AIR应用开发中文指南(BETA2)

第一章. 开始 Adobe AIR 之旅

什么是 Adobe AIR

Adobe Integrated Runtime (AIR) 是一个跨操作系统的运行时，利用现有的 Web 开发技术（Flash，Flex，HTML，JavaScript，Ajax）来构建富 Internet 应用程序并部署为桌面应用程序。

AIR 支持现有的 Web 技术如 Flash，Flex，HTML，JavaScript 和 AJAX，可以用你最熟练的技术来开发您所见过的最具用户体验的 RIA 程序，例如，一个 AIR 程序可以使用如下一种或多种组合技术构建：

- ·Flash / Flex / ActionScript
- ·HTML / JavaScript / CSS / Ajax
- ·PDF 可嵌入任何应用程序中

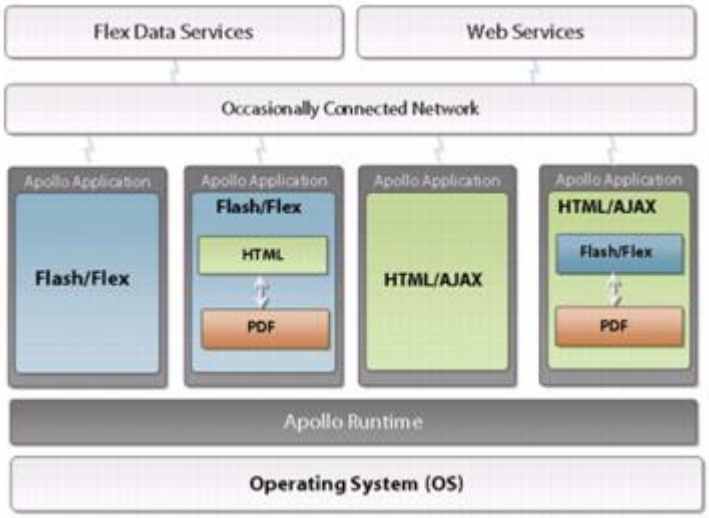
作为结果，AIR 应用程序可以是：

- ·基于 Flash 或 Flex：应用程序根内容（理解为容器）为 Flash/Flex (SWF)
- ·基于 Flash 或 Flex 的 HTML 或 PDF。应用程序的根内容为基于 Flash/Flex (SWF) 的 HTML

(HTML, JS, CSS) 或 PDF

- ·基于 HTML，应用程序根内容为 HTML, JS, CSS
- ·基于 HTML 的 Flash/Flex 或 PDF，应用程序根内容为基于 HTML 的 Flash/Flex (SWF) 或 PDF

用户使用 AIR 应用程序的方式和传统桌面程序是一样的，当运行时环境安装好后，AIR 程序就可以其他桌面程序一样运行了。



因为 AIR 是应用程序运行时环境，因此她很小且对用户来说不可见。运行时环境提供了一套一致的跨操作系统平台和框架来开发和部署应用程序,因此你的程序不必到每个平台上进行测试，在一个平台上开发好就可以在其他平台上运行了，这有很多好处：

- ■·开发 AIR 应用程序不必做额外的跨平台工作，节省了时间，因为跨平台的工作 AIR 都帮我们做好了（只要其他平台能支持 AIR）。
- ■·比起 Web 技术及其设计模式，AIR 应用程序开发迅速，她允许将 Web 开发技术搬到桌面上来而不用另外去学习桌面程序开发技术或复杂的底层代码，这比起低级语言如 C 和 C++更容易学习，且不用去处理每个操作系统复杂的底层 APIs 。

安装运行时及例程

AIR 允许在桌面上运行富 Internet 应用程序（富客户端）。首先，你需要在计算机上安装一个运行时环境，装好后，下载示例程序，试着运行下，看看在桌面上跑的 AIR 程序会是什么样子。

安装运行时环境

按照下面的说明下载和安装 Windows 或 Mac OS X 版本的 AIR,只需要装一次，您就可以在任何时候运行 AIR 程序了。

安装 Windows 版本

1. 从 Adobe Labs 站点下载安装文件(AIR1_win_beta1.exe) 。
2. 双击安装文件 AIR1_win_beta1.exe 。
3. 根据提示完成安装。
4. 如果你修改了 mms.cfg 文件，则在运行 AIR 程序之前删除它，在 Windows 操作系统中该文件放置在 C:\winnt\system32\macromed\flash\mms.cfg。

安装 Mac OS 版本

1. 从 Adobe Labs 站点下载安装文件(AIR1_mac_beta1.dmg) 。
2. 双击安装文件 AIR1_mac_beta1.dmg 。
3. 根据提示完成安装。
4. 如果安装时出现认证窗口，输入 Mac OS 用户名和密码。
5. 如果你修改了 mms.cfg 文件，则在运行 AIR 程序之前删除它，在 Mac OS 操作系统中该文件放置在/Library/Application Support/Macromedia/mms.cfg 。

安装和运行 AIR 示例程序

AIR beta1 包含了一些示例程序。

1. 从 Adobe Labs 站点下载 AIR 示例程序。
2. 双击 AIR 文件 。
3. 在安装窗口中选择 installation 选项，点 Continue 。

4. 安装完成后，打开程序。
5. 在 Windows 系统中，双击桌面上的程序图标。
6. 在 Mac OS 系统中，双击程序图标，它安装在用户目录的 Applications 子目录中(例如, Macintosh HD/Users/JoeUser/Applications/) 。

运行 AIR 应用程序

一旦安装好了运行时环境和 AIR 程序，那运行 AIR 程序和一般的桌面程序就差不多了：

设置开发环境

在编写 AIR 程序之前，你还需要设置开发环境。在 Flex Builder 3 中可开发基于 Flex 和基于 ActionScript 的 AIR 程序，或者使用 Flex 和 AIR SDKs 的命令行工具，如果开发基于 HTML 的 AIR 程序，必须要 AIR SDK，它包含了打包程序所需要的工具。

分发，安装和运行AIR应用程序

AIR 程序是非常易于安装的，无缝安装特性让用户安装最新的 AIR 运行时，安装 AIR 程序，装好后，就可以和普通桌面程序一样运行了。

一旦打包好了 AIR 程序，有几种途径进行分发：

1. 你可以直接通过电子邮件或 Web 页面来发送 AIR 安装包给终端用户。
2. 也可以在 Web 页面上加入一个无缝安装链接，无缝安装特性可在 Web 页面上提供一个链接让用户通过点击改链接来安装 AIR 程序。如果用户没有安装 AIR 运行时，则会提示用户安装运行时，无缝安装技术也可让用户不必下载 AIR 文件就可以安装 AIR 程序。

如果用户下载了 AIR 文件，只要双击文件即可启动 AIR 安装向导，如果用户点击了 Web 页面的无缝安装链接，则会弹出对话框提示用户是否立即安装 AIR 程序。

Windows 系统中在默认设置下，安装 AIR 程序会：

1. 安装在 Program Files 目录
2. 创建桌面快捷键
3. 创建开始菜单快捷键
4. 在添加/删除控制面板中添加程序操作项目

在 Mac OS 中，默认下 AIR 程序安装在用户目录中的 Applications 子目录下。

如果程序已经安装，则安装向导会提示用户是否打开程序或者更新程序，安装器是根据应用程序的 ID（appID）来确定的。

关于AIR安全性

AIR 环境提供和一般桌面程序类似的，能够访问操作系统资源的能力，AIR 程序比起 SWF 或浏览器中 HTML 文件来约束更小些，因此可能会出现安全问题，所以正确理解 AIR 程序的安全模型非常重要。

安装向导安全警告

在 AIR 程序安装过程中，用户会看到一个安全提示，这是 AIR 程序开发者给出的提示信息以及允许程序访问的操作系统类型，这些信息让用户在安装程序时有个大致了解。

安全沙漏

在 AIR beta 版本中，在特定的安全沙漏中 AIR 程序可访问本地文件系统，在将来的 AIR 版本中，应用程序资源将有着不同的安全沙漏，这取决与 AIR 程序是如何被安装的。

对于 SWF 你可以设置安全沙漏类型属性（sandboxType 属性 ）为只读，因为 SWF 可以包含在 AIR 程序里，该属性是 Security.APPLICATION 的一个常量。

所有未和 AIR 程序一起安装的其他资源都被放入同一个安全沙漏中，远程资源根据源域放入沙漏中，本地资源被放入 local-with-networking，local-with-filesystem，或 local-trusted 。

AIR程序安全沙漏中的资源访问特权

AIR 应用程序安全沙漏中的 SWF 可以通过脚本访问来自其他域中的 SWF 文件，但是默认下，外部的 SWF 要想访问 AIR 应用程序安全沙漏内的 SWF 则会受到约束。

AIR 程序沙漏中的 SWF 文件和 HTML 内容可读取任何域中的内容及其数据。

随 AIR 程序安装的 SWF 文件不必寻找 cross-domain 策略文件，要提升权限可调用 Security.allowDomain() 方法。

AIR 为 AIR 程序安全沙漏中的 SWF 文件和 HTML 内容提供了增强型的特权，其中包括读取和写入本地资源和文件。

健壮的应用程序开发最佳实践

当编写 AIR 程序时，你要知道虽然你使用了 Web 技术，但是却不受浏览器安全沙漏限制，也就是说 AIR 程序可能有意或无意中破坏本地系统，AIR 真试图把这种风险降低到最小，但是类似此方面的漏洞仍然可能出现。

产生风险的最大可能就是在读取外部数据或内容时，因此你必须时刻注意使用中的数据是否来自于网络还是本地系统，下面的这些例子都有可能存在潜在的风险：

引入外部内容时

这将导致脚本注入风险：

1. 如果一个 TextField 对象读取的内容中包含链接，则该链接可能会执行不可预料的结果。
2. 如果程序读取一个非信任的 SWF，则该 SWF 可能会以非法特权执行。
3. 如果程序从外部程序中读取 JSON 内容，则该内容可能会访问运行时特权。

影响程序行为的数据

这将会导致弱安全性。比如，如果一个程序使用来自网络上的数据去检测一个文件名或写入一个配置文件，因此需要检测这些数据是否安全以及是否来自信任的数据源。

现在的 AIR 还是 beta 版，如果你遇到了诸如配置和选项的弱安全问题，请告诉 Adobe，目前 Adobe 现在处理这些安全问题，到时他们将会优先给你一份 AIR 1.0 的正式版本。

对于 HTML 内容的安全约束

HTML 内容和其他 AIR 内容一样都放在同一个安全沙漏模型中操作，但是这里有些特殊的情况。 如果内容都在程序安全沙漏内，则 HTML 控制对象中的 HTML 内容只能访问安全约束的 runtime 类（如 JavaScript 对象 window.runtime）。对于基于 HTML 的程序来说，从应用程序资源目录中载入的数据（顶层 Frame）总是可以访问 runtime 类，而从应用程序资源外部载入的数据无论是子框架还是(IFRAME)，还是通过页面定位读取的数据，其权限与原来的域保持一致，不能访问受 AIR 安全约束的 runtime 类，默认下非程序内容是不能访问跨脚本内容，如 JavaScript window 属性，nativeWindow 和 htmlControl 不能在程序沙漏外工作。要想安全访问脚本，可以使用 the flash.system.Door API 创建严密的通信网关在程序内容和非程序内容之间提供一个有限的接口。

AIR文件结构

除了所有文件及图片等资源文件之外，下面两个文件是必须的：

AIR files

用来打包 AIR 程序，主要作为安装文件。

应用程序描述文件

一个 XML 格式的文件，包括每个 AIR 程序（嵌入的 AIR 文件）定义的这种程序属性，例如应用程序名称，appID，以及主程序窗口特征。

当使用 Flex Builder 和 AIR Extensions 时，在创建 AIR 项目时 application.xml 文件会被自动创建。如果你使用 Flex 和 AIR SDKs (包括开发基于 HTML 的应用程序)，你需要手动创建这个文件。另外当使用 Flex Builder 的 AIR Extensions 时，可通过导出应用程序为 AIR 文件，如果使用 Flex 和 AIR SDKs 开发，这需要 ADT 命令行工具生成 AIR 文件。

要用 Flex 开发 Adobe® Integrated Runtime (AIR™) 应用程序，需要如下软件：

- 下载并安装 Adobe Flex Builder 3，用它来创建，测试，调试，打包 AIR 应用程序。
- 下载 Adobe Flex 3 SDK，用你熟悉的文本编辑器和命令行工具来开发 Flex AIR 应用程序。

第二章. 设置 Flex Builder

关于Flex Builder对于AIR的支持情况

Flex Builder 对 AIR 的支持有如下方面：

- 一个新建 AIR 项目的向导

●·自动创建和管理 application.xml 文件
●·运行和调试 AIR 程序
●·自动导出 AIR 项目并打包为 AIR 安装文件

下载 Flex Builder 3

你可以从 Adobe Labs 上下载 Flex Builder 3，下载完安装文件，安装提示安装即可。

从Flex Builder 2.0.1 迁移到Flex Builder 3

使用 Flex Builder 2.0.1 和 AIR Alpha(代号 Apollo)开发的 AIR 程序已经不能被 Flex 和 Flex Builder 3 支持了。

使用 Flex Builder 更新 AIR 程序到新版本

1. 在 Flex Builder 中创建一个 AIR 工程
2. 拷贝和粘帖老的 Apollo 项目代码到新的 AIR 项目中，注意 AIR 程序容器(ApolloApplication) 已经改为 WindowedApplication 。

使用 Flex SDK 更新 AIR 程序到新版本

1. 重新创建 application.xml 文件
2. 更新 AIR 程序容器为(WindowedApplication).

第三章. 设置 Flex SDK

在Windows下安装和配置Flex 3 SDK

Adobe AIR 命令行工具需要安装Java，可以是JRE或JDK（1.4.2 版本以上），JRE到这里下载 http://java.sun.com/j2se/1.4.2/download.html，JDK到这里下载 http://java.sun.com/javase/downloads/index.jsp 。

注意：终端用户不需要 Java 环境。

Flex SDK 包含AIR API和命令行工具用于打包，编译和调试AIR应用程序。

1. 如果还没有，可到 Adobe Labs 上下载一份 Flex 3 SDK。
2. 解压缩 SDK 到指定目录。
3, 定位到 bin 子目录。

编译器设置

Flex SDK 中包含两个编译器，mxmlc 编译器编译 MXML 和 ActionScript 代码为 SWF 文件，compc 编译器编译组件和库为 SWC 文件。两个编译器都可作为本地二进制程序或 Java 程序在命令行下运行。（本地二进制程序实际上是调用 Java 程序）如果要使用本地二进制程序，需要把 Flex 3 SDK\bin 目录加入环境变量 path 中。如果使用 Java 程序命令，需要把 mxmlc.jar 和 compc.jar 加入到环境变量中。

编译器配置文件

使用编译器时可以指定编译的一些可选参数，全局 Flex SDK 配置文件包含一个默认值，你可以编辑这个文件定制自己的开发环境。air_config.xml 这个全局配置文件在 Flex 3 SDK 的 frameworks 目录中。

注意：如果使用 amxmlc 命令启动编译器时 air_config.xml 会代替 flex_config.xml。

关于编译器的可选选项可参考*(http://livedocs.macromedia.com/flex/2/docs/00001490.html)*

调试器设置

AIR 直接支持调试功能，因此不需要一个调试版本的运行时环境。要想管理命令行调试，需要使用调试版的 Flash 播放器，只需要设置下环境变量指定这些命令的所在目录即可。

调试版的 Flash 播放器已在 Flex 3 SDK 目录中。二进制命令 fdb.exe 在 bin 目录，Java 版的在 lib 目录。AIR 调试启动器 adl.exe 或 ADL 在 bin 目录

注意，你不能直接用 FDB 启动 AIR 程序，因为 FDB 会试图用 Flash 播放器运行程序，因此你必须让 AIR 程序连接 FDB 回话。

应用程序打包器设置

AIR 开发工具(ADT)，用来把程序打包为 AIR 文件，要想运行 ADT，必须安装 Java 环境。

SDK 包含一个脚本来执行 ADT。要运行 ADT 脚本需要把 Flex SDK 的 bin 目录加到系统 path 变量中。

在最近的 Flex 项目中需要做个屏幕截图功能，然后保存为图片文件，以前好像在哪里看到这样的例子，找了半天没找着，其实实现起来也挺简单的。

具体步骤如下：

- 创建一个 BitmapData 对象
- 拷贝目标组件的象素数据到 BitmapData 对象上
- 转换 BitmapData 对象为 PNG 编码的 ByteArray (需要用到 PNGEnc 库)
- 转换 ByteArray 为 Base64Encoded 字符串，这样便于发送数据给后台处理
- 在后台程序中(如 PHP 等)，对数据解码然后写入文件

PNG Encoder 库是由Tinic Uro编写的。

这里是 Flex 代码：

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

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
```

```
<mx:Script>
<![CDATA[
import mx.utils.Base64Encoder;
import mx.rpc.events.ResultEvent;
import mx.utils.ObjectUtil;
import mx.controls.Alert;
import mx.rpc.events.FaultEvent;
import mx.core.UIComponent;

public function onResult(event:ResultEvent) :void
{
Alert.show(ObjectUtil.toString(event.result));
}

public function onFault(event:FaultEvent) :void
{
Alert.show("Got error: "+event.message);
}

public function takeSnapshot(target:UIComponent) :void
{
var bd:BitmapData = new BitmapData(target.width,target.height);
bd.draw(target);
var ba:ByteArray = PNGEnc.encode(bd);
var be:Base64Encoder = new Base64Encoder();
be.encodeBytes(ba);
var encodedData:String = be.flush();
ro.saveImage(encodedData);
}
]]>
```

```
</mx:Script>

<mx:Button click="takeSnapshot(targetPanel)" label="Save Image" x="10" y="100"/>

<mx:Panel id="targetPanel">
<mx:Canvas backgroundColor="#EEEEEE">
<mx:Label text="Hello World" />
</mx:Canvas>
</mx:Panel>

<mx:RemoteObject id="ro" destination="serviceEndpoint" result="onResult(event)" fault
="onFault(event)"/>
</mx:Application>
```

后台的 PHP 代码：

```
public function saveImage($encodedPNGData)
{
if ($encodedPNGData != "")
{
$binaryData = base64_decode($encodedPNGData);
$file = "assets/images/something.png";
file_put_contents($file, $binaryData);
return $file;
}
return null;
}
```

GDS服务器（Granite Data Services） 目前版本为 0.4.0 RC1,下载地址为 http://www.graniteds.org/confluence/display/INTRO/Granite+Data+Services

该版本的特性如下：

- Full AMF3 support. See GDS AMF3 documentation.
- EJB3 services with transparent externalization mechanism and lazy initialized ActionScript 3 beans (Entity Beans / Hibernate). See EJB3 Services and Externalizers and Lazy Initialization.
- EJB3 Entity Bean to ActionScript 3 classes code generator. See AS3 Generation.

- Spring services. See documentation on Spring Services.
- POJO services (remote calls to simple Java classes that expose public methods). See Pojo Services.
- (planned) Data push. A Comet-like implementation with AMF3 data polling over HTTP (event/listener based architecture).
- (planned) Entity repository: a client side entity repository that ensures uniqueness (only one instance of each entity is present in the flash VM), weakness (only currently bound objects are kept in memory), and that acts as a services frontend (all server calls/events are managed by this central component). This will be loosely inspired by Cairngorm.
- (planned) Seam integration: a reliable GDS/Seam integration with full scopes (at least conversation) support.
- (planned) A set of Flex components suitable for complex data structures.

呵呵，还是很值得期待的哦！

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

 <!-- http://blog.flexexamples.com/2007/09/10/styling-the-titlewindow-container/ -->

 <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"

      layout="vertical"

      verticalAlign="middle"

      backgroundColor="white">


   <mx:Style>
```

```
    /* Style the Window title message. */
    .windowStyles {
        color: haloBlue;
        letterSpacing: 2;
    }

    /* Style the Window status message. */
    .windowStatus {
        color: red;
        fontWeight: bold;
    }

    TitleWindow {
        dropShadowEnabled: false;
        borderAlpha: 1.0;
        borderColor: haloSilver;
        backgroundColor: haloSilver;
        cornerRadius: 0;
    }
</mx:Style>

<mx:TitleWindow id="titleWindow"
        title="Title"
        status="Status"
        width="160">
    <mx:Text htmlText="The quick brown fox jumped over the lazy dog."
            selectable="false"
            width="100%" />
    <mx:ControlBar horizontalAlign="right">
        <mx:Button label="Button" />
```

```
    </mx:ControlBar>
  </mx:TitleWindow>

</mx:Application>
```

效果：

1：

```
<mx:Button label="Button 1" borderColor="red" />
```

2：

```
<mx:Button label="Button 2">
  <mx:borderColor>red</mx:borderColor>
</mx:Button>
```

3：

```
<mx:Style>
  .MyButton {
    borderColor: red;
  }
</mx:Style>

<mx:Button label="Button 3" styleName="MyButton" />
```

4：

```
<mx:Style>
  Button {
    borderColor: red;
  }
</mx:Style>
```

```
<mx:Button label="Button 3" />
```

5:

```
<mx:Style source="styles.css" />

<mx:Button label="Button 3" />
/* CSS file */
Button {
    borderColor: red;
}
```

6:

```
<mx:Script>
    <![CDATA[
        private function button4_init():void {
            button4.setStyle("borderColor", "red");
        }
    ]]>
</mx:Script>

<mx:Button id="button4"
        label="Button 4"
        creationComplete="button4_init();" />
<?xml version="1.0" encoding="utf-8"?>
<!-- http://blog.flexexamples.com/2007/09/10/finding-out-which-fonts-are-installed-on-a-users-system/ -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
        layout="vertical"
        verticalAlign="middle"
```

```
    backgroundColor="white"
    creationComplete="init()">

<mx:Script>
    <![CDATA[
        import flash.text.Font;

        private function init():void {
            arr = Font.enumerateFonts(true);
            arr.sortOn("fontName", Array.CASEINSENSITIVE);
        }
    ]]>
</mx:Script>

<mx:Array id="arr" />
<mx:String id="str">The quick brown fox jumped over the lazy dog.</mx:String>

<mx:ApplicationControlBar dock="true">
    <mx:Label text="String:" />
    <mx:TextInput id="textInput" text="{str}" />

    <mx:Spacer width="100%" />

    <mx:Label text="Number of installed fonts: {arr.length}" />
</mx:ApplicationControlBar>

<mx:DataGrid id="dataGrid" dataProvider="{arr}">
    <mx:columns>
        <mx:DataGridColumn dataField="fontName"
            width="200"
```

```
            itemRenderer="mx.controls.Label" />
        <mx:DataGridColumn dataField="fontStyle" />
        <mx:DataGridColumn dataField="fontType" />
    </mx:columns>
</mx:DataGrid>

<mx:Label id="lbl"
        text="{textInput.text}"
        width="{dataGrid.width}"
        height="32"
        fontFamily="{dataGrid.selectedItem.fontName}"
        fontSize="16" />

</mx:Application>
```

效果：

如果修改上面的例子，并使用下面的代码嵌入一个字体，那么在表格中也会显示出来“Base02"字体：

```
<mx:Style>
    @font-face{
        src: url("./fonts/base02.ttf");
        fontFamily: "Base02";
    }
</mx:Style>
```

如果把 enumerateFonts()方法的参数改为 flase，那么只会列举出嵌入字体：

```
<mx:Script>
    <![CDATA[
        import flash.text.Font;
```

```
        private function init():void {
            arr = Font.enumerateFonts(false);
            arr.sortOn("fontName", Array.CASEINSENSITIVE);
        }
    ]]>
</mx:Script>
```

删除mms.cfg设置

如果你事先修改了 mms.cfg 文件(如 Flash 播放器的安全设置)，在测试 AIR 之前先删除它，在 AIR 的 M2 版本里，这个配置文件的一些设置会限制 AIR 的功能。

- 在 Mac OS 系统中，这个文件在/Library/Application Support/Macromedia/mms.cfg 。
- 在 Microsoft Windows 系统中，这个文件在如 C:\winnt\system32\macromed\flash\mms.cfg 。

第四章. 用 Flex Builder 创建第一个 Flex AIR 程序

创建一个AIR工程

开始本章内容之前，记得先安装好 Adobe AIR 运行时以及设置好开发环境。

在 Flex Builder 里创建工程

1. 打开 Flex Builder 3.
2. 选择菜单 File | New | AIR Project (如果你使用 Eclipse 插件形式，选择菜单 File | New | Other，打开 Flex 节点，选择 AIR 工程)。
3. 接受默认设置，点击 Next。
4. 在下个对话框中(设置构建路径)，不用改变，点 Next。
5. 指定下面的设置，然后点 Finish：
   ID：AIRHelloWorld
   Name：Hello World
   Description：A test AIR application
   Copyright：2007

其中 Description 和 Copyright 域是可选的，但是其他的都是必填的。

设置应用程序窗口的透明性

1. 在 Project Navigator 视图中打开 AIRHelloWorld-app.xml 文件。
2. 在 rootContent 节点中，设置 systemChrome 和 transparent 属性为：

systemChrome="standard" transparent="false"

保存修改，关闭 AIRHelloWorld-app.xml 文件。

编写程序代码

要编写这个"Hello World"程序代码，需要编辑此程序的 MXML 文件(AIRHelloWorld.mxml)，在 Project Navigator 视图中找到它并打开。

所有的 Flex AIR 程序都包含在 MXML WindowedApplication 标签内，它创建了一个简单的窗口，只包含标题栏和关闭按钮。

添加代码

1. 在 WindowedApplication 组件中添加一个 title 属性，赋值为"Hello World"：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute" title="Hello World">


</mx:WindowedApplication>
```

2. 添加一个 Label 组件，设置 text 属性为"Hello AIR"，设置对齐方式：

```
<?xml version="1.0" encoding="utf-8"?><?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute" title="Hello World">
    <mx:Label text="Hello AIR" horizontalCenter="0" verticalCenter="0"/>
</mx:WindowedApplication>
```

3. 添加下面的样式：

```
<mx:Style>
    Application
    {
```

```
        background-image:"";
        background-color:"";
        background-alpha:"0.5";
    }
</mx:Style>
```

该样式会应用到整个程序，窗口背景有点灰色透明。

完整的代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" l
ayout="absolute" title="Hello World">
    <mx:Style>
        Application
        {
            background-image:"";
            background-color:"";
            background-alpha:"0.5";
        }
    </mx:Style>
    <mx:Label text="Hello AIR" horizontalCenter="0" verticalCenter=
"0"/>
</mx:WindowedApplication>
```

测试程序

1. 点击工具栏上的调试按钮 。

   也可以选择 Run | Debug | AIRHelloWorld 命令

   运行后的结果可能会是这样(用户桌面的背景为绿色)：

2. 使用 Label 组件的 horizontalCenter 和 verrticalCenter 属性，文本会放置在窗口的中央，你可以任意移动或改变窗口的大小，这和普通的桌面程序没什么不同。

打包并运行程序

现在可以用 Flex Builder 对"Hello World"程序打包为 AIR 文件以便分发。一个 AIR 文件是一个压缩文件，它包含程序文件(这些文件都包含在工程的 bin 目录中)，把这些 AIR 文件分发给用户以便用户用它进行安装。

确定你的程序没有编译错误信息以及运行异常。

选择菜单 File | Export.
选择 AIR Package 选项，点 Next.
默认工程名和 MXML 文件被选中作为 AIR 包中的内容，点 Finish 完成创建 AIR 包。

现在你可以在桌面上双击运行 AIR 文件。

第五章. 用 Flex SDK 创建第一个 Flex AIR 程序

创建应用程序的XML文件

每个 AIR 程序都需要一个应用程序描述文件，这个 XML 文件定义各种属性，并嵌入到 AIR 包中分发给用户。

要创建应用程序描述文件，可使用文本编辑器来创建一个 XML 文件并命名为 AIRHelloWorld-app.xml ，然后加入一下内容：

```
<?xml version="1.0" encoding="UTF-8"?>

<application xmlns="http://ns.adobe.com/air/application/1.0.M4"

    appId="com.adobe.air.example.AIRHelloWorld" version="1.0">

    <name>Hello World</name>

    <description>A test AIR application.</description>

    <copyright>2007</copyright>

<rootContent systemChrome="none" transparent="true" visible="true">AIRHelloWorld.swf</rootContent>

</application>
```

编写代码

和所有的 Flex 程序一样，用 Flex framework 构建的 AIR 程序也包含一个主 MXML 文件，但是不同的是其根组件不是 Application 而是 WindowedApplication。WindowedApplication 组件提供了作为桌面程序所需要的最基本的窗口以及窗口控件，下面的步骤将创建一个 Hello World 程序。

1. 使用文本编辑器，创建一个名为 AIRHelloWorld.mxml 文件，其内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="absolute" title="Hello World">
</mx:WindowedApplication>
```

2. 下一步，添加 Label 组件，设置 text 属性为"Hello AIR"，设置对齐方式：

```
<?xml version="1.0" encoding="utf-8"?><?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
   title="Hello World">
      <mx:Label text="Hello AIR" horizontalCenter="0" verticalCenter="0"/>
</mx:WindowedApplication>
```

3. 添加样式：

```
<mx:Style>
   Application
   {
      background-image:"";
      background-color:"";
      background-alpha:"0.5";
   }
```

```
</mx:Style>
```

这些样式将应用与整个程序。

这么是完整代码：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute" title="Hello World">
    <mx:Style>
        Application
        {
            background-image:"";
            background-color:"";
            background-alpha:"0.5";
        }
    </mx:Style>
    <mx:Label text="Hello AIR" horizontalCenter="0" verticalCenter="0"/>
</mx:WindowedApplication>
```

编译程序

在运行和调试程序之前需要把 MXML 编译为 SWF 文件。（确定已经把 AIR 命令行工具加入到 ClassPath）。

打开控制台，定位到源文件所在目录

输入以下命令：

amxmlc AIRHelloWorld.mxml

测试程序

要在命令行下运行和测试程序，需要用 AIR Debug Launcher (ADL) 根据应用程序描述文件来运行程序。

在控制台输入以下命令：

adl AIRHelloWorld-app.xml

运行结果大概如下：

使用 Label 组件的 horizontalCenter 和 verticalCenter 属性可使文本居中显示。

打包程序

现在准备把"Hello World"程序打包为 AIR 文件以便分发，一个 AIR 包是一个压缩文件，AIR 包分发给用户，用户使用它进行安装。

1. 确定你的程序没有编译错误以及运行异常。
2. 在命令行中输入以下命令：

```
adt -package AIRHelloWorld.air AIRHelloWorld-app.xml AIRHelloWorld.swf
```

第一个参数指定 AIR 文件名，第二个参数指定应用程序描述文件，接下来的参数指定程序需要的模块，脚本，资源，这里只是指定一个文件，实际上可以是任意多个文件和目录。

AIR 包创建后，双击，Ctrl-Click，或在命令行下输入 AIR 文件名安装。

第六章. 用 Flex Bilder 开发 AIR 程序

创建AIR工程

1. 打开 Flex Builder 3.
2. 选择菜单 File > New > AIR Project
3. 根据默认选项，点 Next.
4. 输入工程名，点 Next.
5. 使用 ActionScript 定义主类，改变主应用程序为 as 文件扩展(默认下为 Flex MXML .mxml 文件扩展)
6. 点 Next
7. 设定 AIR application 设置，点 Finish.

调试AIR程序

Flex Builder完美支持AIR程序的调试，更多信息请看Flex Builder 帮助。

1. 打开程序源文件(如 MXML 文件)
2. 点击调试按钮 .

也可选择菜单 Run > Debug.

程序启动并在 ADL(AIR Debugger Launcher)内运行,Flex Builder 调试器会捕获任何断点或运行时错误，跟调试 Flex 程序没什么不同。

你还可以使用 AIR Debug Launcher 命令行工具调试程序。

打包AIR程序

当程序已经开发完成，并准备分发，可把它打包为 AIR 文件。

1. 打开工程并确认程序无编译错误以及运行异常。
2. 选择菜单 File > Export.
3. 选择部署为 AIR 文件，点 Next.
4. 选择文件(如媒体文件或 SWF 文件) 加入 AIR 文件，application.xml 文件和主 SWF 文件默认已经加入。
5. 设置 AIR 文件描述信息，包括文件名，点 Finish.

你也可以使用 ADT 命令行工具打包程序。

创建AIR库工程

Beta 版本的 Flex Builder 3 没有提供创建 AIR 库工程的工程向导。因此要创建自定义 AIR 组件，必须先创建一个标准的库工程，然后手动编辑并创建指向 AIR 类的库工程。

1. 选择菜单 File > New > Flex Library Project.
2. 设定工程名，点 Next.
3. 点击 Library Path 标签
4. 删除${FRAMEWORKS}/libs/playerglobal.swc.
5. 添加 airglobal.swc ， 设置 link 类型为 External.
6. 点 Finish.

第七章. 使用 Flex AIR 组件

关于Flex AIR组件

Flext 提供了下列 AIR 组件：

WindowedApplication 容器

AIR 应用程序的顶层组件，该组件提供一些与桌面窗口有关的功能。

HTML 控件

可显示 HTML 页面

FileSystemComboBox 控件

定义一个组合框用于选择本地文件

FileSystemDataGrid 控件

用表格来显示文件信息，其信息包括文件名，创建日期，修改日期，类型，大小。

FileSystemHistoryButton 控件

使用 FileSystemList 或 FileSystemDataGrid 控件的时候用此控件可导航历史操作记录。

FileSystemList 控件

显示文件系统目录内容。

FileSystemTree 控件

以树结构的形式显示文件系统目录内容。

更多信息请参阅 *Flex 3 Language Reference.*

例子：用 Flex AIR 显示一个目录结构

下面的例子使用了 WindowedApplication 容器和 FileSystemTree 以及 FileSystemDataGrid 控件。在这里例子中，FileSystemTree 控件显示目录结构，FileSystemTree 控件中的目录名，FileSystemDataGrid 控件就显示该目录下的文件信息：

```
<?xml version="1.0" encoding="utf-8"?>
  <mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:HDividedBox>
    <mx:FileSystemTree id="tree"
      width="200" height="100%"
      directory="{new File('C:\')}"
      enumerationMode="directoriesOnly"
      change="dataGrid.directory = File(tree.selectedItem);"/>
    <mx:FileSystemDataGrid id="dataGrid"
      width="100%" height="100%"
      directory="{new File('C:\')}"/>
    </mx:HDividedBox>
  </mx:WindowedApplication>
```

使用WindowedApplication组件

mx:WindowedApplication 容器组件定义了包含 AIR 应用程序的窗口控件。在 MXML AIR 程序里<Application> 标签被替换为<WindowedApplication>标签。

一个 WindowedApplication 组件提供下列控件：

1. 一个标题栏
2. 一个最小化按钮
3. 一个最大化按钮
4. 一个关闭按钮

WindowedApplication 组件的窗口遵循底层操作系统的标准行为，比如可以拖到标题栏移动窗口以及改变窗口大小。

默认下，WindowedApplication 组件创建的应用程序窗口，其 windowMode 属性设置为 systemChrome，visibile 设置为 true，这些设置都在 application.xml 文件中。

下面的例子简单演示了 WindowedApplication 组件：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication
   xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="absolute"
   <mx:Text text="Hello World" />
</mx:WindowedApplication>
```

关于 HTML 组件

HTML 组件用于显示 HTML 网页内容，被用于在 AIR 程序中渲染外部的指定 HTML 内容。它提供了轻量级的浏览器的功能，包括载入 HTML 页面，历史记录导航，以及访问 HTML 内容的能力。HTML 组件并不是用来代替 Text 和 TextArea 组件来显示格式化文本数据。

创建一个 HTML 组件

使用<mx:HTML> 标签在 MXML 中定义一个 HTML 组件，下面的例子中，给其指定一个 id 以便在其他地方能够引用。

指定 HTML 页面的 location 属性显示指定页面内容。

下面的例子演示如何使用 HTML 组件。HTML 组件的 location 属性设置为"http://labs.adobe.com/"，这样当载入时 URL 地址将被打开，另外"back" 和"forward" 按钮调用组件的 historyBack() 和 historyForward()方法。TextInput 组件让用户输入 url 地址，当"go" 按钮被点击后，HTML 组件的 location 属性被设置为 TextInput 的 text 属性值。

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

<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml">

   <mx:ControlBar width="100%">

      <mx:Button label="< Back" click="content.historyBack();"/>

      <mx:Button label="Forward >" click="content.historyForward();"/>
```

```
        <mx:TextInput id="address" text="{content.location}" width="100%"/>

        <mx:Button label="Go!" click="content.location = address.text"/>

    </mx:ControlBar>

    <mx:Canvas width="100%" height="100%">

        <mx:HTML id="content" location="http://labs.adobe.com/"/>

    </mx:Canvas>

</mx:WindowedApplication>
```

HTML 组件之用户交互能力

对于用户交互性而言，HTML 组件就像一个简单的浏览器，没有菜单栏和导航按钮。HTML 页面的内容显示在组件中。用户通过表单域，按钮和超链接操作内容，任何动作都会使浏览器载入新页面（比如点击一个连接或提交一个表单），改变组件的 location 属性可载入新的页面。

Window 容器

Window组件是一个 Flex容器，用于定义一个程序运行后出现所包含的内容和布局的操作系统窗口，也就是说它不同于初始或主窗口（如WindowedApplication或Application组件）。除此之外和WindowedApplication组件具有共同的功能，Window组件允许定义窗口的特性如窗口类型，样式，是否包含特定的窗口操作（如改变大小和最大化）。这些都是通过组件初始化时（还没有显示出来）设置其属性来访问，但是，一旦窗口打开后，这些属性将不能被设置和访问。

创建和使用 Window 容器

<mx:Window>容器定义了包含自身的 AIR 程序对象。在 MXML AIR 程序里<mx:Window> 标签作为 MXML 部件的最顶层标签，MXML 部件的文档内容作为 window 容器内容，Window 组件不能用在其他 MXML 文档中，只能通过 ActionScript 来创建组件实例。

因许多 Window 组件只能在 window 打开之前进行设置，所以其属性可以在 <mx:Window> MXML 标签中或用 ActionScript 代码进行设置。

一旦 windows 的初始属性设置完毕，调用 Window 组件的 open() 方法后操作系统将会把它显示在用户桌面上。

下面是一个简单使用 Window 组件的例子，这个例子包含两个 MXML 文件，第一个使用 Application 容器且是程序的初始窗口，第二个使用 Window 容器作为程序的第二个窗口，在这个例子中，主窗口模拟一个应用程序的欢迎屏幕窗口，3 秒后关闭欢迎屏幕且打开第二个窗口。

下面的代码定义主程序 MXML 文件，包含当程序运行时自动打开的初始化窗口(欢迎屏幕窗口)：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute" creationComplete="init();">
    <mx:Script>
        <![CDATA[
            private const LOAD_DELAY:int = 3;
            private var timeElapsed:int = 0;
            private var loadTimer:Timer;

            private var splashScreen:NativeWindow;
            private var docWindow:DocumentWindow;

            private function init():void
            {
                // center the window on the screen
                splashScreen = Shell.shell.openedWindows[0];
                var screenBounds:Rectangle = Screen.mainScreen.bounds;
                splashScreen.x = (screenBounds.width - splashScreen.width) / 2;
                splashScreen.y = (screenBounds.height - splashScreen.height) / 2;

                // start the timer, which simulates a loading delay
                loadTimer = new Timer(1000);
```

```
            loadTimer.addEventListener(TimerEvent.TIMER, incrementTime);
            loadTimer.start();

            updateStatus();
        }

        private function incrementTime(event:TimerEvent):void
        {
            timeElapsed++;

            updateStatus();

            // if the loading delay has passed, stop the timer,
            // close the splash screen, and open the document window
            if ((LOAD_DELAY - timeElapsed) == 0)
            {
                loadTimer.stop();
                loadTimer.removeEventListener(TimerEvent.TIMER, incrementTime);
                loadTimer = null;

                splashScreen.close();

                // open a new instance of the document window
                docWindow = new DocumentWindow();
                docWindow.open();
            }
        }

        private function updateStatus():void
        {
```

```
            loadStatusMessage.text = "initializing... " + (LOAD_DELAY - timeElapsed).to
String() + " seconds remaining.";
        }
      ]]>
    </mx:Script>
    <mx:VBox horizontalCenter="0" verticalCenter="0">
      <mx:Text text="My Splash Screen" fontFamily="Courier New" fontSize="36"/>
      <mx:Text id="loadStatusMessage" text="initializing..."/>
    </mx:VBox>
  </mx:Application>
```

incrementTime()方法每秒钟调用一次，当时间结束时 DocumentWindow 实例被创建并调用其 open()方法。DocumentWindow 类被独立定义在 MXML 文件中，其顶层标签为<mx:Window>，也就是说它是 Window 类（Window 组件）的子类。下面是 DocumentWindow MXML 文件代码：

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

<mx:Window xmlns:mx="http://www.adobe.com/2006/mxml"

    layout="absolute"

    title="Document window"

    width="550" height="450">

    <mx:Text text="This is a document window." horizontalCenter="0" verticalCenter="0
"/>

</mx:Window>
```

关于Window容器的更多信息，请参阅 *Flex 3 语言参考*

第八章. 使用命令行工具创建 AIR 应用程序

使用amxmlc编译器编译程序代码

使用命令行 MXML 编译器（amxmlc）编译 ActionScript 和 MXML 资源：

amxmlc *[compiler options]* -- MyAIRApp.mxml

这里的*[compiler options]* 指定编译器选项。

Amxmlc 命令调用 mxmlc，再加上额外的参数，+configname=air，它指示编译器使用 air-config.xml 代替 flex_config.xml 文件。

编译器根据 air-config.xml 配置文件编译 AIR 程序，你也可以使用本地的，工程级别的配置文件来代替全局配置文件，比如你可以先复制全局文件进行修改，然后通过-load-config 选项载入：

-load-config=project-config.xml 替换全局文件

-load-config+=project-config.xml 增加额外的参数值到全局变量中，比如-library-path 选项

你也可以使用指定的命名约定，让编译器自动载入配置文件，例如如果你的程序主 MXML 文件名为 RunningMan.mxml，那么配置文件名为 RunningMan-config.xml。编译程序只需要输入：

amxmlc RunningMan.mxml

示例

下面的例子演示了如何使用 amxmlc 编译器

编译一个 AIR MXML 文件：

amxmlc myApp.mxml

编译并设置输出：

amxmlc -output anApp.swf -- myApp.mxml

编译一个 AIR ActionScript 文件：

amxmlc myApp.as

指定编译器配置文件：

amxmlc -load-config config.xml -- myApp.mxml

从其他配置文件读取额外参数：

amxmlc -load-config+=moreConfig.xml -- myApp.mxml

添加外部库：

amxmlc -library-path+=/libs/libOne.swc,/libs/libTwo.swc -- myApp.mxml

不使用配置文件编译 AIR MXML 文件：

mxmlc -library-path [AIR SDK]/frameworks/libs/air/airframework.swc, ^
[AIR SDK]/frameworks/libs/air/airframework.swc, ^
-library-path [Flex 2 SDK]/frameworks/libs/framework.swc ^
-- myApp.mxml

使用 runtime-shared library（RSL）编译 AIR MXML 文件：

amxmlc -external-library-path+=../lib/myLib.swc -runtime-shared-libraries=myrsl.swf -- myApp.mxml

使用 Java 版本的编译器：

java flex2.tools.Compiler +flexlib [Flex SDK 2]/frameworks +configname=air [additional compiler options] -- myApp.mxml

Flexlib 选项指定 Flex SDK frameworks 目录，使编译器找到 flex_config.xml 文件。

java -jar [Flex SDK 2]/lib/mxmlc.jar +flexlib [Flex SDK 2]/frameworks +configname=air [additional compiler options] -- myApp.mxml

使用acompc编译器编译AIR组件或库

使用组件编译器 acompc 编译 AIR 库或独立组件。组件编译器很类似 amxmlc 编译器，只是需要注意以下事项：

1. 你必须指定哪些类将被编译进库或组件。
2. Acompc 不会自己去寻找本地配置文件，你必须手动使用-load-config 选项加载。

Acompc 和 compc 一样，除了载入配置文件 air-config.xml 文件代替 flex_config.xml 文件。

组件编译器配置文件

下面的例子演示建立用两个类所建立的库：ParticleManager 和 Particle，都在 com.adobe.samples.particles 包中，存放在 source/com/adobe/samples/particles 目录下：

```
<flex-config>
    <compiler>
  <source-path>
        <path-element>source</path-element>
```

```
    </source-path>
        </compiler>
        <include-classes>
            <class>com.adobe.samples.particles.ParticleManager</class>
            <class>com.adobe.samples.particles.Particle</class>
  </include-classes>
    </flex-config>
```

然后输入下面的命令：

compc -source-path source -include-classes com.adobe.samples.particles.Particle

com.adobe.samples.particles.ParticleManager

示例

编译 AIR 组件或库：

acompc -load-config myLib-config.xml -output lib/myLib.swc

编译 runtime-shared library：

acompc -load-config myLib-config.xml -directory -output lib

(注意，在运行此命令之前 lib 目录必须存在且空的)

使用AIR Debug Launcher进行调试

在开发过程中可使用 AIR Debug Launcher (ADL)来调试基于 flex 或 HTML 的 AIR 程序。使用 ADL，你可不必先对程序打包和安装，使用 ADL 也不需要安装运行时。

ADL 所支持的调试只限于 trace 语句的输出，如果你开发基于 Flex 的程序，可使用 Flash Debugger (或 Flex Builder) 调试复杂问题。

用 ADL 运行程序

使用下面的语法：

adl [-runtime <runtime-directory>] <application.xml> [<root-directory>] [-- arguments]

-runtime <runtime-directory> 指定要使用的运行时，如果没指定，则默认为 ADL 所在的 SDK 目录（如果 ADL 被移动到 SDK 目录外，则需要手动指定）

<application.xml> 应用程序描述文件

<root-directory> 应用程序运行所在的根目录，如果没有指定，则该目录就是程序描述文件所在目录。

-- arguments 任意字符串参数

注意：当你想运行的 AIR 程序已经在运行的话，新的实例将不能运行。

打印 trace 语句

要在 ADL 下输出 trace 语句到控制台，可使用 trace() 函数：

trace("debug message");

ADL 示例

在当前目录下运行程序：

adl myApp-app.xml

在当前目录的子目录下运行程序：

adl source/myApp-app.xml release

运行程序并传递两个命令行参数，"foo"和"bar"：

adl myApp-app.xml -- foo bar

用指定的运行时运行程序：

adl -runtime /AIR/XYZ/AIRSDK/bin myApp-app.xml

在 Flash Debugger 中设置断点

要在 Flash Debugger 中调试基于 SWF 的 AIR 程序，需要启动一个 FDB 会话，且运行 debug 版本的应用程序。Debug 版本的 SWF 文件会自动连接到 FDB 会话。

1. 启动 FDB，FDB 命令在 Flex 2 SDK 的 bin 目录中。

   在控制台中会显示 FDB 提示：〈fdb〉

2. 执行 Run 命令：〈fdb〉run 【回车】

3. 运行 debug 版本的程序：

   adl myApp-debug.xml

4. 使用 FDB 命令设置断点。

输入：continue 【回车】
设置断点
输入：continue 【回车】

使用AIR开发工具打包程序

用 AIR Developer Tool（ADT）打包程序为 AIR 文件。ADT 创建基于 SWF 或 HTML 的安装文件（如果你使用 Flex Builder，可用导出功能）

ADT 是 java 程序，类似与 Ant 需要在命令行下运行。SDK 中包含了命令行脚本用于执行该命令。

最简单的 AIR 程序至少需要一个程序描述文件和主 SWF 或 HTML 文件。任何其他用到的资源都会被打包进 AIR 文件。

ADT 用法

使用下面的语法：

adt -package *air_file app_xml* [*file_or_dir* | -C *dir file_or_dir* ...] ...

air_file ： 即将创建的 AIR 文件名。

app_xml ： 程序描述文件路径，不管该文件名是什么，最后打包后都改为"application.xml"，该路径可以是相对也可以是绝对路径。

file_or_dir ：将被打包的文件和目录，可以指定任意数量的文件和目录，用空格符分隔开。如果是目录，则该目录下的所有文件和子目录，除了隐藏文件都被添加到安装包中。指定的文件和目录必须在当前目录下或是当前目录的子目录，可使用-C 选项改变当前目录。

ADT 示例

在当前目录中打包指定的程序文件：

adt -package myApp.air myApp.xml myApp.swf components.swc

打包当前目录中的所有文件和子目录：

adt -package myApp.air myApp.xml .

只打包主文件和 images 子目录：

adt -package myApp.air myApp.xml myApp.swf images

打包和(release\bin)目录下的程序描述文件和 SWF 文件：

adt -package myApp.air release\bin\myApp.xml -C release\bin myApp.swf

下面的例子演示如何打包多个目录下的文件：

/devRoot

/myApp

/release

/bin

myApp.xml

myApp.swf

/artwork

/myApp

/images

image-1.png

image-n.png

/libraries

/release

/libs

lib-1.swf

lib-n.swf

下面的 ADT 命令在/devRoot/myApp 目录下运行：

adt -package myApp.air release/bin/myApp.xml -C release/bin myApp.swf

-C ../artwork/myApp images -C ../audio

该命令执行结果：

/myAppRoot

/META-INF

/AIR

application.xml

hash

myApp.swf

mimetype

/images

image-1.png

image-n.png

/libs

lib-1.swf

lib-n.swf AIRAlias.js

运行 ADT 命令(没有设置 classpath)：

java -jar {AIRSDK}\lib\ADT.jar -package myApp.air myApp.xml myApp.swf

运行 ADT 命令(把 ADT.jar 包加入到 classpath 变量中)：

java com.adobe.air.ADT -package -package myApp.air myApp.xml myApp.swf

在简单的工程项目中使用Ant

这个例子展示如何用 Ant 构建 AIR 程序，一个非常简单的工程，所有文件都放在一个目录里。

注意：这个例子使用的是 AIR SDK 而不是 Flex Builder，Flex Builder 中的工具和配置文件的目录结构有所不同。

为了使用更简单些，例子中定义了一些变量属性，这一组属性定义了命令行工具的所在路径：

<property name="SDK_HOME" value="C:/FlexSDK"/>

<property name="MXMLC.JAR" value="${SDK_HOME}/lib/mxmlc.jar"/>

<property name="ADL" value="${SDK_HOME}/bin/adl.exe"/>

<property name="ADT.JAR" value="${SDK_HOME}/lib/adt.jar"/>

这一组定义了工程的一些属性，这些变量将被转换为应用程序描述文件。这里还定义了程序根目录，还有 MXMLC debug 参数为 true ：

<property name="APP_NAME" value="ExampleApplication"/>

<property name="APP_ROOT" value="."/>

<property name="MAIN_SOURCE_FILE" value="${APP_ROOT}/${APP_NAME}.mxml"/>

<property name="APP_DESCRIPTOR" value="${APP_ROOT}/${APP_NAME}-app.xml"/>

<property name="AIR_NAME" value="${APP_NAME}.air"/>

<property name="DEBUG" value="true"/>

调用编译器

要调用编译，这个例子使用一个 java 任务来运行 mxmlc.jar ：

```
<target name="compile">
    <java jar="${MXMLC.JAR}" fork="true" failonerror="true">
        <arg value="-debug=${DEBUG}"/>
        <arg value="+flexlib=${SDK_HOME}/frameworks"/>
        <arg value="+configname=air"/>
        <arg value="-file-specs=${MAIN_SOURCE_FILE}"/>
    </java>
</target>
```

当使用 Java 调用 mxmlc 时，必须指定+flexlib 参数。+configname=air 参数告诉 mxmlc 载入 AIR 配置文件来代替原来的 Flex 配置文件。

调用 ADL 来测试程序

要想用 ADL 运行程序，使用下面的任务：

```
<target name="test" depends="compile">
    <exec executable="${ADL}">
        <arg value="${APP_DESCRIPTOR}"/>
    </exec>
</target>
```

调用 ADT 打包程序

用下面的 Java 任务运行 adt.jar 工具：

```
<target name="package" depends="compile">
    <java jar="${ADT.JAR}" fork="true" failonerror="true">
        <arg value="-package"/>
        <arg value="${AIR_NAME}"/>
        <arg value="${APP_DESCRIPTOR}"/>
        <arg value="${APP_NAME}.swf"/>
        <arg value="*.png"/>
    </java>
</target>
```

如果你还想打包进更多的文件，可继续添加<arg> 元素。

在复杂的工程项目中使用Ant

因为有些程序会把所有的文件都放在一个目录中，下面的例子演示一个构建文件被用来编译，测试，打包 AIR 程序。

这个例子项目把源代码和图标都存在 src 目录，构建脚本创建了下面的工作目录：

build

存储正式版的 SWF 文件

debug

存储调试版的 SWF 文件和资源文件

release

存储最终的 AIR 包

Compiling

Mxmlc 编译器允许指定编译后的文件存放路径，通过-output 选项指定。

Testing
ADL 的第二个参数指定 AIR 程序的根目录
Packaging

```
<?xml version="1.0" ?>

<project>

    <!-- SDK properties -->
```

```
    <property name="SDK_HOME" value="C:/FlexSDK"/>
    <property name="MXMLC.JAR" value="${SDK_HOME}/lib/mxmlc.jar"/>
    <property name="ADL" value="${SDK_HOME}/bin/adl.exe"/>
    <property name="ADT.JAR" value="${SDK_HOME}/lib/adt.jar"/>

    <!-- Project properties -->
    <property name="APP_NAME" value="ExampleApplication"/>
    <property name="APP_ROOT_DIR" value="."/>
    <property name="MAIN_SOURCE_FILE" value="${APP_ROOT_DIR}/src/${APP_NAM
E}.mxml"/>
    <property name="APP_ROOT_FILE" value="${APP_NAME}.swf"/>
    <property name="APP_DESCRIPTOR" value="${APP_ROOT_DIR}/${APP_NAME}-app.
xml"/>
    <property name="AIR_NAME" value="${APP_NAME}.air"/>
    <property name="build" location="${APP_ROOT}/build"/>
    <property name="debug"  location="${APP_ROOT_DIR}/debug"/>
    <property name="release"  location="${APP_ROOT_DIR}/release"/>
    <property name="assets"  location="${APP_ROOT_DIR}/src/assets"/>

     <target name="init" depends="clean">
          <tstamp/>
          <mkdir dir="${build}"/>
       <mkdir dir="${debug}"/>
       <mkdir dir="${release}"/>
    </target>

    <target name="debugcompile" depends="init">
       <java jar="${MXMLC.JAR}" fork="true" failonerror="true">
          <arg value="-debug=true"/>
          <arg value="+flexlib=${SDK_HOME}/frameworks"/>
```

```
        <arg value="+configname=air"/>
        <arg value="-file-specs=${MAIN_SOURCE}"/>
        <arg value="-output=${debug}/${APP_ROOT_FILE}"/>
    </java>
    <copy todir="${debug}">
        <fileset dir="${assets}"/>
     </copy>
</target>

<target name="releasecompile" depends="init">
    <java jar="${MXMLC.JAR}" fork="true" failonerror="true">
        <arg value="-debug=false"/>
        <arg value="+flexlib=${SDK_HOME}/frameworks"/>
        <arg value="+configname=air"/>
        <arg value="-file-specs=${MAIN_SOURCE_FILE}"/>
        <arg value="-output=${build}/${APP_ROOT_FILE}"/>
    </java>
</target>

<target name="test" depends="debugcompile">
    <exec executable="${ADL}">
        <arg value="${APP_DESCRIPTOR}"/>
        <arg value="${debug}"/>
     </exec>
</target>

<target name="package" depends="releasecompile">
    <java jar="${ADT.JAR}" fork="true" failonerror="true">
        <arg value="-package"/>
        <arg value="${release}/${AIR_NAME}"/>
```

```
        <arg value="${APP_DESCRIPTOR}"/>
        <arg value="-C"/>
        <arg value="${build}"/>
        <arg value="${APP_ROOT_FILE}"/>
        <arg value="-C"/>
        <arg value="${assets}"/>
        <arg value="icons"/>
      </java>
    </target>

    <target name="clean" description="clean up">
      <delete dir="${build}"/>
      <delete dir="${debug}"/>
      <delete dir="${release}"/>
    </target>
</project>
```

第九章. 设置应用程序属性

应用程序描述文件的结构

应用程序描述文件 application.xml，包含了整个程序的属性，如名称，版本，版权等等。任何文件名都可作为程序描述文件，Flex Builder 当创建工程时会自动创建描述文件。当打包程序时无论使用 Flex Builder 还是 ADT，都会把描述文件重命名为 application.xml。

这里是一个描述文件的例子：

```
<?xml version="1.0" encoding="utf-8" ?>
<application appId="com.adobe.air.examples.HelloWorld" version="2.0"
    xmlns="http://ns.adobe.com/air/application/1.0.M4">
    <name>AIR Hello World</name>
    <description>
        This is the Hello World sample file from the Adobe AIR documentation.
```

```
    </description>
    <title>HelloWorld -- AIR Example</title>
    <copyright>Copyright © 2006</copyright>
    <rootContent systemChrome="none"
                 transparent="true"
                 visible="true"
                 width="640"
                 height="480">
        HelloWorld-debug.swf
    </rootContent>
    <installFolder>Adobe/Examples</installFolder>
    <icon>
        <image16x16>icons/smallIcon.png</image16x16>
        <image32x32>icons/mediumIcon.jpg</image32x32>
        <image48x48>icons/bigIcon.gif</image48x48>
        <image128x128>icons/biggestIcon.png</image128x128>
    </icon>
    <handleUpdates/>
    <fileTypes>
        <fileType>
            <name>adobe.VideoFile</name>
            <extension>avf</extension>
            <description>Adobe Video File</description>
            <contentType>application/vnd.adobe.video-file</contentType>
        </fileType>
    </fileTypes>
</application>
```

定义应用程序描述文件的属性

程序描述文件的根元素为 application 字段，其包含几个属性：

<application appId="com.adobe.air.HelloWorld" version="1.0"

xmlns="http://ns.adobe.com/air/application/1.0.M4">

appID：程序唯一标识符，该属性可由下列字符组成：

0-9

a-z

A-Z

. (点)

- (横杆)

该值必须包含 17 到 255 个字符。

version：设定程序的版本信息

xmlns：AIR 名称空间，每个版本的 AIR 其名称空间都会不同。

定义程序名称，标题，描述，版权和安装目录

name：应用程序名称，必须定义

<name>TestApp</name>

在 Windows 中，这个值会显示在程序的标题栏中

title (可选)：在 AIR 程序安装器中显示

<title>TestApp from Adobe Systems Inc.</title>

description (可选) AIR 程序安装时显示

<description>An MP3 player.</description>

copyright (可选) AIR 程序的版权信息

<copyright>Copyright © 2006 *[YourCompany, Inc.]*</copyright>

installFolder (可选) 默认安装目录的子目录

<installFolder>Acme</installFolder>

在 Windows 中，默认安装子目录为 Program Files 目录，在 Mac OS 中，为/Applications 目录。例如安装目录属性为"Acme"和程序名称为"ExampleApp"，则程序将被安装在 C:\Program Files\Acme\Example 。

使用反斜杠(/) 作为目录分隔符：

<installFolder>Acme/Power Tools</installFolder>

installFolder 属性可以包含任意 Unicode (UTF-8) 字符，除了下面的：

Character	Hex Code
various	0x00 - x1F
*	x2A
"	x22
:	x3A

>	x3C
<	x3E
?	x3F
\	x5C
|	x7C

定义 rootContent 元素

Application.xml 也指明了 rootContent 文件，因为这个文件是第一个被程序载入的。无论是 SWF 还是 HTML 文件。

rootContent 元素值是一个 URL，相对于 application.xml 文件所在路径，例如下面的 rootContent 元素中 AIRTunes.swf 文件和 application.xml 文件在同一个目录：

```
<rootContent
    systemChrome="none"
    transparent="true"
    visible="true"
    height="400"
    width="600">
      AIRTunes.swf
</rootContent>
```

rootContent 元素的属性值主要设定将被创建的窗口的属性。

systemChrome：如果设置为 standard，窗口继承操作系统窗口样式，程序没有透明，如果设置为 false，则窗口不继承操作系统窗口样式。当使用 Flex WindowedApplication 组件时，该组件将应用自己的窗口样式。

transparent：设置为 true，则窗口支持 Alpha 混合，窗口被创建后 transprent 属性将不可更改，透明的窗口需要更多内存且渲染速度慢。

重要提示：当 systemChrome="none"时你只能设置 transparent="true"。

visible：设置为 false，使窗口创建后被隐藏。

你可以需要在主窗口创建时先隐藏，等设置好位置和大小后，再通过 stage.window.visible 属性设置为 true.再显示主窗口。

height, width：主窗口的高度和宽度。

指定图标文件

Icon 属性指定一个或多个可使用的图标文件，该属性是可选的，即使没有指定，操作系统会显示默认图标。

图标路径是相对与程序的根目录，PNG, GIF, 和 JPEG 格式都支持。

```
<icon>
    <image16x16>icons/smallIcon.png</image16x16>
    <image32x32>icons/mediumIcon.jpg</image32x32>
    <image48x48>icons/bigIcon.gif</image48x48>
    <image128x128>icons/biggestIcon.png</image128x128>
</icon>
```

注意：图标不会自动被添加到 AIR 包中。

Signaling the inclusion of an update interface

一般情况，AIR 的安装和升级都是使用默认的安装对话框，但是，你可以定义自己的方法使用 AIR Updater API 来更新程序，要想这样，必须把 handleUpdates 元素添加到描述文件中：

```
<handleUpdates/>
```

注册文件类型

fileTypes 属性指定那些类型将被注册：

```
<fileTypes>
    <fileType>
        <name>adobe.VideoFile</name>
        <extension>avf</extension>
        <description>Adobe Video File</description>
        <contentType>application/vnd.adobe.video-file</contentType>
  </fileType>
    </fileTypes>
```

fileTypes 元素是可选的。

下面的例子演示如何使用 HTTPService 标签一个载入的 XML 文件，通过设置 resultFormat 属性为“object”，转换为 ActionScript 对象 Object 。

全代码：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- http://blog.flexexamples.com/2007/09/19/converting-xml-to-objects-using-the-flex-httpservice-mxml-tag/ -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
```

```
    layout="vertical"
    verticalAlign="middle"
    backgroundColor="white"
    creationComplete="serv.send();">

<mx:Script>
    <![CDATA[
        import mx.rpc.events.FaultEvent;
        import mx.rpc.events.ResultEvent;

        private function serv_result(evt:ResultEvent):void {
            var resultObj:Object = evt.result;
            /* Assign the values... */
            nameText.text = resultObj.album.name;
            img0Text.text = resultObj.album.images.image[0];
            img1Text.text = resultObj.album.images.image[1];
            img2Text.text = resultObj.album.images.image[2];
        }

        private function serv_fault(evt:FaultEvent):void {
            /* Show the error label. */
            error.text += evt.fault.faultString;
            error.visible = true;
            /* Hide the form. */
            form.visible = false;
        }
    ]]>
</mx:Script>

<mx:String id="XML_URL">album.xml</mx:String>
```

```
<mx:HTTPService id="serv"
        url="{XML_URL}"
        resultFormat="object"
        result="serv_result(event);"
        fault="serv_fault(event);" />

<mx:ApplicationControlBar dock="true">
    <mx:Label text="{XML_URL}" />
</mx:ApplicationControlBar>

<mx:Label id="error"
        color="red"
        fontSize="36"
        fontWeight="bold"
        visible="false"
        includeInLayout="{error.visible}"/>

<mx:Form id="form"
        includeInLayout="{form.visible}">
    <mx:FormItem label="resultObj.album.name:">
        <mx:Label id="nameText" />
    </mx:FormItem>
    <mx:FormItem label="resultObj.album.images.image[0]:">
        <mx:Label id="img0Text" />
    </mx:FormItem>
    <mx:FormItem label="resultObj.album.images.image[1]:">
        <mx:Label id="img1Text" />
    </mx:FormItem>
    <mx:FormItem label="resultObj.album.images.image[2]:">
```

```
        <mx:Label id="img2Text" />
      </mx:FormItem>
    </mx:Form>

</mx:Application>
```

XML 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<album>
   <name>One</name>
   <images>
      <image>image1.jpg</image>
      <image>image2.jpg</image>
      <image>image3.jpg</image>
   </images>
</album>
```

结果：

这个例子演示使用 SimpleXMLDecoder 类的 decodeXML()转换 XMLDocument 对象为 ActionScript Object 对象。

全代码:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- http://blog.flexexamples.com/2007/09/19/converting-xml-to-objects-using-the-flex-simplexmldecoder-class/ -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
      layout="vertical"
      verticalAlign="middle"
      backgroundColor="white"
      creationComplete="serv.send();">
```

```
<mx:Script>
   <![CDATA[
      import mx.rpc.events.FaultEvent;
      import mx.rpc.events.ResultEvent;
      import mx.rpc.xml.SimpleXMLDecoder;

      private function serv_result(evt:ResultEvent):void {
         /* Convert XMLNode to XMLDocument. */
         var xmlStr:String = evt.result.toString();
         var xmlDoc:XMLDocument = new XMLDocument(xmlStr);
         var decoder:SimpleXMLDecoder = new SimpleXMLDecoder(true);
         var resultObj:Object = decoder.decodeXML(xmlDoc);
         /* Assign the values... */
         nameText.text = resultObj.album.name;
         img0Text.text = resultObj.album.images.image[0];
         img1Text.text = resultObj.album.images.image[1];
         img2Text.text = resultObj.album.images.image[2];
      }

      private function serv_fault(evt:FaultEvent):void {
         // Show the error label.
         error.text += evt.fault.faultString;
         error.visible = true;
         // Hide the form.
         form.visible = false;
      }
   ]]>
</mx:Script>

<mx:String id="XML_URL">album.xml</mx:String>
```

```
<mx:HTTPService id="serv"
      url="{XML_URL}"
      resultFormat="xml"
      result="serv_result(event);"
      fault="serv_fault(event);" />

<mx:ApplicationControlBar dock="true">
   <mx:Label text="{XML_URL}" />
</mx:ApplicationControlBar>

<mx:Label id="error"
      color="red"
      fontSize="36"
      fontWeight="bold"
      visible="false"
      includeInLayout="{error.visible}"/>

<mx:Form id="form"
      includeInLayout="{form.visible}">
   <mx:FormItem label="resultObj.album.name:">
      <mx:Label id="nameText" />
   </mx:FormItem>
   <mx:FormItem label="resultObj.album.images.image[0]:">
      <mx:Label id="img0Text" />
   </mx:FormItem>
   <mx:FormItem label="resultObj.album.images.image[1]:">
      <mx:Label id="img1Text" />
   </mx:FormItem>
   <mx:FormItem label="resultObj.album.images.image[2]:">
```

```
        <mx:Label id="img2Text" />
      </mx:FormItem>
    </mx:Form>

</mx:Application>
```

XML 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<album>
   <name>One</name>
   <images>
      <image>image1.jpg</image>
      <image>image2.jpg</image>
      <image>image3.jpg</image>
   </images>
</album>
```

结果：

下面的例子使用 DataGrid 组件的 dragEnabled, dropEnabled,和 dragMoveEnabled 属性实现表格之间的行数据移动

全代码：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- http://blog.flexexamples.com/2007/09/19/dragging-rows-between-two-different-flex-datagrid-controls/ -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
      layout="horizontal"
      verticalAlign="middle"
      backgroundColor="white">
```

```
<mx:Array id="arr">
   <mx:Object colA="Item A.0" colB="Item B.0" colC="Item C.0" />
   <mx:Object colA="Item A.1" colB="Item B.1" colC="Item C.1" />
   <mx:Object colA="Item A.2" colB="Item B.2" colC="Item C.2" />
   <mx:Object colA="Item A.3" colB="Item B.3" colC="Item C.3" />
   <mx:Object colA="Item A.4" colB="Item B.4" colC="Item C.4" />
   <mx:Object colA="Item A.5" colB="Item B.5" colC="Item C.5" />
   <mx:Object colA="Item A.6" colB="Item B.6" colC="Item C.6" />
   <mx:Object colA="Item A.7" colB="Item B.7" colC="Item C.7" />
   <mx:Object colA="Item A.8" colB="Item B.8" colC="Item C.8" />
   <mx:Object colA="Item A.9" colB="Item B.9" colC="Item C.9" />
</mx:Array>

<mx:ApplicationControlBar dock="true">
   <mx:Form>
      <mx:FormItem label="DataGrid #1:"
           direction="horizontal">
         <mx:CheckBox id="dg1_dragEnabled"
              label="dragEnabled" />
         <mx:CheckBox id="dg1_dropEnabled"
              label="dropEnabled" />
         <mx:CheckBox id="dg1_dragMoveEnabled"
              label="dragMoveEnabled" />
      </mx:FormItem>
      <mx:FormItem label="DataGrid #2:"
           direction="horizontal">
         <mx:CheckBox id="dg2_dragEnabled"
              label="dragEnabled" />
         <mx:CheckBox id="dg2_dropEnabled"
              label="dropEnabled" />
```

```
            <mx:CheckBox id="dg2_dragMoveEnabled"
                label="dragMoveEnabled" />
        </mx:FormItem>
    </mx:Form>
</mx:ApplicationControlBar>

<mx:VBox width="50%">
    <mx:Label text="DataGrid #1" />
    <mx:DataGrid id="dataGrid1"
        width="100%"
        rowHeight="22"
        dataProvider="{arr}"
        dragEnabled="{dg1_dragEnabled.selected}"
        dragMoveEnabled="{dg1_dragMoveEnabled.selected}"
        dropEnabled="{dg1_dropEnabled.selected}"
        verticalScrollPolicy="on">
        <mx:columns>
            <mx:DataGridColumn dataField="colA"
                headerText="Column A" />
            <mx:DataGridColumn dataField="colB"
                headerText="Column B" />
            <mx:DataGridColumn dataField="colC"
                headerText="Column C" />
        </mx:columns>
    </mx:DataGrid>
    <mx:Label text="{dataGrid1.dataProvider.length} items" />
</mx:VBox>

<mx:VBox width="50%">
    <mx:Label text="DataGrid #2" />
```

```
    <mx:DataGrid id="dataGrid2"
        width="100%"
        rowHeight="22"
        dataProvider="[]"
        dragEnabled="{dg2_dragEnabled.selected}"
        dragMoveEnabled="{dg2_dragMoveEnabled.selected}"
        dropEnabled="{dg2_dropEnabled.selected}"
        verticalScrollPolicy="on">
      <mx:columns>
        <mx:DataGridColumn dataField="colA"
            headerText="Column A" />
        <mx:DataGridColumn dataField="colB"
            headerText="Column B" />
        <mx:DataGridColumn dataField="colC"
            headerText="Column C" />
      </mx:columns>
    </mx:DataGrid>
    <mx:Label text="{dataGrid2.dataProvider.length} items" />
  </mx:VBox>

</mx:Application>
```

Adobe AIR新增功能

这一节给 AIR 新增的功能做一个概览。

- ·新的运行时类
- ·运行时类的新功能
- ·新的监控伺服类

新的运行时类（runtime classes）

下面都是 Adobe AIR 新增的运行时类，这些功能不能用于浏览器中运行的 SWF：

类	包
Clipboard	flash.desktop
ClipboardFormats	flash.desktop
CompressionAlgorithm	flash.utils
ClipboardTransferMode	flash.desktop
DockIcon	flash.display
DragActions	flash.desktop
DragManager	flash.desktop
DragOptions	flash.desktop
DRMAuthenticateEvent	flash.events
DRMStatusEvent	flash.events
EncryptedLocalStore	flash.filesystem
File	flash.filesystem
FileListEvent	flash.events
FileMode	flash.filesystem
FileStream	flash.filesystem
HTMLControl	flash.html
HTMLHistoryItem	flash.html
HTMLHost	flash.html
HTMLPDFCapability	flash.html
HTMLUncaughtJavaScriptExceptionEvent	flash.html
HTMLWindowCreateOptions	flash.html
Icon	flash.desktop
InteractiveIcon	flash.display
InvokeEvent	flash.events
JavaScriptFunction	flash.html
JavaScriptObject	flash.html
NativeDragEvent	flash.events
NativeMenu	flash.display
NativeMenuItem	flash.display
NativeWindow	flash.display
NativeWindowBoundsEvent	flash.events
NativeWindowDisplayState	flash.display
NativeWindowDisplayStateEvent	flash.events
NativeWindowErrorEvent	flash.events
NativeWindowIcon	flash.display
NativeWindowInitOptions	flash.display
NativeWindowResize	flash.display

NativeWindowSystemChrome	flash.display
NativeWindowType	flash.display
NotificationType	flash.display
OutputProgressEvent	flash.events
Screen	flash.display
Shell	flash.system
SQLCollationType	flash.data
SQLColumnNameStyle	flash.data
SQLColumnSchema	flash.data
SQLConnection	flash.data
SQLError	flash.errors
SQLErrorCode	flash.errors
SQLErrorEvent	flash.events
SQLErrorOperation	flash.errors
SQLEvent	flash.events
SQLIndexSchema	flash.data
SQLResult	flash.data
SQLSchema	flash.data
SQLSchemaResult	flash.data
SQLStatement	flash.data
SQLTableSchema	flash.data
SQLTransactionLockType	flash.data
SQLTriggerSchema	flash.data
SQLUpdateEvent	flash.events
SQLViewSchema	flash.data
SystemTrayIcon	flash.display
Updater	flash.system
URLRequestDefaults	flash.net
XMLSignatureValidator	flash.utils
TransferableTransferMode	flash.desktop
URLRequestDefaults	flash.net
Updater	flash.system

大多数类只能在 AIR 程序安全沙箱中可用，不过下面的这些类也可以用在其他沙箱之上：

- Door
- URLRequest.

运行时类新增功能

下面的类能用在浏览器上，但是 AIR 下提供了额外的属性和方法：

类	属性和方法
HTTPStatusEvent	HTTP_RESPONSE_STATUS responseURL responseHeaders
URLRequest	followRedirects manageCookies shouldAuthenticate shouldCacheResponse userAgent userCache setLoginCredentials()
URLStream	httpResponseStatus event
Stage	nativeWindow
Security	APPLICATION

大多数这些属性和方法都在 AIR 程序安全沙漏中使用，不过 URLRequest 类也可以用在其他安全沙箱中。

ByteArray.compress() 和 ByteArray.uncompress()方法包含一个新的算法参数，允许你选择压缩或 zlib 压缩。

新的 Flex 组件

下面的 Flex 组件在 AIR 程序开发时可用：

FileEvent

FileSystemComboBox
FileSystemDataGrid
FileSystemEnumerationMode
FileSystemHistoryButton
FileSystemList
FileSystemSizeDisplayMode
FileSystemTree
HTML
WindowedApplication

监控伺服类

Air.net 包包含一些网络监测类，该包也只能在 Adobe AIR 中使用，需要引入 ServiceMonitor.swc 文件。

该包包含下列类：

- ServiceMonitor
- SocketMonitor
- URLMonitor

第十章. 窗体（Windows）和菜单

AIR 窗体基础

窗体 API 包含下列类：

Package	Classes
flash.display	NativeWindow, NativeWindowInitOptions 下面的类中定义了些 Window 字符常量：NativeWindowDisplayState, NativeWindowResize, NativeWindowSystemChrome
flash.events	NativeWindowBoundsEvent, NativeWindowDisplayStateEvent, NativeWindowErrorEvent
flash.system	NativeWindowCapabilities

AIR 提供了易用的，跨平台的窗体 API，使用 Flash, Flex, 和 HTML 编程技术创建本地操作系统窗体。

用 AIR，你可以很自由的创建应用程序的主题样式，创建的窗体看起来和标准桌面程序一样，在不同的操作系统上就有不同的系统窗体风格，或者使用皮肤，扩展窗体主题风格。你甚至可以用矢量图或位图绘制窗体。

AIR 支持两种不同类型的 APIs 用于创建窗体：面向 Flash 的 NativeWindow 类和面向 HTML 的 JavaScript Window 类。通过 Flash stage 和显示列表可直接用 NativeWindow 创建窗体。要添加可视化对象到 NativeWindow 上，需要添加对象到 window's stage 的显示列表上。使用 HTML，CSS 和 JavaScript 创建窗体来显示内容，如果要添加可视化对象到 HTML 窗体上，需要用 DOM 添加对象。

以前的项目中的窗体都是通过 AIR 项目的描述文件自动创建的，因为已经指定了 rootContent 元素属性值。如果根内容是一个 SWF 文件，则 NativeWindow 被创建，SWF 被载入到窗体中。如果根内容为 HTML 文件，则 HTML 窗体被创建，HTML 页被载入。

Native windows 使用基于事件驱动的编程模型。改变任何属性或调用窗体的方法都将影响显示或组件的行为。例如当系统按钮最大化按钮被点击时，下列事件将被触发：

1. 用户点击了最大化按钮。
2. 窗体发出一个 displayStateChanging 事件。
3. 如果没有注册监听器取消事件，则窗体被最大化。
4. displayStateChange 事件通知监听器。

另外，相关联的事件也会被触发：

1. 如果窗体左上角坐标因为最大化而改变则发出 move 事件。
2. 如果窗体大小因最大化发生变化则发出 resize 事件。

类似的事件还有 minimizing, restoring, closing, moving, resizing window.

本地窗体的样式

窗体的基本外观和样式取决于三个属性：type, systemChrome, 和 transparent。Type 属性设置窗体的功能。systemChrome 属性设置窗体是否遵循操作系统窗体样式。Transparent 属性设置是否支持 alpha 混合。这些属性都是设置在将被创建的窗体的 NativeWindowInitOptions 对象上且不能被改变。自动创建的初始窗体是根据描述文件的这些属性值进行设置的，且 type 属性不能在描述文件中设置，其值都是 normal。

这些属性的设置有些是不兼容的，如当 transparent 设置为 true 时或 type 为 lightweight 时 systemChrome 不能设置为 standard 。

窗体类型

Type 属性决定创建何种类型的窗体，且只有在创建新的 NativeWindow.时可设置，AIR 支持下列类型的窗体：

Normal

典型的窗体，Normal 窗体具有操作系统普通窗体的功能。

Utility

一个面板窗体，Utility 窗体是简化的窗体，且不显示在系统任务栏中。

Lightweight

Lightweight 窗体也不显示在系统任务栏中，另外它没有系统菜单。它很适合做提示水泡窗体，当 type 设置为 lightweight 时，systemChrome 必须设置为"none".

注意：在 Beta 版本中，模态窗体类型还不支持

Window chrome

Window chrome 是一组操作窗体的控件，作为 AIR 程序 window chrome 有一下几种：

System chrome

System chrome 显示的窗体具有操作系统窗体一样的一组控件和样式。使用 system chrome 是 AIR 窗体和普通桌面程序具有一样的样式风格。System chrome 是由系统管理的，你的程序不能直接访问这些控件，但可以处理这些控件发出的事件，如果窗体被设置为 system chrome ，则 transparency 属性必须为 false 。

Flex chrome

当使用 Flex WindowedApplication 和 Window 组件时，窗体样式由 Flex framework 提供，如果 Flex 组件在 system chrome 下使用，则 Flex chrome 不会被显示。

Custom chrome

如果你创建的窗体没有 system chrome 且不使用 Flex mx:WindowedApplication 组件，你必须添加自己的控件处理用户和窗体的交互。这里你可以制作透明的，不规则的窗体。

窗体透明

要允许窗体与桌面环境进行 alpha 混合，需要设置窗体的 transparent 属性为 true。Transparent 属性必须在窗体创建之前设置，且不能改变。该属性在 NativeWindowInitOptions 对象上。

透明窗体在以下方面很有用：

1. 程序边框为不规则图形
2. 程序必须”淡出“或出现为不可视

当窗体设置为 system chrome. 时透明不可用。

MXML 程序窗体的透明性

如果 Flex 程序使用 CSS 样式设置背景色或颜色，则窗体不会透明。

```
Application
{
  background-image:"";
  background-color:"";
}
```

可视化窗体目录

下面的表格说明不同的属性值影响窗体的视觉效果：

Type	System Chrome	Transparent	Mac OS X	Microsoft Windows

Standard	normal	Not supported		
Utility	normal	Not supported		
Any	none	false		
Any	none	true		
Any	none	true	mx:WindowedApplication	

Beta 版本的限制

下面的特性在 beta 版本中还不支持：

1. 窗体 API 还不支持 Toolbars|OS X Toolbar 或 OS X Proxy Icon.
2. 系统托盘图标
3. 程序控制应用程序图标
4. 桌面屏幕信息
5. 本地窗体菜单

创建窗体

AIR 主要提供了下列方法用于创建程序窗体：

- ·AIR 自动为每个程序创建第一个窗体。这个窗体根据应用程序描述文件设置进行初始化。如果 root 上下文已经在描述文件里定义，那么就可以通过 Stage.window 属性和 NativeWindow API 来访问 window 实例的属性和方法。另外，SWF 文件的主类必须继承 Sprite 或 Sprite 的子类(WindowedApplication 和 Application 组件都是 Sprite 之类）如果 root 上下文是一个 HTML 文件，这可通过 JavaScript Window 对象访问 window 的属性和方法。
- ·可以创建 NativeWindow 类实例，通过 NativeWindowInitOptions 对象实例作为 NativeWindows 构造函数的参数进行初始化。可直接通过该对象来访问属性和方法。
- ·可以使用 HTML 组件的 createRootWindow() 方法创建一个窗体用于显示 HTML 内容。

- ·还可以使用 JavaScript Window.open() 方法通过 javascript 打开一个新窗体。JavaScript 创建的窗体只能通过 javascript 访问其属性和方法，且该窗体只能显示 HTML 内容。

创建一个新的本地窗体（NativeWindow）

要创建一个新的 NativeWindow，可先创建一个 NativeWindowInitOptions 对象并传递给 NativeWindow 构造函数。

```
var options:NativeWindowInitOptions = new NativeWindowInitOptions();
options.systemChrome = NativeWindowSystemChrome.STANDARD;
options.transparent = false;
var newWin:NativeWindow = new NativeWindow(false, options);
```

NativeWindow 构造函数第一个参数指定是否创建时立即显示该窗体，为了避免显示出如设置大小，位置和内容时窗体所处的中间状态，当你完成窗体初始化后，设置 NativeWindow.visible 属性为 true。第二个参数为 NativeWindowInitOptions 对象，该对象用于设置窗体的属性，一旦窗体创建后将不能更改。

注意：设置 systemChrome="standard" 和 transparent="true" 这种组合不支持。

一旦窗体被创建，你可以用 stage 属性和 Flash 显示列表初始化一些属性并载入内容到窗体之上。

注意：要确定操作系统窗体的最大化和最小化大小，可通过 NativeWindowCapabilities 类得到：

```
var maxOSSize:Point = NativeWindowsCapabilites.windowMaxSize;
var minOSSize:Point = NativeWindowsCapabilites.windowMinSize;
```

往窗体中添加内容

要添加内容到本地窗体上，可通过窗体的 stage 来添加可视化对象。你可以动态创建可视化对象或者通过 flash.display.Loader 类载入现成的内容。对于 HTML 窗体，可通过 location 属性来改变加载的内容或插入 HTML 内容到 DOM。

当你载入包含 JavaScript 的 SWF 或 HTML 内容时必须要考虑到 AIR 安全模型。任何安全沙箱中的程序，无论是安装的内容或通过 url 载入的资源都有能力访问到 AIR APIs，任何从沙箱外部载入的内容在访问受安全约束的 AIR APIs 和跨脚本内容时将受到限制。应用程序安全沙箱外部的 JavaScript 内容将不能使用 JavaScript 窗体对象的 nativeWindow 或 htmlControl 属性。

要想允许安全跨脚本访问，你可以使用 flash.system.Door API 创建一个严格受限的通讯网关，它提供有限的接口用于程序内容和非程序内容之间的访问。

载入一个 SWF 或 图片

可以使用 flash.display.Loader 类载入 flash 或图片到本地窗体的显示列表上。

```
package {

    import flash.display.Sprite;

    import flash.events.Event;

    import flash.net.URLRequest;

    import flash.display.Loader;


    public class LoadedSWF extends Sprite

    {

        public function LoadedSWF(){

            var loader:Loader = new Loader();

            loader.load(new URLRequest("visual.swf"));

            loader.contentLoaderInfo.addEventListener(Event.COMPLETE,loadFlash);

        }


```

```
        private function loadFlash(event:Event):void{

            addChild(event.target.loader);

        }

    }

}
```

要在 HTML 窗体中载入可视化的 Flash 内容，必须显示在 HTML 内容的最顶层，或在透明的 HTML 内容之下。且大小和位置计算独立于 HTML 内容。

要想在基于 HTML 的程序中载入包含库代码的 SWF 文件，最简单的办法就是使用 script 标签，也可以直接使用 flash.display.Loader API 。

载入 HTML 内容到本地窗体（NativeWindow）上

要载入 HTML 内容到本地窗体上，必须在窗体的 stage 上添加一个 HTMLControl 控件，然后载入 HTML 内容到 HTML 控件上。

```
//newWindow is a NativeWindow instance

var htmlView:HTMLControl = new HTMLControl();

html.width = newWindow.width;

html.height = newWindow.height;



//set the stage so display objects are added to the top-left and not scaled
```

```
newWindow.stage.align = "TL";

newWindow.stage.scaleMode = "noScale";

newWindow.stage.addChild( htmlView );



//urlString is the URL of the HTML page to load

htmlView.load( new URLRequest(urlString) );
```

要载入 HTML 页面到 HTML 窗体上，使用 JavaScript 方法如 window.open 。

要载入 HTML 页面到 Flex 程序上，使用 Flex HTML 组件。

载入 Flash 内容到 HTML 页面上

在这个 Beta 版本中，将不支持直接在 HTML 页面中嵌入 Flash 内容。页面中的任何 Flash 对象都将被显示为空白区域，但是使用 AIR APIs 载入或创建的 Flash 内容都将作为 HTML 层的形式存在。

在 HTML 窗体上层添加 Flash 内容

因为 HTML 窗体包含在 NativeWindow 实例之内，可以在 HTML 层的上层或下层中添加 Flash 可视化对象。

要添加可视化对象到 HTML 层之上，可通过 window.nativeWindow.stage 属性的 addChild() 方法，addChild()方法将把内容放在任何现成内容之上。

要添加可视化对象到 HTML 层之下，使用 window.nativeWindow.stages 属性的 addChildAt()方法，传递一个 0 值作为索引参数，这将导致其他层都向上移动，最后把新加入的对象放在底部。要想让 HTML 层以下的内容可见，必须设置 window.htmlControl 对象的 paintsDefaultBackground 属性为 false。

下面的例子演示如何添加一个 flash 对象，该例子创建了两个图形对象，一个添加在 HTML 层下面，一个在上面。

```
<html>

<head>

<title>Bouncers</title>

<script src="AIRAliases.js" type="text/javascript"></script>

<script language="JavaScript" type="text/javascript">

air.Shape = window.runtime.flash.display.Shape;



function Bouncer(radius, color){

    this.radius = radius;

    this.color = color;



    //velocity

    this.vX = -1.3;

    this.vY = -1;
```

```
    //Create a Shape object and draw a circle with its graphics property
    this.shape = new air.Shape();
    this.shape.graphics.lineStyle(1,0);
    this.shape.graphics.beginFill(this.color,.9);
    this.shape.graphics.drawCircle(0,0,this.radius);
    this.shape.graphics.endFill();

    //Set the starting position
    this.shape.x = 100;
    this.shape.y = 100;


    //Moves the sprite by adding (vX,vY) to the current position
    this.update = function(){
```

```
        this.shape.x += this.vX;

        this.shape.y += this.vY;


        //Keep the sprite within the window

        if( this.shape.x - this.radius < 0){

            this.vX = -this.vX;

        }

        if( this.shape.y - this.radius < 0){

            this.vY = -this.vY;

        }

        if( this.shape.x  + this.radius > window.nativeWindow.stage.stageWidth){

            this.vX = -this.vX;

        }

        if( this.shape.y  + this.radius > window.nativeWindow.stage.stageHeight){

            this.vY = -this.vY;
```

```
    }

};

}


function init(){

    //turn off the default HTML background

    window.htmlControl.paintsDefaultBackground = false;

    var bottom = new Bouncer(60,0xff2233);

    var top = new Bouncer(30,0x2441ff);


    //listen for the enterFrame event

    window.htmlControl.addEventListener("enterFrame",function(evt){

        bottom.update();

        top.update();
```

```
    });


    //add the bouncing shapes to the window stage

    window.nativeWindow.stage.addChildAt(bottom.shape,0);

    window.nativeWindow.stage.addChild(top.shape);

}

</script>

<body onload="init();">

<h1>de Finibus Bonorum et Malorum</h1>

<p>Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium

doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis

et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia

voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui

ratione voluptatemsequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia

dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora
```

```
incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam,

quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea

commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit

esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas

nulla pariatur?</p>

<p style="background-color:#FFFF00; color:#660000;">This paragraph has a background color.
</p>

<p>At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis

praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias

excepturi sint occaecati cupiditate non provident, similique sunt in culpa qui

officia deserunt mollitia animi, id est laborum et dolorum fuga. Et harum quidem

rerum facilis est et expedita distinctio. Nam libero tempore, cum soluta nobis est

eligendi optio cumque nihil impedit quo minus id quod maxime placeat facere possimus,

omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam

et aut officiis debitis aut rerum necessitatibus saepe eveniet ut et voluptates

repudiandae sint et molestiae non recusandae. Itaque earum rerum hic tenetur a
```

```
sapiente delectus, ut aut reiciendis voluptatibus maiores alias consequatur aut

perferendis doloribus asperiores repellat.</p>

</body>

</html>
```

示例：用 ActionScript 创建窗体

下面的例子演示如何创建新窗体：

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

<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"

  layout="absolute" applicationComplete="createNativeWindow();">

  <mx:Script>

    <![CDATA[

      public function createNativeWindow():void{

        //create the init options

        var options:NativeWindowInitOptions =

          new NativeWindowInitOptions();
```

```
options.transparent = false;

options.systemChrome = NativeWindowSystemChrome.STANDARD;

options.type = NativeWindowType.NORMAL;



//create the window

var newWindow:NativeWindow = new NativeWindow(false,options);

newWindow.title = "A title";

newWindow.width = 600;

newWindow.height = 400;



//add a sprite to the window

newWindow.stage.align = StageAlign.TOP_LEFT;

newWindow.stage.scaleMode = StageScaleMode.NO_SCALE;
```

```
            //show the new window
            newWindow.visible = true;
        }
    ]]>
    </mx:Script>
</mx:WindowedApplication>
```

控制窗体

这一章节讨论如何使用 NativeWindow 类的属性和方法控制应用程序窗体的外观和行为。

得到 NativeWindow 实例

要想操作窗体，必须先得到窗体实例，可在下面这些地方得到窗体实例：

窗体构造函数

也就是新建 NativeWindow 所用的构造函数.

窗体的 stage

也就是 stage.nativeWindow

任何可视化对象的 stage

也就是 myDisplayObject.stage.nativeWindow.

窗体事件

Event 对象的 target 属性指向窗体引用

HTMLControl 或 HTML 窗体的全局属性 nativeWindow

也就是 window.nativeWindow.

Shell 对象

Shell.shell.activeWindow 指向应用程序的活动窗体（激活状态，如果没有任何窗体处于激活状态则返回 null）。Shell.shell.openedWindows 数组包含应用程序中所有未被打开的窗体。

因为 Flex Application, WindowedApplication, 和Window 对象都是可视化对象，我们可以很容易通过stage属性得到窗体引用：

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

<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" applicationC
omplete="init();">

    <mx:Script>

        <![CDATA[

            import flash.display.NativeWindow;

            public function init():void{

                var appWindow:NativeWindow = this.stage.nativeWindow;

                //set window properties

                appWindow.visible = true;

            }

        ]]>

    </mx:Script>

</WindowedApplication>
```

激活，显示，隐藏窗体

调用 NativeWindow.activate()方法激活一个窗体，激活窗体后将使窗体显示在最顶层，开始获得键盘和鼠标焦点，如果有必要可恢复 window 的 visible 属性或设置 visible=true，激活一个窗体并不会改变其他窗体的顺序。

要想显示没有激活的隐藏窗体，可设置其 visible 属性为 true，这将使窗体显示在最顶层，但是由于未激活，不能接收键盘和鼠标反应。

要隐藏窗体，可设置 NativeWindow.visible 属性为 false。隐藏一个窗体将取消窗体的显示以及相关的任何栏图标。

最大化，最小化和还原窗体

使用 NativeWindow.maximize()方法使窗体最大化显示

```
myWindow.maximize();
```

使用 NativeWindow.minimize()方法使窗体最小化显示

```
myWindow.minimize();
```

还原一个窗体指恢复到最大化和最小化之前的状体。

```
myWindow.restore();
```

改变窗体显示顺序

AIR 为窗体提供两组显示顺序，由 alwaysInFront 属性进行控制。通过设置 alwaysInFront=false，意思是把窗体放在正规的组群众，大多数窗体都放在这里。而第二个组群所放的窗体总是显示在正规组群窗体的上层。如果没有这个特殊组群的话，改变窗体顺序将会显得非常混乱，下列情况都将用到这个窗体组群：

- 临时的弹出式窗体，比如提示框，弹出式列表，自定义菜单或组合框，因为这些窗体用完即关闭。
- 屏幕保护
- 一些急需的错误信息框
- 生命周期很短的窗体
- 游戏或视频的全屏窗体

NativeWindow 类提供下列方法用于设置窗体之间的显示顺序：

alwaysInFront 属性

指定使用哪个窗体组，一般情况下 alwaysInFront=false ，如果设为 true，这表示所有窗体都处于顶层位置（只是没有激活），如果在设为 false，表示把窗体放入正规组的顶层（但是仍在顶层组的后面）。设置该值并不会改变窗体的顺序。

orderToFront()

设置该窗体为组群中最前

orderInFrontOf()

设置该窗体排在某窗体之前

orderToBack()

设置该窗体为组群中最后

orderBehind()

设置该窗体排在某窗体之后

注意：如果该窗体被隐藏或最小化，调用这些方法将无效。

关闭窗体

使用 NativeWindow.close 方法关闭窗体

关闭一个窗体将卸载该窗体所包含的内容，不过如果还有其他对象引用这些内容，将不会被销毁。NativeWindow.close()方法的执行是异步的，应用程序仍然在关闭窗体过程中继续运行，当关闭完成时，该 close 方法将触发一个 close 事件。窗体被关闭后将无法访问其大部分属性和方法，否则会抛出 IllegalOperationError 异常，关闭的窗体将不能再次打开，可通过 closed 属性检测窗体是否已被关闭。要简单的隐藏窗体，只要设置其 visible 属性为 false 就可以了。

如果 Shell.autoExit 属性为 true（默认设置），当应用程序的所有窗体都关闭后，程序也将停止运行。

允许撤销对窗体的操作

当窗体使用操作系统窗体风格，用户与窗体的交互可通过一些事件进行取消，比如说，单用户点击一个窗体上系统菜单的关闭按钮，这 closing 事件将被触发，任何注册的监听器调用 preventDefault()方法可以取消关闭窗体。

如果窗体不使用系统窗体风格，这需要手动出发相应的事件。如果你调用一个方法要关闭窗体或改变窗体大小或设置 bounds 属性，这些改变将不能被取消，除非在窗体发生变化之前通知应用程序，应用程序逻辑通过 dispatchEvent()发出相关的事件。

看下面的例子，下面的逻辑实现了可取消的窗体关闭按钮。：

```
public function onCloseCommand(event:MouseEvent):void{

    var closingEvent:Event = new Event(Event.CLOSING,true,true);

    dispatchEvent(closing);

     if(!closingEvent.isDefaultPrevented()){
```

```
        win.close();

    }

}
```

注意：如果事件的 preventDefault() 方法被监听器调用，则 dispatchEvent()方法返回 false，但是也有其他原因可能返回 false，所以最好的办法是使用 isDefaultPrevented()方法检测是否已经取消操作。

例子：最小化，最大化，还原和关闭窗体

下面的这段代码演示窗体的 maximize()，minimize()，restore()，和 close() 方法。

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

<mx:WindowedApplication

    xmlns:mx="http://www.adobe.com/2006/mxml"

    layout="vertical">

    <mx:Script>

    <![CDATA[

    public function minimizeWindow():void

    {

        this.stage.nativeWindow.minimize();
```

```
}

public function maximizeWindow():void

{

    this.stage.nativeWindow.maximize();

}

public function restoreWindow():void

{

    this.stage.nativeWindow.restore();

}

public function closeWindow():void

{

    this.stage.nativeWindow.close();

}

]]>

</mx:Script>
```

```
    <mx:VBox>

        <mx:Button label="Minimize" click="minimizeWindow()"/>

        <mx:Button label="Restore" click="restoreWindow()"/>

        <mx:Button label="Maximize" click="maximizeWindow()"/>

        <mx:Button label="Close" click="closeWindow()"/>

    </mx:VBox>

</mx:WindowedApplication>
```

例子：改变大小和移动窗体

使用 NativeWindow.startResize()方法改变窗体大小，该方法实际上是触发系统控制功能来改变窗体大小。当该方法在 mouseDown 事件里调用时，大小改变是由鼠标确定的，当系统接收到 mouseUp 事件则停止大小调整。

如果不改变大小只移动窗体，可使用 NativeWindow.startMove()方法，和 startResize()方法一样，当 startMove()方法 mouseDown 事件里调用时，窗体移动过程是由鼠标确定的，直到系统接收到鼠标的 mouseUp 事件。

下面的这个例子演示如何初始化改变窗体大小和移动窗体操作：

```
package

{

    import flash.display.Sprite;

    import flash.events.MouseEvent;
```

```
import flash.display.NativeWindowResize;

public class NativeWindowResizeExample extends Sprite

{

    public function NativeWindowResizeExample():void

    {

        // Fills a background area.

        this.graphics.beginFill(0xFFFFFF);

        this.graphics.drawRect(0, 0, 400, 300);

        this.graphics.endFill();

        // Creates a square area where a mouse down will trigger a resize.

        var resizeHandle:Sprite = createSprite(0xCCCCCC, 20, this.width - 20, this.height - 2
0);

        resizeHandle.addEventListener(MouseEvent.MOUSE_DOWN, onStartResize);

        // Creates a square area where a mouse down will trigger a move.

        var moveHandle:Sprite = createSprite(0xCCCCCC, 20, this.width - 20, 0);
```

```
            moveHandle.addEventListener(MouseEvent.MOUSE_DOWN, onStartMove);

        }

        public function createSprite(color:int, size:int, x:int, y:int):Sprite

        {

            var s:Sprite = new Sprite();

            s.graphics.beginFill(color);

            s.graphics.drawRect(0, 0, size, size);

            s.graphics.endFill();

            s.x = x;

            s.y = y;

            this.addChild(s);

            return s;

        }

        public function onStartResize(event:MouseEvent):void

        {
```

```
            this.stage.nativeWindow.startResize(NativeWindowResize.BOTTOM_RIGHT);

        }

        public function onStartMove(event:MouseEvent):void

        {

            this.stage.nativeWindow.startMove();

        }

    }

}
```

监听窗体事件

如要监听窗体发出的事件，可通过窗体注册一个监听器，例如，要监听 closing 事件，用下面的代码注册：

myWindow.addEventListener(Event.CLOSING, onClosingEvent);

当事件发出时，窗体引用的 target 属性发出该事件。

大多数窗体事件都有两种消息，第一个消息是该窗体的变化快要临近（是可以取消的），而另一个消息表示变化已经发生。例如，当用户点击关闭按钮，则 closing 事件消息被触发，如果没有监听器取消该事件，则窗体被关闭。

flash.events.Event 类的相关事件：

ACTIVATE
DEACTIVATE
CLOSING
CLOSE

NativeWindowBoundsEvent:

使用 beforeBounds 和 afterBounds 属性来检测窗体边界是否即将改变或已经完成改变。

MOVING

MOVE

RESIZING

RESIZE

NativeWindowDisplayStateEvent:

使用 beforeDisplayState 和 afterDisplayState 属性检测窗体改变状态是即将改变还是已经完成改变。

DISPLAY_STATE_CHANGING

DISPLAY_STATE_CHANGE

屏幕

根据 AIR screen API 可以获得系统桌面显示屏幕的信息。

屏幕简介

screen API 包含一个简单类，Screen，获得系统屏幕信息和屏幕的详细描述。

计算机系统可能有多个监视器或显示设备，这样对应的多个桌面屏幕排列在虚拟空间上。AIR Screen 类提供了关于屏幕信息，如果有多个监视器映射到同一个屏幕上，那只有一个屏幕可显示，如果屏幕的尺寸大于监视器显示范围，没有办法确定是哪一部分处于可视状态。

一个屏幕表示一个独立的桌面显示区域，被描述为虚拟桌面的一个矩形区域，屏幕左上角为初始坐标，单位为像素。

上面的屏幕排列中，虚拟桌面上有两个屏幕，主屏幕（#1）的左上角坐标总是（0，0），如果屏幕排列设置屏幕#2 作为主屏幕，则屏幕#1 的坐标将为负坐标，一般指屏幕的可使用边界不包括菜单栏，任务栏。

枚举屏幕

通过下列屏幕方法和属性枚举虚拟桌面上的屏幕：

Screen.screens

数组对象，表示可用的屏幕，注意数组的元素顺序不是有效的。

Screen.mainScreen

表示代表主屏幕的屏幕对象，在 Mac OS X 系统中，主屏幕为显示菜单栏的所在屏幕，在 Windows 中为系统指定的主屏幕。

Screen.getScreensForRectangle()

通过指定的区域获得屏幕对象数组，该矩形区域作为参数传递给该方法，如果没有屏幕在范围内则返回空数组。

示例：在屏幕中移动窗体

这个例子使用 screen API 通过方向键在多个屏幕中移动窗体。

```
package {

    import flash.display.Sprite;

    import flash.display.Screen;

    import flash.events.KeyboardEvent;

    import flash.ui.Keyboard;

    import flash.display.StageAlign;

    import flash.display.StageScaleMode;



    public class ScreenExample extends Sprite

    {

        public function ScreenExample()

        {

            stage.align = StageAlign.TOP_LEFT;

            stage.scaleMode = StageScaleMode.NO_SCALE;
```

```
            stage.addEventListener(KeyboardEvent.KEY_DOWN,onKey);

        }



        private function onKey(event:KeyboardEvent):void{

            if(Screen.screens.length > 1){

                switch(event.keyCode){

                    case Keyboard.LEFT :

                        moveLeft();

                        break;

                    case Keyboard.RIGHT :

                        moveRight();

                        break;

                    case Keyboard.UP :

                        moveUp();

                        break;
```

```
            case Keyboard.DOWN :

                moveDown();

                break;

            }

        }

    }


    private function moveLeft():void{

        var currentScreen = getCurrentScreen();

        var left:Array = Screen.screens;

        left.sort(sortHorizontal);

        for(var i:int = 0; i < left.length - 1; i++){

            if(left[i].bounds.left < stage.nativeWindow.bounds.left){

                stage.nativeWindow.x +=

                    left[i].bounds.left - currentScreen.bounds.left;
```

```
                stage.nativeWindow.y += left[i].bounds.top - currentScreen.bounds.top;

            }

        }

    }


    private function moveRight():void{

        var currentScreen:Screen = getCurrentScreen();

        var left:Array = Screen.screens;

        left.sort(sortHorizontal);

        for(var i:int = left.length - 1; i > 0; i--){

            if(left[i].bounds.left > stage.nativeWindow.bounds.left){

                stage.nativeWindow.x +=

                    left[i].bounds.left - currentScreen.bounds.left;

                stage.nativeWindow.y += left[i].bounds.top - currentScreen.bounds.top;

            }
```

```
        }

    }


    private function moveUp():void{

        var currentScreen:Screen = getCurrentScreen();

        var top:Array = Screen.screens;

        top.sort(sortVertical);

        for(var i:int = 0; i < top.length - 1; i++){

            if(top[i].bounds.top < stage.nativeWindow.bounds.top){

                stage.nativeWindow.x += top[i].bounds.left - currentScreen.bounds.left;

                stage.nativeWindow.y += top[i].bounds.top - currentScreen.bounds.top;

                break;

            }

        }

    }

```

```
        private function moveDown():void{
            var currentScreen:Screen = getCurrentScreen();


            var top:Array = Screen.screens;
            top.sort(sortVertical);
              for(var i:int = top.length - 1; i > 0; i--){
                  if(top[i].bounds.top > stage.nativeWindow.bounds.top){
                      stage.nativeWindow.x += top[i].bounds.left - currentScreen.bounds.left;
                      stage.nativeWindow.y += top[i].bounds.top - currentScreen.bounds.top;
                      break;
                  }
            }
        }


```

```
    private function sortHorizontal(a:Screen,b:Screen):int{

        if (a.bounds.left > b.bounds.left){

            return 1;

        } else if (a.bounds.left < b.bounds.left){

            return -1;

        } else {return 0;}

    }



    private function sortVertical(a:Screen,b:Screen):int{

        if (a.bounds.top > b.bounds.top){

            return 1;

        } else if (a.bounds.top < b.bounds.top){

            return -1;

        } else {return 0;}

    }
```

```
        private function getCurrentScreen():Screen{
            var current:Screen;
            var screens:Array = Screen.getScreensForRectangle(stage.nativeWindow.bounds);
            (screens.length > 0) ? current = screens[0] : current = Screen.mainScreen;
            return current;
        }
    }
}
```

本地菜单

Native Menu API 提供了创建程序，窗口，上下文菜单和弹出式菜单的相关类。

AIR 菜单概要

native menu 类运行程序访问到操作系统的本地菜单特性。NativeMenu 对象用于创建应用程序菜单（OSX 系统可用），window menus (Windows 系统可用), 上下文菜单和弹出式菜单。

菜单的类型

AIR 支持一下类型的菜单：

应用程序菜单

应用程序菜单必须通过 Shell 类访问，在 Mac OS X,系统上应用程序菜单由操作系统自动提供。应用程序使用这些AIR菜单API添加菜单项和子菜单到标准菜单上并注册处理函数，当然你也可以完全替换掉标准菜单。

窗体菜单

通过创建 NativeMenu 对象，然后赋值给 NativeWindow.menu 属性来添加窗体菜单。Windows 操作系统支持窗体菜单，但是 Mac OS X 不可以。

上下文菜单

上下文菜单被添加到 HTMLControl 组件的交互对象和文档元素上。最典型的例子就是在 HTML 元素上使用 Webkit API，当然也可以直接在 HTMLControl 上使用上下文菜单。

系统托盘菜单

通过创建NativeMenu对象并赋值给Shell.shell.icon.menu属性来创建系统托盘菜单。

Flex 菜单

Flex framework提供了一组Flex菜单类。这些Flex菜单类不需要窗体继承系统窗体风格即可使用而且还可在MXML格式中声明。如果你使用Flex Framework，这时就可以用Flex菜单类来代替本地菜单类了。

自定义菜单

本地菜单是在 Flash 和 HTML 渲染模型之外由操作系统负责绘制的，当然我们也可以自己绘制自己的非本地菜单，不过遗憾的是目前 AIR 菜单类还不支持“自绘制”菜单。

菜单结构

一个菜单包含一个或多个菜单项。一个菜单项可以是一个命令，一个子菜单，或一个分隔符，一般情况下，一个菜单项就是一个命令，如果菜单项的 submenu 属性被赋值为 NativeMenu 对象则变为子菜单，如果菜单项构造函数的 isSeparator 参数为 true，则变为分隔符。

最顶层的菜单项显示在菜单栏上,其他子菜单添加在应用程序和窗体菜单的顶层菜单中（虽然有些操作系统可以直接把菜单项放在菜单栏上，这样看起来好像很不符合常规）。

下面的图标演示典型的菜单结构。根菜单包含子菜单“File”和“Edit”。"File"菜单包含两个命令和一个子菜单（用于显示最近打开的文档，它也是一个子菜单，包含三个菜单项）。"Edit"菜单包含三个命令和一个分隔符。

菜单事件

菜单和菜单项都可以发出 displaying 和 select 事件：

Displaying:

在菜单即将被显示之前，菜单及其子菜单会发出 displaying 事件通知其注册的监听器。Displaying 事件给你一个机会来更新菜单内容或菜单项状态。例如在"Open Recent"菜单项的 displaying 处理函数可以改变反应当前阅读过的文档菜单项。

Event 对象的 target 属性指向被显示的菜单，currentTarget 则是被注册监听器监听的对象，既可以是菜单本身，也可以是其菜单项。

Select:

当用户点击命令项时，该命令项会发出一个 select 事件给注册的监听器，不过子菜单和分隔符不能被选择也不会发出 select 事件。

菜单项的 Select 事件会一直往上传递到顶层菜单，所以你可以直接在菜单项上监听也可以在菜单结构的上层的监听。当监听 select 事件时，需要通过 event 的 target 属性确定是哪个菜单项触发了事件，而 currentTarget 属性可以确定当前的菜单对象。

快捷键

快捷键是菜单上操作系统键盘的一部分功能。无论是 Mac OS X 还是 Windows 系统都允许用户用键盘打开或选择某个菜单命令项.

用字符串的索引指定快捷字符，字符串的第一个字母索引值为“0”。这样如果把“Format”字符串的“r“字母作为快捷字符，应设置其索引值 mnemonicIndex 属性为 2。

```
var item:NativeMenuItem = new NativeMenuItem("Format");
item.mnemonicIndex = 2;
```

菜单项状态

菜单项有两个状态属性，checked 和 enabled:

checked

设置为 true 会在菜单项字符前加上标志

```
var item:NativeMenuItem = new NativeMenuItem("Format");
item.checked = true;
```

enabled

设置菜单项是否可用

```
var item:NativeMenuItem = new NativeMenuItem("Format");
item.enabled = false;
```

关联对象到菜单项

NativeMenuItem 类的 data 属性允许与任意类型的对象相关联。例如"Open Recent"菜单中，你可以赋值一个 File 对象到菜单项上：

```
var file:File = new File("app-storage:/GreatGatsby.pdf");
var menuItem:NativeMenuItem = docMenu.addItem(new NativeMenuItem(file.name));
menuItem.data = file;
```

创建本地菜单

要创建一个菜单，先构造一个 NativeMenu 对象作为根菜单：

```
var root:NativeMenu = new NativeMenu();
```

作为窗体或应用程序的根菜单，所有的菜单项必须为子菜单（上下文菜单的根菜单可包含所有三种类型的菜单项），AIR 提供了两种方法创建子菜单。你可以通过菜单的 addSubmenu()方法添加菜单项：

```
var editMenu:NativeMenuItem = root.addSubmenu(new NativeMenu(), "Edit");
```

你也可以创建菜单项然后再赋值给子菜单：

```
var editMenu:NativeMenuItem = root.addItem("Edit");
editMenu.submenu = new NativeMenu();
```

菜单创建好后，再赋值给应用程序，窗体或上下文菜单：

设置应用程序菜单：

```
Shell.shell.menu = root;
```

设置窗体菜单：

```
nativeWindowObject.menu = root;
```

设置上下文菜单：

```
interactiveObject.contextMenu = root;
```

设置 dock icon 菜单

```
DockIcon(Shell.shell.icon).menu = root;
```

设置系统托盘菜单

```
SystemTrayIcon(Shell.shell.icon).menu = root;
```

定义菜单项，子菜单和分隔符

一个菜单项可以是一个命令项，一个子菜单或一个分隔符。一个菜单项如果设置构造函数的 isSeparator 参数为 true 这变成一个分隔符。一个菜单项如果赋值给 NativeMenu 对象的 submenu 属性则变成一个子菜单（或使用 addSubmenu() 方法），否则菜单项就是一个命令项，只有命令项可以发出 select 事件。

菜单中的菜单项顺序就是其加入的先后顺序，除非你自己修改了其索引值，如通过 addItemAt()方法在指定位置加入菜单项。

创建一个命令项：

```
var copy:NativeMenuItem = new NativeMenuItem("Copy",false);
copy.addEventListener(Event.SELECT,onCopyCommand);
editMenu.addItem(copy);
```

创建一个子菜单：

```
var editMenu:NativeMenuItem = new NativeMenuItem("Edit", false);
editMenu.submenu = new NativeMenu();
```

也可以使用菜单的 addSubmenu()方法添加菜单项：

```
var editMenu:NativeMenuItem = root.addSubmenu(new NativeMenu(), "Edit");
```

创建一个分隔符：

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

创建一个上下文菜单

在 AIR，ContextMenu API 类继承自 NativeMenu 类。ContextMenu 提供了一个事件属性，ContextMenuEvent.mouseTarget，其确定打开该菜单的对象。

上下文菜单（Context menus）可赋值给任何继承自 InteractiveObject 类的 Flash 对象。通过设置对象的 contextMenu 属性进行赋值。

上下文菜单仍可以包含命令项及分隔符，在 AIR 上下文菜单中没有默认的菜单项。

下面的例子创建了一个简单的 edit 菜单：

```
var editContextMenu:ContextMenu = new ContextMenu();
var item:NativeMenuItem;
item = editContextMenu.addItem(new ContextMenuItem("Cut"));
item.name="Cut";
editContextMenu.addItem(new ContextMenuItem("Copy"));
item.name="Copy";
editContextMenu.addItem(new ContextMenuItem("Paste"));
item.name="Paste";
editContextMenu.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT, function (event:ContextMenuEvent):void{
    var command:NativeMenuItem = event.target as NativeMenuItem;
```

```
        switch (command.name){
            case "Cut" :
                doCutCommand(event.mouseTarget);
                break;
            case "Copy" :
                doCopyCommand(event.mouseTarget);
                break;
            case "Paste" :
                doPasteCommand(event.mouseTarget);
                break;
        }
    });
this.contextMenu = editContextMenu; //Where "this" is an InteractiveObject
```

显示弹出式菜单

通过调用菜单的 display()方法可以随时在窗体的任意位置显示弹出式菜单。

下面的方法显示一个定义弹出式对象的菜单：

```
private function onMouseClick(event:MouseEvent):void{
    popupMenu.display(event.target.stage, event.stageX, event.stageY);
}
```

注意：实际上没有必要在事件处理函数中显示菜单，任何方法都可以调用 display()方法。

处理菜单事件

当一个菜单项被点击时，事件就会被触发，AIR 程序会对此作出反应。

下面的代码输出被激活的菜单名称：

```
var submenu:NativeMenu = new NativeMenu();
var red:NativeMenuItem = new NativeMenuItem("Red");
red.addEventListener(Event.SELECT,announceSelection)
var green:NativeMenuItem = new NativeMenuItem("Green");
green.addEventListener(Event.SELECT,announceSelection)
var blue:NativeMenuItem = new NativeMenuItem("Blue");
blue.addEventListener(Event.SELECT,announceSelection)
var menuItem:NativeMenuItem = new NativeMenuItem("Change Color");
submenu.addItem(red);
```

```
submenu.addItem(green);
submenu.addItem(blue);
menuItem.submenu = submenu;
Shell.shell.menu.addItem(menuItem);
function announceSelection(e:Event):void {
   var menuItem:NativeMenuItem = e.target as NativeMenuItem;
   trace(menuItem.label + " has been selected")
}
```

声明式定义菜单

输入菜单和菜单项的属性是一件很枯燥的事情，不过菜单是一个有规律的层级结构，可以使用 XML 格式定义直接写在函数里。

下面的类继承自 NativeMenu，在构造函数里传入 XML 对象，像这样：

```
package


{

   import flash.display.NativeMenu;

   import flash.display.NativeMenuItem;

   import flash.events.Event;



   public class DeclarativeMenu extends NativeMenu

    {

       public function DeclarativeMenu(XMLMenuDefinition:XML):void{

         super();
```

```
        addChildrenToMenu(this, XMLMenuDefinition.children());

    }


    private function addChildrenToMenu(menu:NativeMenu,
                                       children:XMLList):NativeMenuItem{
        var menuItem:NativeMenuItem;
        var submenu:NativeMenu;

        for each (var child:XML in children){
            if(String(child.@label).length > 0){
                menuItem = new NativeMenuItem(child.@label);
                menuItem.name = child.name();
            } else {
                menuItem = new NativeMenuItem(child.name());
                menuItem.name = child.name();
```

```
            }

            menu.addItem(menuItem);

              if(child.children().length() > 0){

                menuItem.submenu = new NativeMenu();

                addChildrenToMenu(menuItem.submenu,child.children());

            }

        }

        return menuItem;

    }

}//End class

}//End package
```

要使用这个类创建菜单，只需要传入一个 XML 格式的菜单定义即可：

```
var menuDefinition:String =
    "<root>" +
        "<FileMenu label='File'>" +
            "<NewMenu label='New'>" +
                "<NewTextFile label='Text file'/>" +
                "<NewFolder label='Folder'/>" +
                "<NewProject label='Project'/>" +
```

```
            "</NewMenu>" +
            "<OpenCommand label='Open'/>" +
            "<SaveCommand label='Save'/>" +
        "</FileMenu>" +
        "<EditMenu label='Edit'/>" +
            "<CutCommand label='Cut'/>" +
            "<CopyCommand label='Copy'/>" +
            "<PasteCommand label='Paste'/>" +
        "<EditMenu/>" +
        "<FoodItems label='Food Items'>" +
            "<Jellyfish/>" +
            "<Tripe/>" +
            "<Gizzard/>" +
        "</FoodItems>" +
    "</root>";
var test:DeclarativeMenu = new DeclarativeMenu(new XML(menuDefinition));
```

要监听菜单事件，你可以直接监听根菜单即可，通过 `event.target.name` 属性判定哪个命令项被触发，你也可以根据菜单名称搜索菜单项并添加独立的事件监听器。

例子：窗体和应用程序菜单

下面的例子使用菜单结构创建菜单，该例子也演示了事件处理过程：

```
package {

    import flash.display.NativeMenu;

    import flash.display.NativeMenuItem;

    import flash.display.NativeWindow;

    import flash.display.Sprite;

    import flash.events.Event;
```

```
import flash.filesystem.File;

import flash.system.Shell;

public class MenuExample extends Sprite

{

    private var recentDocuments:Array = new Array(new File("c:/GreatGatsby.pdf"),

                                new File("c:/WarAndPeace.pdf"),

                                new File("c:/Iliad.pdf"));

    public function MenuExample()

    {

        var fileMenu:NativeMenuItem;

        var editMenu:NativeMenuItem;

        if(NativeWindow.supportsMenu){

            stage.nativeWindow.menu = new NativeMenu();

            stage.nativeWindow.menu.addEventListener(Event.SELECT, selectCommandMenu);

            fileMenu = stage.nativeWindow.menu.addItem(new NativeMenuItem("File"));
```

```
            fileMenu.submenu = createFileMenu();

            editMenu = stage.nativeWindow.menu.addItem(new NativeMenuItem("Edit"));

            editMenu.submenu = createEditMenu();

        }



        if(Shell.supportsMenu){

            Shell.shell.menu.addEventListener(Event.SELECT, selectCommandMenu);

            fileMenu = Shell.shell.menu.addItem(new NativeMenuItem("File"));

            fileMenu.submenu = createFileMenu();

            editMenu = Shell.shell.menu.addItem(new NativeMenuItem("Edit"));

            editMenu.submenu = createEditMenu();

        }

    }



    public function createFileMenu():NativeMenu{
```

```
        var fileMenu:NativeMenu = new NativeMenu();

        fileMenu.addEventListener(Event.SELECT,selectCommandMenu);



        var newCommand:NativeMenuItem = fileMenu.addItem(new NativeMenuItem("New
"));

        newCommand.addEventListener(Event.SELECT,selectCommand);

        var saveCommand:NativeMenuItem = fileMenu.addItem(new NativeMenuItem("Save
"));

        saveCommand.addEventListener(Event.SELECT,selectCommand);

        var openRecentMenu:NativeMenuItem =

              fileMenu.addItem(new NativeMenuItem("Open Recent"));

        openRecentMenu.submenu = new NativeMenu();

        openRecentMenu.submenu.addEventListener(Event.DISPLAYING,

                              updateRecentDocumentMenu);

        openRecentMenu.submenu.addEventListener(Event.SELECT,selectCommandMenu);


```

```
            return fileMenu;

        }


        public function createEditMenu():NativeMenu{

            var editMenu:NativeMenu = new NativeMenu();

            editMenu.addEventListener(Event.SELECT,selectCommandMenu);


            var copyCommand:NativeMenuItem = editMenu.addItem(new NativeMenuItem("Copy
"));

            copyCommand.addEventListener(Event.SELECT,selectCommand);

            var pasteCommand:NativeMenuItem =

                    editMenu.addItem(new NativeMenuItem("Paste"));

            pasteCommand.addEventListener(Event.SELECT,selectCommand);

            editMenu.addItem(new NativeMenuItem("",true));

            var preferencesCommand:NativeMenuItem =
```

```
            editMenu.addItem(new NativeMenuItem("Preferences"));

        preferencesCommand.addEventListener(Event.SELECT,selectCommand);



        return editMenu;
    }


    private function updateRecentDocumentMenu(event:Event):void{

        trace("Updating recent document menu.");

        var docMenu:NativeMenu = NativeMenu(event.target);


        for each(var item:NativeMenuItem in docMenu.items){

            docMenu.removeItem(item);

        }


        for each(var file:File in recentDocuments){
```

```
            var menuItem:NativeMenuItem =

                docMenu.addItem(new NativeMenuItem(file.name));

            menuItem.data = file;

            menuItem.addEventListener(Event.SELECT, selectRecentDocument);

        }

    }


    private function selectRecentDocument(event:Event):void{

        trace("Selected recent document: " + event.target.data.name);

    }


    private function selectCommand(event:Event):void{

        trace("Selected command: " + event.target.label);

    }


```

```
        private function selectCommandMenu(event:Event):void{

            if(event.currentTarget.parent){

             var menuItem:NativeMenuItem =

                    findItemForMenu(NativeMenu(event.currentTarget));

               if(menuItem){

                trace("Select event for "" +

                      event.target.label +

                      "" command handled by menu: " +

                      menuItem.label);

             }

            } else {

             trace("Select event for "" +

                    event.target.label +

                    "" command handled by root menu.");

        }
```

```
    }


    private function findItemForMenu(menu:NativeMenu):NativeMenuItem{

        for each(var item:NativeMenuItem in menu.parent.items){

            if(item){

                if(item.submenu == menu){

                    return item;

                }

            }

        }

        return null;

    }

}
```

第十一章. 文件与数据

AIR 文件系统概要

Adobe AIR 提供了众多类来支持访问，创建和管理文件及其目录。这些类都包含在 flash.filesystem 包中，如下：

File

一个 File 对象表示一个文件或目录的路径。

FileMode

FileMode 类定义了一些字符串常量作为 FileStream 类的 open()和 openAsync()方法参数使用。这些参数决定 FileStream 对象打开的文件操作模式，如写入，读取，追加或更新。

FileStream

FileStream 对象用于打开文件并进行读写。一旦创建了一个 File 对象引用，就需要传递给 FileStream 对象以便进行打开或数据操作。

File 类的有些方法同时有同步和异步两个版本：

File.copyTo() 和 File.copyToAsync()

File.deleteDirectory() 和 File.deleteDirectoryAsync()

File.deleteFile() 和 File.deleteFileAsync()

File.getDirectoryListing() 和 File.getDirectoryListingAsync()

File.moveTo() 和 File.moveToAsync()

File.moveToTrash() 和 File.moveToTrashAsync()

相应的，FileStream 操作方式是否是异步取决于 FileStream 对象打开文件的方式：是调用 open()还是 openAsync()方法。

异步方法使进程在后台运行，当完成时再触发相应的事件通知。异步进程执行时，其他代码可以继续执行，而同步方法必须设定监听器如 File 或 FileStream 的 addEventListener()方法。

同步方法简单些，只要设定监听器处理就可以了，但是后台进程没有完成，程序将一直处于等待状态，这点特别的对于显示渲染或动画很不利，看起来好像无反应一样。

File对象的路径

每个 File 对象有两个属性定义路径：

nativePath 指定特定平台文件路径。例如，在 Windows 上路径大概是"c:\Sample directory\test.txt" 而在 Mac OS 上应该是"/Sample directory/test.txt"。注意不同的操作系统目录分隔符是不同的。

url 用 URL 格式定义文件路径。例如，在 Windows 上的路径大概是"file:///c:/Sample%20directory/test.txt"而在 Mac OS 上是"file:///Sample%20directory/test.txt"。

指向目录的 File 对象

这里有几种方法设置指向一个目录的 File 对象。

指向明确的目录

设定 File 对象指定用户的 home 目录，在 Windows，home 目录就是"My Documents"目录的父目录（例如"C:\Documents and Settings\userName\My Documents"）。在 Mac OS 上，就是 Users/userName 目录。下面的代码设置 File 对象指向用户目录的一个 AIR Test 子目录：

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

设置 File 对象指向用户的 documents 目录。在 Windows 上就是"My Documents"目录(如"C:\Documents and Settings\userName\My Documents")，在 On Mac OS,上，就是 Users/userName/Documents 目录，下面的代码设置 File 对象指向文档目录的子目录 AIR Test：

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

还可以指向桌面，下面的代码设置 File 对象指向桌面的 AIR Test 子目录：

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

File 对象还可指向应用程序的存储目录。每个 AIR 程序都有个独立的存储目录用于存储数据和配置文件。例如，下面的代码 File 对象指向配置文件 prefs.xml，该文件保存在应用程序存储目录：

```
var file:File = File.applicationStorageDirectory;
file = file.resolvePath("prefs.xml");
```

File 对象还可指向应用程序安装目录，通过 File.applicationResourceDirectory 属性指向应用程序资源目录。可用此目录检查应用程序描述文件或其他资源，例如下面的代码 File 对象指向资源目录的 images 子目录：

```
var file:File = File.applicationResourceDirectory;
file = file.resolvePath("images");
```

File.getRootDirectories()方法列出根目录卷标，如 winodws 上的 C：，在 Mac 上为"/"目录。

设置 nativePath 属性可指向明确的目录：

```
var file:File = new File();
file.nativePath = "C:\\AIR Test\\";
```

resolvePath()方法获得相对路径，例如下面的代码获得用户目录的"AIR Test"子目录路径：

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

还可通过 url 属性获得 url 格式的路径信息：

```
var urlStr:String = "file:///C:/AIR Test/";
var file:File = new File()
file.url = urlStr;
```

让用户浏览选择目录

File 类包含一个 browseForDirectory()方法，会弹出一个系统对话框让用户选择目录，该方法是异步的，如果用户点击 Open 按钮它会触发 select 事件，否则触发 cancel 事件。

例如，下面的代码让用户选择一个目录，然后输出该目录路径：

```
var file:File = new File();
file.addEventListener(Event.SELECT, dirSelected);
file.browseForDirectory();
function dirSelected(e:Event):void {
   trace(file.nativePath);
}
```

指向文件的 File 对象

有几种方法：

指向明确的文件路径

使用 resolvePath()方法获得文件相对路径：

```
var file:File = File.applicationStorageDirectory;
file = file.resolvePath("log.txt");
```

使用 url 格式的路径：

```
var urlStr:String = "file:///C:/AIR Test/test.txt";
var file:File = new File()
file.url = urlStr;
```

URL 可作为构造函数参数：

```
var urlStr:String = "file:///C:/AIR Test/test.txt";
var file:File = new File(urlStr);
```

注意 url 属性总是返回 URI 编码的字符串（如空格转换为%20）：

```
file.url = "file:///c:/AIR Test";
trace(file.url); // file:///c:/AIR%20Test
```

nativePath 属性设置文件明确路径，例如下面的代码在 windows 中设置文件对象指向 test.txt 文件：

```
var file:File = new File();
file.nativePath = "C:/AIR Test/test.txt";
```

也可作为构造函数参数：

var file:File = new File("C:/AIR Test/test.txt");

在 Windows 系统中分隔符既可以用(/) 也可以是(\) ，在 Mac OS 系统中只可以用(/)：

var file:File = new File(/Users/dijkstra/AIR Test/test.txt");

列举目录文件

使用 getDirectoryListing()方法获得文件和子目录数组。

让用户选择文件

File 类包含下列方法打开系统对话框让用户选择文件：

browseForOpen()

browseForSave()

browseForMultiple()

例如，下面的代码演示用户点击 Open 对话框让用户选择文件：

```
var fileToOpen:File = File.documentsDirectory;
selectTextFile(fileToOpen);

function selectTextFile(root:File):void
{
    var txtFilter:FileFilter = new FileFilter("Text", "*.as;*.css;*.html;*.txt;*.xml");
    root.browseForOpen("Open", [txtFilter]);
    root.addEventListener(Event.SELECT, fileSelected);
}

function fileSelected(event:Event):void
{
    trace(fileToOpen.nativePath);
}
```

如果程序已经打开了一个该对话框，再次打开会抛出 runtime 异常。

修改文件路径

可通过 resolvePath()方法修改路径或修改 nativePath 及 url 属性：

```
var file1:File = File.documentsDirectory;
file1 = file1.resolvePath("AIR Test");
trace(file1.nativePath); // C:\Documents and Settings\userName\My Documents\AIR Test
```

var file2:File = File.documentsDirectory;

file2 = file2.resolvePath("..");

trace(file2.nativePath); // C:\Documents and Settings\userName

var file3:File = File.documentsDirectory;

file3.nativePath += "/subdirectory";

trace(file3.nativePath); // C:\Documents and Settings\userName\My Documents\subdirector
y

var file4:File = new File();

file.url = "file:///c:/AIR Test/test.txt"

trace(file3.nativePath); // C:\AIR Test\test.txt

支持 URL 模式

可使用下列任意 URL 格式定义 File 对象的 url 属性：

file 指定相对于文件系统的路径：

file:///c:/AIR Test/test.txt

app-resource 指定相对于应用程序安装目录的相对路径，例如下面的 images 子目录位于程序安装目录之下：

app-resource:/images

app-storage 指定相对于程序储存目录的相对路径，每个安装的程序都有唯一的数据存储目录：app-storage:/settings/prefs.xml

两个文件的相对路径

使用 getRelativePath()方法找出两个文件的相对路径：

var file1:File = File.documentsDirectory.resolvePath("AIR Test");

var file2:File = File.documentsDirectory

file2 = file2.resolvePath("AIR Test/bob/test.txt");

trace(file1.getRelativePath(file2)); // bob/test.txt

第二个参数为 true 表示返回的结果中含有（..）路径表示法：

var file1:File = File.documentsDirectory;

file1 = file1.resolvePath("AIR Test");

var file2:File = File.documentsDirectory;

file2 = file2.resolvePath("AIR Test/bob/test.txt");

var file3:File = File.documentsDirectory;

file3 = file3.resolvePath("AIR Test/susan/test.txt");

trace(file2.getRelativePath(file1, true)); // ../..

trace(file3.getRelativePath(file2, true)); // ../../bob/test.txt

获取正确的文件名

文件和路径名是不区分大小写的，下面的两个 File 对象实际上指向同一个文件：

File.documentsDirectory.resolvePath("test.txt");

File.documentsDirectory.resolvePath("TeSt.TxT");

然而，文件和目录名字确实包含大写字母的使用，例如假定文档目录中有一个目录叫 AIR Test ：

var file:File = File.documentsDirectory.resolvePath("AIR test");

trace(file.nativePath); // ... AIR test

file.canonicalize();

trace(file.nativePath); // ... AIR Test

Canonicalize 方法转换 nativePath 对象为正确的大小写字母。

Canonicalize 方法还可以把路径缩写转换为 Windows 下的长文件名：

var path:File = new File();

path.nativePath = "C:\\AIR~1";

path.canonicalize();

trace(path.nativePath); // C:\AIR Test

获取文件系统信息

File 类包含下列静态属性提供文件系统信息：

File.lineEnding 行结束符，这取决于具体操作系统。

File.separator 路径分隔符，在 Mac OS 中是(/)，在 Windows 中是(\)。

Capabilities 类也包含一些和文件有关的系统信息：

File.systemCharSet 操作系统采用的文件编码

Capabilities.hasIME 操作系统是否安装输入法

Capabilities.language 操作系统语言

Capabilities.os 当前操作系统

使用 File API，可完成如下功能：

创建目录

File.createDirectory()方法用于创建目录。例如，下面的代码在用户目录创建名为 AIR Test 的子目录：

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

如果目录已经存在，则 createDirectory()方法无效果。

创建临时目录

File 类的 createTempDirectory()方法在系统临时目录中创建临时目录：

```
var temp:File = File.createTempDirectory();
```

createTempDirectory()方法自动在临时目录下创建唯一的临时子目录。

使用此方法可在临时目录创建临时目录，注意 createTempFile()方法会创建唯一的临时目录。

在应用程序关闭前可删除临时目录，因为该目录不会自动删除。

枚举目录

File 对象的 getDirectoryListing()方法或 getDirectoryListingAsync()方法得到目录下的文件和子目录数组。

例如，下面的代码列举用户文档目录下的文件和子目录：

```
var directory:File = File.documentsDirectory;
var contents:Array = directory.getDirectoryListing();
for (var i:uint = 0; i < contents.length; i++)
{
   trace(contents[i].name, contents[i].size);
}
```

如果使用该方法的异步版本，directoryListing 事件对象有一个 files 属性就是获取的文件数组：

```
var directory:File = File.documentsDirectory;
directory.getDirectoryListingAsync();
directory.addEventListener(FileListEvent.DIRECTORY_LISTING, dirListHandler);

function dirListHandler(event:FileListEvent):void
```

```
{
    var contents:Array = event.files;
    for (var i:uint = 0; i < contents.length; i++)
    {
        trace(contents[i].name, contents[i].size);
    }
}
```

拷贝和移动目录

复制和移动目录使用同一个方法，如下代码：

```
var sourceDir:File = File.documentsDirectory.resolvePath("AIR Test");
var resultDir:File = File.documentsDirectory.resolvePath("AIR Test Copy");
sourceDir.copyTo(resultDir);
```

注意这里指定 copyTo() 方法第二个参数为 true（默认），会删除已存在的目标目录下所有内容（即便源目录没有任何内容）。

删除目录内容

File 类还提供了 deleteDirectory()方法和 deleteDirectoryAsync() 方法，两个方法都是删除目录，两个方法都包含一个 deleteDirectoryContents 参数（布尔值），如果为 true（默认为 false）表示删除非空目录。

例如，下面的代码异步删除用户文档目录下的 AIR Test 子目录：

```
var directory:File = File.documentsDirectory.resolvePath("AIR Test");
directory.deleteDirectory(true);
```

下面的代码异步删除目录：

```
var directory:File = File.documentsDirectory.resolvePath("AIR Test");
directory.addEventListener(Event.COMPLETE, completeHandler)
directory.deleteDirectoryAsync(true);

function completeHandler(event:Event):void {
    trace("Deleted.")
}
```

还有一些方法如 moveToTrash()和 moveToTrashAsync 方法，它把目录移动到回收站。