisher source, but the initial application installation workflow itself cannot be modified.*

This restriction is advantageous for users because all AIR applications share a secure, streamlined, and consistent installation procedure administered by Adobe AIR. If application customization is necessary, it can be provided when the application is first executed.

The default application installer provides the user with security-related information. AIR displays the publisher name during installation when the AIR application has been signed with a certificate that is trusted, or which chains to a certificate that is trusted on the installation computer. Otherwise the publisher name is displayed as "Unknown." This lets the user make an informed decision whether to install the application or not:

AIR applications first require the runtime to be installed on a user's computer, just as SWF files first require the Flash Player browser

Author: jonniespratley	Subject: Highlight	Date: 1/14/09 3:23:28 AM -08'00'
Author: jonniespratley	Subject: Underline	Date: 1/14/09 3:26:39 AM -08'00'
Author: jonniespratley	Subject: Underline	Date: 1/14/09 3:25:02 AM -08'00'

A user can remove an AIR application:

- On Windows: Using the Add/Remove Programs panel to remove the application.
- On Mac OS: Deleting the application file from the install location.

Removing an AIR application removes all files in the application directory. However, it does not remove files that the application may have written outside the application directory. Removing AIR applications does not revert changes the AIR application has made to files outside the application directory.

Page: 112

Author: jonniespratley	Subject: Underline	Date: 1/14/09 3:28:05 AM -08'00'
On exam		
Author: jonniespratley	Subject: Underline	Date: 1/14/09 3:27:58 AM -08'00'
Author: jonniespratley	Subject: Underline	Date: 1/14/09 3:29:09 AM -08'00'
Author: jonniespratley	Subject: Underline	Date: 1/14/09 3:29:13 AM -08'00'

## 5.6 - List and describe the AIR security contexts.

### a. Security Contexts

#### a) Code Signing

Adobe AIR requires all AIR applications to be digitally signed. Code signing is a process of digitally signing code to ensure integrity of software and the identity of the publisher. Developers can sign AIR applications with a certificate issued by a Certification Authority (CA) or by constructing a self-signed certificate.

Digitally signing AIR files with a certificate issued by a recognized certificate authority (CA) provides significant assurance to users that the application they are installing has not been accidentally or maliciously altered.

Digitally signing AIR files with a certificate issued by a recognized certificate authority (CA) identifies the developer as the signer (publisher). AIR recognizes code-signing certificates issued by the VeriSign and Thawte certificate authorities.

The AIR application installer displays the publisher name during installation when the developer has signed the AIR file with a VeriSign or Thawte certificate.

The AIR application installer displays the publisher name during installation when the AIR application has been signed with a certificate that is trusted, or which chains to a certificate that is trusted on the installation computer. The Certification Authority (CA) verifies the publisher or developer's identity using established verification processes before issuing a high assurance certificate.

Developers can also sign AIR applications using a self-signed certificate; one that they create themselves. However, the AIR application installer presents these applications as originating from an unverified publisher.

When an AIR file is signed, a digital signature is included in the installation file. The signature includes a digest of the package, which is used to verify that the AIR file has not been altered since it was signed, and it includes information about the signing certificate, which is used to verify the publisher identity.

AIR uses the public key infrastructure (PKI) supported through the operating system's certificate store. The computer on which an AIR application is installed must either directly trust the certificate used to sign the AIR application, or it must trust a chain of certificates linking the certificate to a trusted certificate authority in order for the publisher information to be verified.

If an AIR file is signed with a certificate that does not chain to one of the trusted root certificates (and normally this includes all

self-signed certificates), then the publisher information cannot be verified. While AIR can determine that the AIR file has not been altered since it was signed, there is no way to verify who actually created and signed the file.

---

[T] Author: jonniepratley Subject: Underline Date: 1/14/09 3:29:41 AM -08'00'

---

[T] Author: jonniepratley Subject: Underline Date: 1/14/09 3:29:49 AM -08'00'

---

[T] Author: jonniepratley Subject: Underline Date: 1/14/09 3:30:01 AM -08'00'

## b) Security sandboxes

AIR provides a comprehensive security architecture that defines permissions for each file in an AIR application. This includes both those files installed with the application and other files loaded by the application.

Permissions are granted to files according to their origin, and are assigned to logical security groupings called sandboxes.

Files installed with the application are in a directory known as the application directory, and as such, they are, by default, placed in a security sandbox — known as the application sandbox — that has access to all AIR APIs.

This includes APIs that would pose a great security risk if made available to content from sources other than the application resource directory (in other words, files that are not installed with the application).

The AIR security model of sandboxes is composed of the Flash Player security model with the addition of the application sandbox.

Files that are not in the application sandbox have security restrictions like those specified by the Flash Player security model.

The runtime uses these security sandboxes to define the range of data that a file may access and the operations it may execute. To maintain local security, the files in each sandbox are isolated from the files of other sandboxes.

For example, a SWF file loaded into an AIR application from an external Internet URL is placed into the remote sandbox, and does not by default have permission to script into files that reside in the application directory, which are assigned to the application sandbox

## c) Accessing the file system

Applications running in a web browser have only limited interaction with the user's local file system. Web browsers implement security policies that ensure that a user's computer cannot be compromised as a result of loading web content.

For example, SWF files running through Flash Player in a browser cannot directly interact with files already on a user's computer.

Shared objects can be written to a user's computer for the purpose of maintaining user preferences and other data, but this is the limit of file system interaction.

Because AIR applications are natively installed, they have a different security contract with the end user. This contract between the application and the end user is made at install time just like native applications, and it includes the capability for the application to read and write across the local file system.

This freedom comes with a higher degree of responsibility for developers. Accidental application security gaps jeopardize not only the functionality of the application, but also the integrity of the user's computer. The developer documentation includes an "AIR Security" chapter that addresses best practices.

Unless there are administrator restrictions applied to the user's computer, AIR applications are privileged to write to any location on the user's hard drive. However, developers are encouraged to use the user- and application-specific application storage