针对 ADOBE® AIR® 开发本机扩展

法律声明

目录

第 1 章 : Adobe AIR 的本机扩展简介

“Adobe AIR 本机扩展 ” 是一些代码库，其中包含用 ActionScript API 包装的本机代码。您可以在 AIR 应用程序中使用本机扩展，以访问 AIR 不支持的平台功能，从而受益于用于关键算法的本机代码级别的性能，并重用现有本机代码库。

关于本机扩展

什么是 Adobe AIR?

Adobe® AIR® 是一种跨操作系统的运行时，内容开发人员可以使用它构建丰富 Internet 应用程序 (RIA)。开发人员可以将 RIA 部署到桌面、移动设备和数字家庭设备。可以使用 Adobe® Flex® 和 Adobe® Flash®（基于 SWF）也可以使用 HTML、JavaScript 和 Ajax（基于 HTML）构建 AIR 应用程序。有关可用于构建 AIR 应用程序的 Adobe Flash Platform 工具的详细信息，请参阅构建 Adobe AIR 应用程序中的适用于 AIR 开发的 Adobe Flash Platform 工具。

什么是 Adobe ActionScript?

基于 SWF 的 AIR 应用程序可以使用 Adobe ActionScript® 3.0。 ActionScript 3.0 是一种面向对象的语言，可以为 RIA 添加交互和数据处理功能。有关该语言的详细信息，请参阅学习使用 ActionScript 3.0 和 ActionScript 3.0 开发人员指南。

ActionScript 提供了许多内置类。例如， MovieClip、 Array 和 NetConnection 都是内置 ActionScript 类。此外，内容开发人员还可以创建特定于应用程序的类。有时，可以从内置类派生出特定于应用程序的类。

运行时执行 ActionScript 类中的代码。运行时还执行在基于 HTML 的应用程序中使用的 JavaScript 代码。

什么是本机扩展?

本机扩展是以下各项的组合：

- ActionScript 类。
- 本机代码。在本文档中，本机代码定义为在运行时外部执行的代码。例如，使用 C 语言编写的代码就是本机代码。在一些平台上，支持在扩展中使用 Java 代码。对于本文档，这些代码也将视为 “ 本机 ” 代码。

编写本机扩展的原因如下：

- 本机代码实现提供对特定于设备的功能的访问。这些特定于设备的功能在内置 ActionScript 类中不可用，也无法在特定于应用程序的 ActionScript 类中实现。本机代码实现可以提供此类功能，因为它可以访问特定于设备的硬件和软件。
- 本机代码实现有时可能比仅使用 ActionScript 的实现速度更快。
- 本机代码实现允许您重复使用现有代码。

例如，您可以创建一个本机扩展，允许应用程序执行以下操作：

- 使移动设备振动。
- 与特定于设备的库和功能交互。

在完成 ActionScript 和本机实现后，便可以打包您的扩展。然后， AIR 应用程序开发人员可以使用该包来调用此扩展的 ActionScript API，以执行特定于设备的功能。该扩展与 AIR 应用程序运行在同一进程中。

本机扩展与 NativeProcess ActionScript 类

ActionScript 3.0 提供了一个 NativeProcess 类。此类允许 AIR 应用程序在主机操作系统上执行本机进程。此功能与本机扩展的功能类似，后者提供对特定于设备的功能和库的访问。在决定使用 NativeProcess 类还是创建本机扩展时，请考虑以下因素：

- 只有 extendedDesktop AIR 配置文件支持 NativeProcess 类。因此，对于使用 AIR 配置文件 mobileDevice 和 extendedMobileDevice 的应用程序，本机扩展是唯一选择。
- 本机扩展开发人员通常为各种平台提供本机实现，但其提供的 ActionScript API 在各平台上通常相同。使用 NativeProcess 类时，不同平台上启动本机进程的 ActionScript 代码可能会不同。
- NativeProcess 类启动一个单独的进程，而本机扩展与 AIR 应用程序运行在同一进程中。因此，如果担心代码崩溃，则使用 NativeProcess 类比较安全。不过，单独的进程意味着可能需要实现进程间的通信处理。

本机扩展与 ActionScript 类库 （SWC 文件）

本机扩展与 SWC 文件最重要的区别是 SWC 文件不包含本机代码。因此，如果您确定不使用本机代码可以实现目标，请使用 SWC 文件而不使用本机扩展。

更多帮助主题

关于 SWC 文件

支持的设备

可为以下设备创建本机扩展：

- Android 设备，从 AIR 3 和 Android 2.2 开始。
- iOS 设备，从 AIR 3 和 iOS 4.0 开始
- iOS Simulator，从 AIR 3.3 开始
- Blackberry PlayBook，从 AIR 2.7 开始
- 支持 AIR 3.0 的 Windows 桌面设备
- 支持 AIR 3.0 的 Mac OS X 桌面设备

一个扩展可以面向多种平台。有关详细信息，请参阅第 4 页的 “ 面向多种平台 ”。

支持的设备配置文件

以下 AIR 配置文件支持本机扩展：

- extendedDesktop，从 AIR 3.0 开始
- mobileDevice，从 AIR 3.0 开始

更多帮助主题

AIR 配置文件支持

本机扩展体系结构

体系结构概述

AIR 允许扩展执行以下操作：

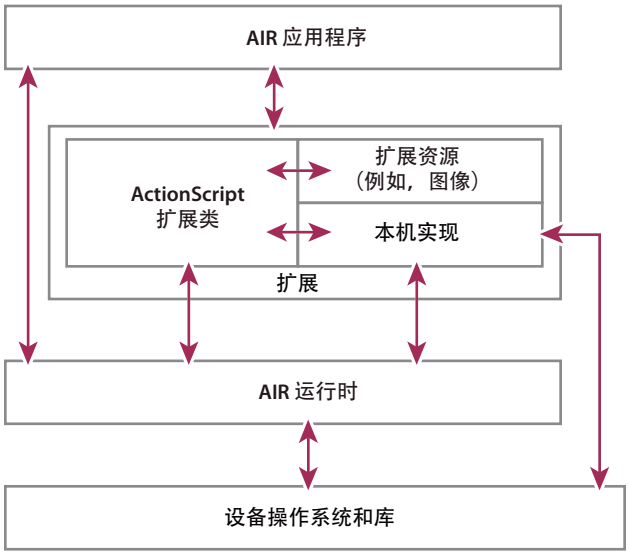
- 从 ActionScript 调用在本机代码中实现的函数。
- 在 ActionScript 和本机代码之间共享数据。
- 从本机代码向 ActionScript 调度事件。

在创建本机扩展时，您需要提供以下内容：

- 所定义的 ActionScript 扩展类。这些 ActionScript 类使用允许访问本机代码并与之交换数据的内置 ActionScript API。
- 本机代码实现。本机代码使用允许访问 ActionScript 扩展类并与之交换数据的本机代码 API。
- ActionScript 扩展类或本机代码使用的资源 （例如，图像）。

您的本机扩展可以面向多种平台。当面向多种平台时，您可以为每种目标平台提供一组不同的 ActionScript 扩展类以及不同的本机代码实现。有关详细信息，请参阅第 4 页的 “ 面向多种平台 ”。

下图说明了本机扩展、 AIR 运行时和设备之间的交互关系。



本机扩展体系结构

本机代码编程语言

Adobe AIR 提供了本机代码实现用来与 ActionScript 扩展类进行交互的本机代码 API。这些 API 采用以下语言提供：

- C 编程语言。
- Java

您的本机代码实现使用 C API 或 Java API（但不能同时使用两者）与 ActionScript 扩展类进行交互。不过，您的本机代码实现的其余部分不必专门使用与 API 相同的语言。例如，使用 C API 的开发人员还可以使用：

- C++
- Objective-C
- 使用汇编程序代码来利用高度优化的例程

下表说明了根据目标设备，需要使用哪种扩展 API：

设备	要使用的本机代码 API
Android 设备	具有 Android SDK 的 Java API。 具有 Android NDK 的 C API。
iOS 设备	C API
Blackberry PlayBook	C API
Windows 桌面设备	C API
Mac OS X 桌面设备	C API

面向多种平台

一个本机扩展通常面向多种平台。例如，一个扩展可以面向运行 iOS 的设备和运行 Android 的设备。在这种情况下，您的 ActionScript 类实现和本机代码实现 （包括本机代码语言）可能因目标平台而异。

对于 ActionScript 扩展类，最佳做法是提供与其实现无关的相同 ActionScript 公共接口。通过保持相同的公共接口，可以创建一个真正的跨平台本机扩展。如果 ActionScript 公共接口相同，但 ActionScript 实现不同，则需要为每个平台创建一个不同的 ActionScript 库。

您还可以为某些目标平台创建没有本机代码实现的扩展。此类扩展在以下情形中非常有用：

- 只有一些目标平台支持所需功能的本机实现时。

  扩展可以在这些平台上使用本机实现，但在其他平台上使用仅 ActionScript 实现。例如，假设一个平台为计算机进程之间的通信提供专用机制。对于该平台，扩展具有一个本机实现。对于另一平台，该扩展采用使用 ActionScript Socket 类的仅 ActionScript 实现。

  当应用程序开发人员使用扩展时，他们可以编写一个应用程序，无需了解扩展在不同目标平台上的实现方式。

- 在测试扩展时。

  假设有一个本机扩展使用移动设备的特定功能。您可以为桌面创建仅 ActionScript 扩展。然后，应用程序开发人员可以在开发期间使用该桌面扩展进行模拟测试，然后才在实际目标设备上测试。同样，作为扩展开发人员，您也可以在涉及本机代码实现之前测试扩展的 ActionScript 端。

在发布扩展时，可以在扩展描述符文件的 <platform> 元素中指定目标平台。每个 <platform> 元素指定一个目标，如 iPhone-ARM 或 Windows-x86。您还可以指定一个名为 default 的 <platform> 元素。default 平台有一个未使用 <platform> 元素指定的、在所有平台上使用的仅 ActionScript 实现。有关详细信息，请参阅第 63 页的 “ 本机扩展描述符文件 ”。

注：针对至少一个目标平台的实现必须包含本机代码。如果无任何目标平台需要本机代码，则使用本机扩展不是正确选择。在这样的情况下，请创建一个 SWC 库。

扩展在运行时的提供方式

本机扩展在运行时可以通过下列方式之一供应用程序使用：

应用程序绑定 扩展与 AIR 应用程序一起打包，并随应用程序安装在目标设备上。一个扩展包通常包含用于多种平台的本机实现和 ActionScript 实现，但只能包含一个平台的本机实现和 ActionScript 实现。有时，扩展包还包含一个用于不支持的平台或测试平台的仅 ActionScript 实现。

设备绑定 扩展单独安装在目标设备上的某个目录中，独立于任何 AIR 应用程序而存在。要使用设备绑定，通常需要与设备制造商协作将扩展安装在设备上。

下表说明了哪些设备支持应用程序绑定和设备绑定：

	应用程序绑定	设备绑定
Android	是	否
iOS	是	否
Blackberry PlayBook	是	是
Windows	是	否
Mac OS X	是	否

扩展上下文

本机扩展在每次运行应用程序时都会加载。不过，要使用本机实现，扩展的 ActionScript 部分需要调用一个特殊的 ActionScript API 来创建扩展上下文。

本机扩展可以执行以下任一操作。

- 仅创建一个扩展上下文。

  对于本机实现中仅提供一个函数集的简单扩展，通常只有一个扩展上下文。

- 创建多个共存的扩展上下文。

  在将 ActionScript 对象与本机对象关联时，多个扩展上下文非常有用。ActionScript 对象与本机对象之间的每个关联都是一个扩展上下文实例。这些扩展上下文实例可以具有不同的上下文类型。本机实现可以为每种上下文类型提供不同的函数集。

每个扩展上下文都可以具有在本机实现中定义和使用的特定于上下文的数据。

扩展上下文只能由扩展中的 ActionScript 代码创建，而不能由本机代码或应用程序代码创建。

创建本机扩展的任务概述

要创建本机扩展，请执行以下任务：

1 定义 ActionScript 扩展类的方法和属性。

2 编写 ActionScript 扩展类代码。

  请参阅第 6 页的 “ 编写 ActionScript 端代码 ”。

3 编写本机实现代码。

  请参阅第 13 页的 “ 使用 C 语言编写本机端代码 ” 和第 26 页的 “ 使用 Java 语言编写本机端代码 ”。

4 构建 ActionScript 端和本机端，创建一个扩展描述符文件，并打包该扩展及其资源。

  参见第 35 页的 “ 打包本机扩展 ” （针对所有设备）。

5 记录 ActionScript 扩展类的公共接口。

与任何软件开发一样，通常需要反复执行这些步骤。

第 2 章：编写 ActionScript 端代码

本机扩展由两部分组成：

- 所定义的 ActionScript 扩展类。
- 本机实现。

ActionScript 扩展类可以访问本机实现并与之交换数据。通过 ActionScript 类 ExtensionContext 来实现数据访问功能。只有扩展本身的 ActionScript 代码才能访问 ExtensionContext 类方法。

对扩展的 ActionScript 端进行编码包括以下任务：

- 声明 ActionScript 扩展类的公共接口。
- 使用静态方法 ExtensionContext.createExtensionContext() 创建 ExtensionContext 实例。
- 使用 ExtensionContext 实例的 call() 方法调用本机实现中的方法。
- 向 ExtensionContext 实例添加事件侦听器，侦听从本机实现调度的事件。
- 使用 dispose() 方法删除 ExtensionContext 实例。
- 在 ActionScript 端与本机端之间共享数据。共享的数据可以是任何 ActionScript 对象。
- 使用 getExtensionDirectory() 方法访问安装扩展的目录。与扩展相关的所有信息和资源都位于此目录中。（对于 iOS 设备，此规则存在例外情况。）

有关本机扩展的示例，请参阅 Adobe AIR 的本机扩展。

有关 ExtensionContext 类的详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考。

声明公共接口

创建本机扩展的第一步是确定扩展的公共接口。应用程序代码使用这些公共接口与扩展交互。 ActionScript 代码会编译到扩展名为 .as 的文件中。请使用自己的类定义创建 .as 文件。例如，以下代码显示了一个尚未编写类实现的简单 TVChannelController 扩展类的声明。应用程序可以利用该类来操作假想的电视的频道设置。

```
package com.example {
    public class TVChannelController extends EventDispatcher {

        public function TVChannelController() {
        }

        public function set currentChannel(channelToSet:int):void {
        }

        public function get currentChannel():int {
        }
    }
}
```

注：在设计公共接口时，应考虑是否会发布扩展的后续版本。如果要发布后续版本，在初始设计中要考虑向后兼容性支持问题。有关设备绑定扩展的向后兼容性问题的详细信息，请参阅第 11 页的 “ 本机扩展向后兼容性 ”。

检查本机扩展支持

最佳做法是始终定义一个公共接口，在本机扩展与 AIR 应用程序之间提供握手。指示使用扩展的 AIR 应用程序开发人员在调用任何其他扩展方法之前先检查此方法。

例如，假定有一个名为 isSupported() 的 ActionScript 扩展类公共接口。使用 isSupported() 方法，AIR 应用程序可以根据运行应用程序的设备是否支持扩展来进行逻辑判断。如果 isSupported() 返回 false，则 AIR 应用程序必须决定在不支持扩展的情况下怎么办。例如， AIR 应用程序可以决定退出应用程序。

创建 ExtensionContext 实例

要开始处理本机实现， ActionScript 扩展类需要使用 ExtensionContext 静态方法 createExtensionContext()。该方法返回 ExtensionContext 类的新实例。

```
package com.example {
    public class TVChannelController extends EventDispatcher {

        private var extContext:ExtensionContext;

        public function TVChannelController() {
            extContext = ExtensionContext.createExtensionContext(
                            "com.example.TVControllerExtension", "channel");

        }
        .
        .
        .
    }
}
```

在本示例中，构造函数调用 createExtensionContext()。虽然扩展类可以通过任何方法调用 createExtensionContext()，但是通常使用构造函数或其他初始化方法来进行调用。在该类的一个数据成员中保存返回的 ExtensionContext 实例。

注：建议不要将 createExtensionContext() 作为静态数据成员定义的一部分进行调用。这样做会使运行时提前创建应用程序暂时不需要的扩展上下文。如果应用程序的执行路径最后没有使用该扩展，则创建上下文会浪费设备资源。

createExtensionContext() 方法具有两个参数：一个扩展 ID 参数和一个上下文类型参数。

扩展 ID

createExtensionContext() 方法具有一个字符串参数，该参数是扩展的标识符或名称。此名称与扩展描述符文件的 id 元素中使用的名称相同。（对扩展进行打包时会创建扩展描述符文件）。应用程序开发人员在其应用程序描述符文件的 extensionID 元素中也会使用此名称。如果具有指定名称的扩展不可用，则 createExtensionContext() 将返回 Null。

为了避免名称冲突， Adobe 建议为扩展 ID 使用反向 DNS。例如， TVControllerChannel 扩展的 ID 为 com.example.TVControllerExtension。由于所有扩展共享一个全局命名空间，所以为扩展 ID 使用反向 DNS 可以避免扩展之间出现名称冲突。

上下文类型

createExtensionContext() 方法具有一个字符串参数，该参数是新扩展上下文的上下文类型。该字符串指定关于新扩展上下文作用的详细信息。

例如，假设扩展 com.example.TVControllerExtension 对频道和音量设置都可以操纵。在 createExtensionContext() 中传递 "channel" 或 "volume" 指示新扩展上下文将用于哪项功能。该扩展中的另一个 ActionScript 类 （如 TVVolumeController）可以通过将 contextType 值设置为 "volume" 来调用 createExtensionContext()。本机实现在上下文初始化中使用 contextType 值。

通常，所定义的每一个可能的上下文类型值与本机实现中不同的方法集对应。因此，上下文类型与本机实现中有效的类对应。如果使用同一上下文类型多次调用 createExtensionContext()，则本机实现通常会创建特定本机类的多个实例。

如果多次调用 createExtensionContext() 所使用的上下文类型各不相同，则本机端通常执行不同的初始化。根据上下文类型，本机端可以创建不同本机类的实例，也可以提供不同的本机函数集。

注：简单扩展通常仅有一个上下文类型。也就是说，本机实现中只有一个方法集。在这种简单情况下，上下文类型字符串参数可为 Null。

调用本机函数

ActionScript 扩展类调用 ExtensionContext.createExtensionContext() 之后，即可调用本机实现中的方法。TVChannelController 示例调用本机方法 "setDeviceChannel" 和 "getDeviceChannel"，如下所示：

```
package com.example {
    public class TVChannelController extends EventDispatcher {

        private var extContext:ExtensionContext;
        private var channel:int;

        public function TVChannelController() {
            extContext = ExtensionContext.createExtensionContext(
                            "com.example.TVControllerExtension", "channel");

        }

        public function set currentChannel(channelToSet:int):void {
            extContext.call("setDeviceChannel", channelToSet);
        }

        public function get currentChannel():int {
            channel = int (extContext.call("getDeviceChannel"));
            return channel;
        }

}
```

ExtensionContext 的 call() 方法采用以下参数：

- functionName。该字符串表示本机实现中的一个函数。在 TVChannelController 示例中，这些字符串与 ActionScript 方法名称不同。您可以选择使用相同的名称。也可以选择 functionName 字符串是否与其表示的本机函数的名称相同。在本机实现中，应在该 functionName 字符串与本机函数之间建立关联。可在 FREContextInitializer() 方法的输出参数中设置这一关联。请参阅第 14 页的 “ 扩展上下文初始化 ”。
- 可选参数列表。每个参数都传递给本机函数。参数可以是基元类型 （如 int），也可以是任何 ActionScript 对象。

ExtensionContext 的 call() 方法的返回值是基元类型或任何 ActionScript 对象。返回的对象的子类取决于本机函数的返回值。例如，本机函数 "getDeviceChannel" 返回 int 值。

侦听事件

本机实现可以调度 ActionScript 扩展代码能够侦听的事件。通过这种机制，本机实现可以异步执行任务，并在任务完成时通知 ActionScript 端。

事件目标是 ExtensionContext 实例。因此，使用 ExtensionContext 实例的 addEventListener() 方法可以订阅来自本机实现的事件。

以下示例向 TVChannelController 添加相应的代码，用于接收来自本机实现的事件。使用扩展的应用程序调用 ActionScript 扩展类方法 scanChannels()，而该方法又调用本机函数 "scanDeviceChannels"。

该本机函数异步扫描所有可用频道。扫描完成后，该函数调度一个事件。 onStatus() 方法通过查询本机方法 "getDeviceChannels" 来处理该事件，从而获取频道列表。 onStatus() 方法将列表存储在 scannedChannelList 数据成员中，并向应用程序的侦听对象调度一个事件。当应用程序对象收到该事件时，即可调用 ActionScript 扩展类属性存取器 availableChannels。

```
package  com.example {
     public  class TVChannelController extends EventDispatcher {

        private var extContext:ExtensionContext;
        private var channel:int;
        private var scannedChannelList:Vector.<int>;

        public function TVChannelController() {
            extContext = ExtensionContext.createExtensionContext(
                            "com.example.TVControllerExtension", "channel");
            extContext.addEventListener(StatusEvent.STATUS, onStatus);
        }
        .
        .
        .
        public function scanChannels():void {
            extContext.call("scanDeviceChannels");
        }
        public function get availableChannels():Vector.<int> {
            return scannedChannelList;
        }
        private function onStatus(event:StatusEvent):void {
            if ((event.level == "status") && (event.code == "scanCompleted")) {
                scannedChannelList = (Vector.<int>)(extContext.call("getDeviceChannels"));
                dispatchEvent (new Event ("scanCompleted") );
            }
        }
    }
}
```

该示例阐释了以下几点：

- 本机实现只能调度 StatusEvent 对象。因此， addEventListener() 方法侦听事件类型 StatusEvent.STATUS。
- 本机实现设置 StatusEvent 对象的 code 和 level 属性。您可以定义要为这些属性使用的字符串。在本示例中，本机实现将 level 属性设置为 “status" 并将 code 属性设置为 "scanCompleted"。通常，StatusEvent 的 level 属性的值为 "status"、"info" 或 "error"。
- 由于 TVChannelController 是 EventDispatcher 的子类，因此它也可以调度一个事件。在本示例中，它调度一个 Event 对象，并将该对象的 type 属性设置为 "scanCompleted"。任何关注该事件的 ActionScript 对象均可侦听该事件。例如，以下代码显示了使用此扩展的 AIR 应用程序的一个代码片段。该应用程序创建 TVChannelController 对象。然后，它要求 TVChannelController 对象扫描频道。最后等待扫描完成。

```
var channelController:TVChannelController = new TVChannelController();
channelController.addEventListener("scanCompleted", onChannelsScanned);
channelController.scanChannels();
var channelList:Vector.<int>;

private function onChannelsScanned(evt:Event):void {
    if (evt.type == "scanCompleted") {
        channelList = channelController.availableChannels;}
}
```

释放 ExtensionContext 实例

ActionScript 扩展类可通过调用 ExtensionContext 方法 dispose() 来释放 ExtensionContext 实例。该方法通知本机实现清理该实例使用的资源。例如， TVChannelController 类可以添加一个方法来执行清理任务：

```
public function dispose (): void {
    extContext.dispose();
    // Clean up other resources that the TVChannelController instance uses.
}
```

ActionScript 扩展类无需显式调用 ExtensionContext 实例的 dispose() 方法。在这种情况下，当运行时的垃圾回收器释放 ExtensionContext 实例时，运行时会调用该方法。但最佳做法是显式调用 dispose()。与等待垃圾回收器清理资源相比，显式调用 dispose() 通常能够更加及时地清理资源。

不管是显式调用还是通过垃圾回收器清理， ExtensionContext 的 dispose() 方法都会导致调用本机实现的上下文终结器。有关详细信息，请参阅第 16 页的 “扩展上下文终止化”。

访问本机扩展的目录

有时，扩展会包含其他文件，如图像。有时，扩展还要访问扩展描述符文件中的信息，如扩展版本号。

要访问这些文件以获取除 iOS 设备之外的所有设备上的扩展，请使用 ExtensionContext 类静态方法 getExtensionDirectory()。例如：

```
var extDir:File = ExtensionContext.getExtensionDirectory("com.example.TVControllerExtension");
```

将扩展名称传递给 getExtensionDirectory()。该字符串值与下面两处使用的名称相同：

- 扩展描述符文件的 id 元素。
- 传递给 ExtensionContext.createExtensionContext() 的扩展 ID 参数。

返回的 File 实例引用基本扩展目录。扩展目录具有以下结构：

```
extension base directory/
    platform independent files
    META-INF/
        ANE/
            extension.xml
            platform name/
                platform dependent files and directories
```

无论扩展目录位于设备中的哪个位置，扩展的文件与基本扩展目录的相对位置始终不变。因此，使用返回的 File 实例和 File 类方法可以找到和操纵扩展所包含的特定文件。

扩展目录位置取决于扩展是通过应用程序绑定方式提供还是通过设备绑定方式提供，如下所述：

- 对于应用程序绑定方式，扩展目录位于应用程序目录中。

- 对于设备绑定方式，扩展目录位置取决于设备。

对于 iOS 设备的 ActionScript 扩展，在使用 getExtensionDirectory() 时存在例外情况。这些扩展的资源不在扩展目录中。而是位于顶级应用程序目录中。有关详细信息，请参阅 第 47 页的 “iOS 设备上的资源 ”。

更多帮助主题

第 4 页的 “ 扩展在运行时的提供方式 ”

从本机扩展标识调用应用程序

扩展的 ActionScript 端可以标识和评估使用扩展的 AIR 应用程序。例如，使用 ActionScript 类 NativeApplication 获取有关 AIR 应用程序的信息，如应用程序 ID 和签名数据。然后， ActionScript 端可以根据该信息作出运行时决定。

有时，本机实现也需要作出类似的运行时决定。在这种情况下， ActionScript 端可使用 ExtensionContext 实例的 call() 方法向本机实现报告应用程序信息。

本机扩展向后兼容性

向后兼容性和扩展的公共接口

一种最佳做法是在您的扩展的 ActionScript 公共接口中保持向后兼容性。请在该扩展的所有后续版本中继续支持该扩展的类、方法、属性和事件。

设备绑定扩展有一个更复杂的向后兼容性问题。有时，在扩展的不同版本之间，扩展的行为是不同的。例如，在该扩展的新版本中，某个特定方法返回一个具有新含义的值。当设备绑定扩展发生此行为时，某个应用程序可能会停止正常工作。如果在生成应用程序时使用的扩展版本与设备上安装的扩展版本具有不同的行为，则可能发生此问题。在此情况下，应用程序期待一个行为，但已安装的扩展提供一个不同的行为。

在这样的情况下，设备上安装的扩展可以决定如何继续。扩展可以完成下列工作：

- 查询生成该 AIR 应用程序所使用的扩展版本以及设备上安装的版本。
- 确定两个版本中扩展的行为是否不同。
- 如果 AIR 应用程序是用该扩展的旧版本生成的，请恢复到旧版本的行为。

  注：用扩展的新版本生成的 AIR 应用程序通常在设备上不可用。有关详细信息，请参阅第 12 页的 “ 向后兼容性和设备的应用程序存储 ”。

要查询生成应用程序所使用的扩展版本号，请执行下列操作：

1. 使用 File.applicationDirectory 获取应用程序安装目录。
2. 使用 File 类 API 访问生成该应用程序所使用的扩展的 extension.xml 文件。该文件位于：

   ```
   <application directory>/META-INF/AIR/extensions/<extensionID>/META-INF/ANE/extension.xml
   ```

3. 读取 extension.xml 文件的内容并查找 <versionNumber> 元素的值。

要查询已安装扩展的版本号，请执行下列操作：

1. 使用静态方法 ExtensionContext.getExtensionDirectory() 获取扩展的基本目录。
2. 使用 File 类 API 访问设备上安装的扩展的 extension.xml 文件。该文件位于：

   ```
   <extension base directory>/META-INF/ANE/extension.xml
   ```

3 读取 extension.xml 文件的内容并查找 <versionNumber> 元素的值。

向后兼容性和设备的应用程序存储

如果生成 AIR 应用程序所使用的扩展版本比设备上安装的扩展版本新，则该应用程序通常在此设备上不可用。该应用程序不可用的原因在于设备制造商处理从设备的应用程序存储发送到服务器的下载此类应用程序的请求的方式。Adobe 向设备制造商推荐以下处理方式：

- 考虑服务器下载使用该扩展的较新版本的应用程序的情况。服务器也会下载该扩展的较新版本。设备的应用程序存储同时安装应用程序和该扩展的较新版本。
- 考虑服务器无法下载该扩展的较新版本的情况。服务器也不会下载使用该版本扩展的应用程序。设备的应用程序存储正常处理这一情况，并且根据需要通知最终用户。
- 考虑服务器下载使用较新版本扩展的应用程序，但不下载较新版本扩展的情况。设备的应用程序存储不允许最终用户运行该应用程序。应用程序存储正常处理这一情况，并且根据需要通知最终用户。

第 3 章：使用 C 语言编写本机端代码

有些设备在其本机实现中使用 C 编程语言。如果您要将本机扩展的目标设定为此类设备，应使用本机扩展 C API 编写您的扩展的本机端代码。

C API 位于文件 FlashRuntimeExtensions.h 中。可在 AIR SDK 的 include 目录中获得该文件。可从 http://www.adobe.com/cn/products/air/sdk/ 获取 AIR SDK。

AIR 运行时用于将扩展的 ActionScript 端连接到扩展的本机端。

使用 C API，您可以执行以下任务：

- 初始化扩展。
- 在创建每个扩展上下文时对其进行初始化。
- 定义 ActionScript 端可以调用的函数。
- 将事件调度到 ActionScript 端。
- 访问从 ActionScript 端传递的数据，以及将数据传递回 ActionScript 端。
- 创建和访问上下文特定的本机数据以及上下文特定的 ActionScript 数据。
- 在扩展的工作完成时清理扩展资源。

有关每个 C API 函数的详细信息（例如参数和返回值），请参阅第 71 页的“本机 C API 参考”。

有关使用 C API 的本机扩展的示例，请参阅 Adobe AIR 的本机扩展。

扩展初始化

运行时在本机端调用扩展初始化函数。使用该扩展的应用程序每次运行时，运行时都会调用一次该初始化函数。尤其是，在扩展首次调用任何上下文的 ExtensionContext.createExtensionContext() 时，运行时都会调用初始化函数。

该函数初始化所有扩展上下文可以使用的数据。使用 FREInitializer() 的签名定义扩展初始值设定项函数。

例如：

```
void MyExtensionInitializer
    (void** extDataToSet, FREContextInitializer* ctxInitializerToSet,
     FREContextFinalizer* ctxFinalizerToSet)
{
    extDataToSet = NULL;  // This example does not use any extension data.
    *ctxInitializerToSet = &MyContextInitializer;
    *ctxFinalizerToSet = &MyContextFinalizer;
}
```

您定义的 FREInitializer() 方法将以下数据返回到运行时：

- 一个指针，指向运行时稍后传递给每个新扩展上下文的数据。例如，如果所有扩展上下文都使用同一实用程序库，则该数据可以包含指向该库的指针。该数据称为扩展数据。

  扩展数据可以是您选择的任何数据。它可以是简单的基元数据类型，也可以是指向您定义的结构的指针。在本示例中，指针为 NULL，因为扩展没有使用此数据。

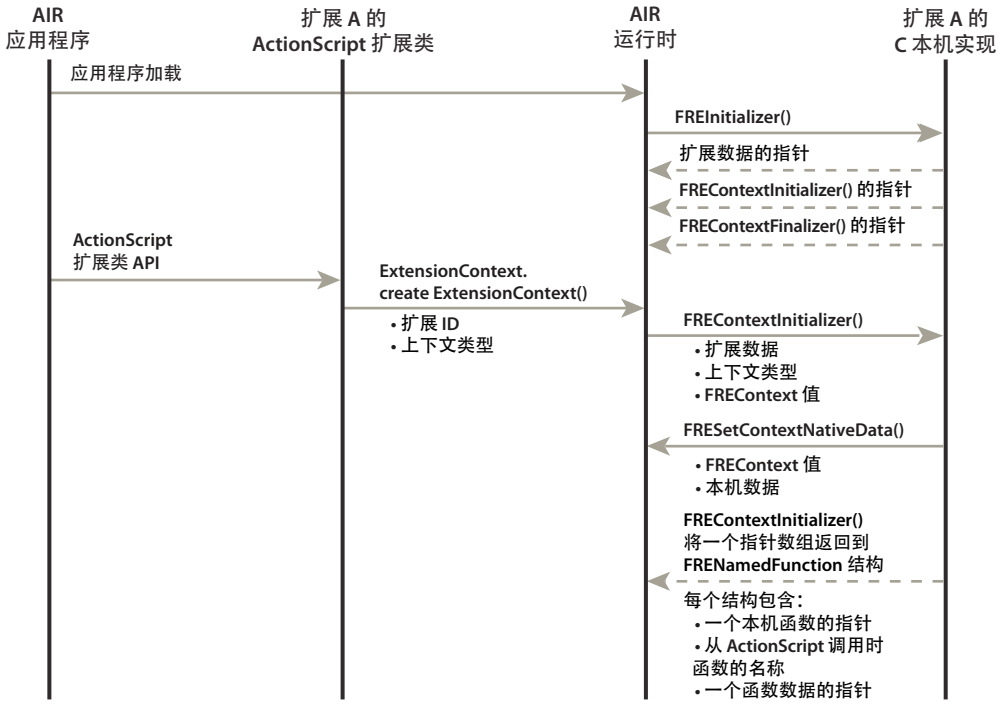
- 指向上下文初始化函数的指针。 ActionScript 端每次调用 ExtensionContext.createExtensionContext() 时，运行时都调用您提供的扩展上下文初始化函数。请参阅 第 77 页的“FREContextInitializer()”。

- 指向上下文终结器函数的指针。当运行时释放扩展上下文时，它会调用该函数。当 ActionScript 端调用 ExtensionContext 实例的 dispose() 方法时，发生该调用。如果未调用 dispose()，运行时将对 ExtensionContext 实例进行垃圾回收。请参阅 第 76 页的 “FREContextFinalizer()”。

对于应用程序绑定的扩展，您的 FREInitializer() 实现可以具有任何名称。在扩展描述符文件中指定初始化函数的名称。请参阅第 63 页的 “ 本机扩展描述符文件 ”。

对于设备绑定的应用程序，指定扩展初始化函数的方式由设备决定。

以下序列图明确调用 FREInitializer() 函数的 AIR 运行时。它还显示上下文初始化。有关详细信息，请参阅第 14 页的 “ 扩展上下文初始化 ”。



扩展初始化序列

扩展上下文初始化

为了使用本机 C 方法，您的扩展的 ActionScript 端首先调用静态方法 ExtensionContext.createExtensionContext()。调用 createExtensionContext() 会导致运行时执行以下操作：

- 创建 ExtensionContext 实例。
- 创建它用于跟踪扩展上下文的内部数据。
- 调用扩展上下文初始化函数。

扩展上下文初始化函数初始化新扩展上下文的本机实现。使用 FREContextInitializer() 的签名定义扩展上下文初始值设定项函数。

例如，振动示例使用以下函数：

```
void ContextInitializer(void* extData, const uint8_t* ctxType, FREContext ctx,
                        uint32_t* numFunctionsToSet,
                        const FRENamedFunction** functionsToSet) {

    *numFunctionsToSet = 2;

    FRENamedFunction* func = (FRENamedFunction*)malloc(sizeof(FRENamedFunction)*2);
    func[0].name = (const uint8_t*)"isSupported";
    func[0].functionData = NULL;
    func[0].function = &IsSupported;

    func[1].name = (const uint8_t*)"vibrateDevice";
    func[1].functionData = NULL;
    func[1].function = &VibrateDevice;

    *functionsToSet = func;
}
```

上下文初始化函数收到以下输入参数：

- 扩展初始化函数创建的扩展数据。请参阅第 13 页的 “ 扩展初始化 ”。
- 上下文类型。向 ActionScript 方法 ExtensionContext.createExtensionContext() 传递一个指定上下文类型的参数。运行时将此字符串值传递给上下文初始化函数。然后该函数使用此上下文类型选择 ActionScript 端可以调用的本机实现中的方法集。每个上下文类型通常都与一组不同的方法相对应。请参阅第 7 页的 “ 上下文类型 ”。

  上下文类型的值是 ActionScript 端和本机端之间约定的任意字符串。

  如果您的扩展在本机实现中只有一组方法，请在 ExtensionContext.createExtensionContext() 中传递 null 或空字符串。然后忽略扩展上下文初始值设定项中的上下文类型参数。
- FREContext 值。当运行时创建扩展上下文时，它会创建内部数据。它将该内部数据与 ActionScript 端的 ExtensionContext 类实例相关联。

  当您的本机实现向 ActionScript 端调度事件时，它会指定该 FREContext 值。运行时使用 FREContext 值将事件调度到相应的 ExtensionContext 实例。请参阅 第 83 页的 “FREDispatchStatusEventAsync()”。

  此外，本机函数可以使用该值访问和设置上下文特定的本机数据和上下文特定的 ActionScript 数据。

扩展上下文初始化函数设置以下输出参数：

- 本机函数的数组。 ActionScript 端可以通过使用 ExtensionContext 实例的 call() 方法来调用其中的每个函数。

  每个数组元素的类型均为 FRENamedFunction。该结构包括一个字符串，该字符串是 ActionScript 端用于调用该函数的名称。该结构还包括指向您编写的 C 函数的指针。运行时将该名称与 C 函数相关联。尽管名称字符串不必与实际的函数名称相匹配，但通常应使用同一名称。
- 本机函数数组中的函数数量。

明确调用 FREContextInitializer() 函数的 AIR 运行时的序列图位于第 13 页的 “ 扩展初始化 ” 中。

更多帮助主题

振动本机扩展示例

上下文特定数据

上下文特定的数据特定于扩展上下文。（重新调用该扩展数据是为了扩展中的所有扩展上下文。）上下文初始化方法、上下文终止化方法和本机扩展方法可以创建、访问和修改上下文特定的数据。

上下文特定的数据可以包括以下这些：

- 本机数据。该数据可以是您选择的任何数据。它可以是简单的基元数据类型，也可以是您定义的结构。请参阅 第 86 页的 “FREGetContextNativeData()” 和 第 100 页的 “FRESetContextNativeData()”。
- ActionScript 数据。该数据是 FREObject 变量。由于 FREObject 变量与 ActionScript 类对象相对应，因此该数据允许您保存并在以后访问 ActionScript 对象。请参阅 第 86 页的 “FREGetContextActionScriptData()” 和 第 99 页的 “FRESetContextActionScriptData()”。另请参阅 第 18 页的 “FREObject 类型 ”。

显示本机实现设置上下文特定的本机数据的序列图位于第 13 页的 “ 扩展初始化 ” 中。显示本机实现获取上下文特定的数据的序列图位于第 17 页的 “ 扩展函数 ” 中。

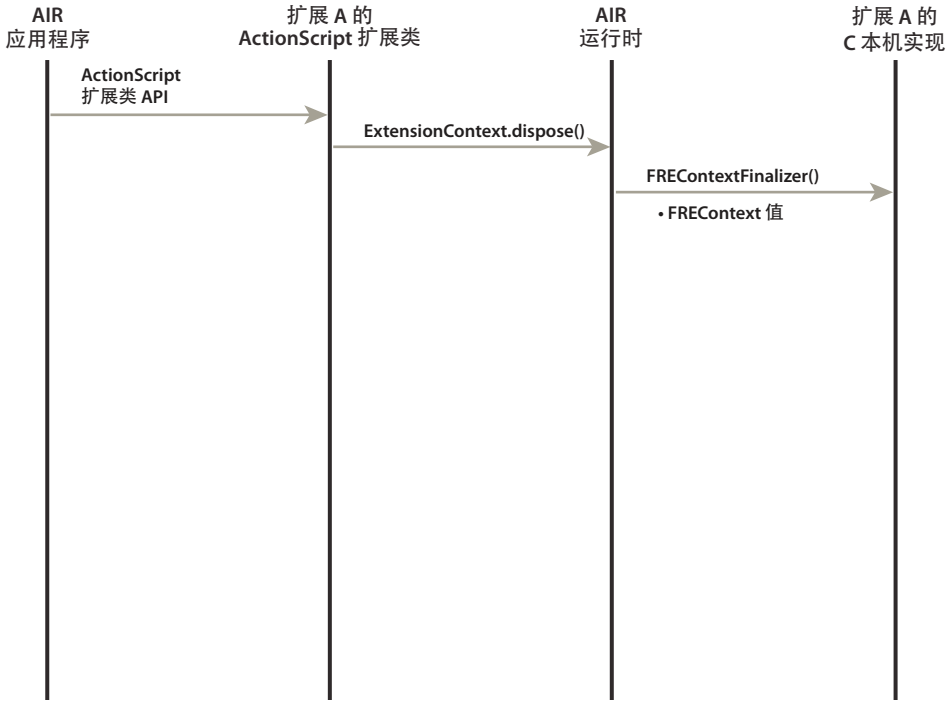
扩展上下文终止化

扩展的 ActionScript 端可以调用 ExtensionContext 实例的 dispose() 方法。调用 dispose() 会导致运行时调用扩展的上下文终止化函数。使用 FREContextFinalizer() 的签名定义扩展上下文终止化函数。

该方法具有一个输入参数：FREContext 值。可以将该 FREContext 值传递给 FREGetContextNativeData() 和 FREGetContextActionScriptData() 以访问上下文特定的数据。清理与该上下文关联的任何数据和资源。

如果 ActionScript 端不调用 dispose()，当不再存在对 ExtensionContext 实例的引用时，运行时垃圾回收器将释放该实例。此时，运行时将调用上下文终止化函数。

以下序列图明确调用 FREContextFinalizer() 函数的 AIR 运行时。



扩展上下文终止化序列

扩展终止化

C API 提供一个扩展终止化函数，供运行时在卸载扩展时调用。但是，运行时并非总是会卸载扩展。因此，运行时并非总是调用扩展终止化函数。

使用 FREFinalizer() 的签名定义扩展终止化函数。该方法具有一个输入参数：您在扩展初始化函数中创建的扩展数据。清理与该扩展关联的任何数据和资源。

对于应用程序绑定的扩展，您的 FREFinalizer() 实现可以具有任何名称。在扩展描述符文件中指定终止化函数的名称。请参阅第 63 页的 “ 本机扩展描述符文件 ”。

对于设备绑定的应用程序，指定扩展终止化函数的方式由设备决定。

扩展函数

扩展的 ActionScript 端通过调用 ExtensionContext 实例的 call() 方法来调用您实现的 C 函数。 call() 方法使用以下参数：

- 函数的名称。您在上下文初始化函数的输出参数中提供该名称。该名称是 ActionScript 端和本机端之间约定的任意字符串。通常，它是本机 C 函数的实际名称。不过，这两个名称可以不同，因为运行时会将任意名称与实际函数相关联。
- 本机函数的参数列表。这些参数可以是任何基元类型的 ActionScript 对象或 ActionScript 类对象。

使用同一函数签名 FREFunction() 定义各个本机函数。运行时会将以下参数传递给各个本机函数：

- FREContext 值。本机函数可以使用该值访问和设置上下文特定的数据。此外，本机实现还使用 FREContext 值将异步事件调度回 ActionScript 端。
- 指向与函数关联的数据的指针。该数据可以是任何本机数据。当运行时调用本机函数时，它会向函数传递该数据指针。
- 函数参数的数量。
- 函数参数。每个函数参数的类型均为 FREObject。这些参数与 ActionScript 类对象或基元数据类型对应。

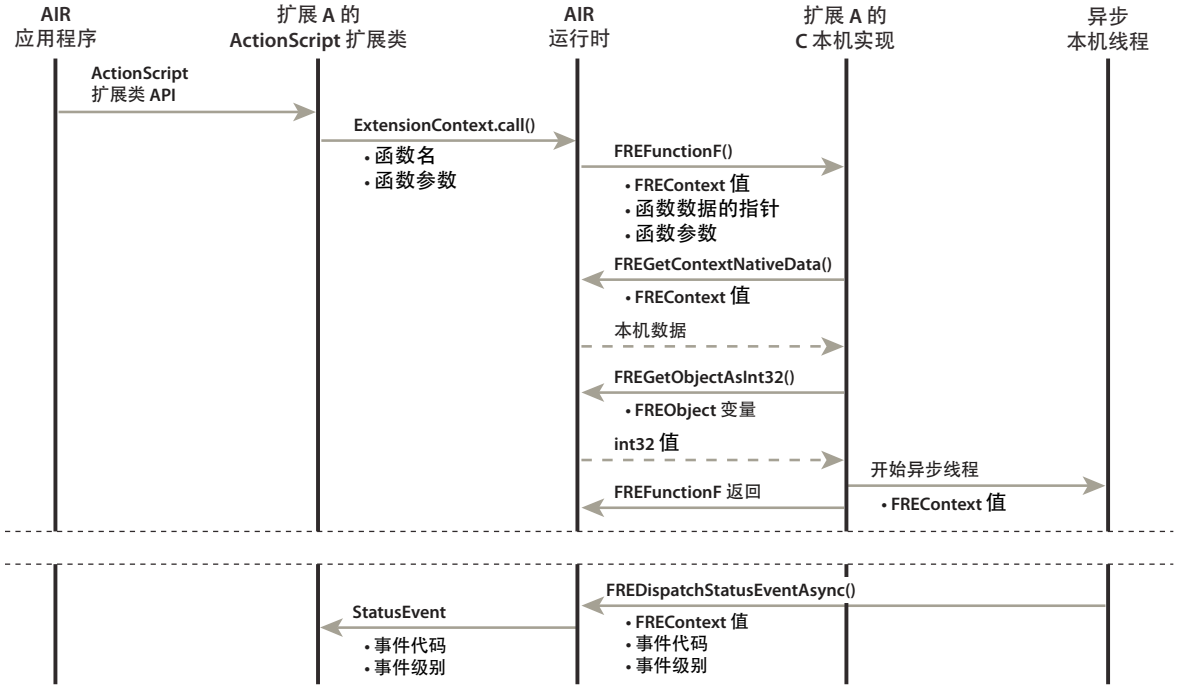
本机函数还具有 FREObject 类型的返回值。运行时返回相应的 ActionScript 对象作为 ExtensionContext call() 方法的返回值。

注：不要将本机函数的可见性设置隐藏。使用默认可见性。

以下序列图显示了 AIR 应用程序进行的函数调用导致调用名为 FREFunctionF() 的本机 C 函数。在本示例中， C 函数：

- 获取上下文特定的本机数据。
- 获取 ActionScript 对象的 int32 值。
- 启动一个异步线程，该线程稍后调度事件。

注：C 函数 FREFunctionF() 的行为只是一个用于演示调用序列的示例行为。



本机函数调用序列示例

调度异步事件

本机 C 代码可以将异步事件调度回扩展的 ActionScript 端。例如，扩展方法可启动另一线程来执行某项任务。当其他线程中的任务完成时，该线程会调用 FREDispatchStatusEventAsync() 来通知扩展的 ActionScript 端。事件的目标是 ActionScript ExtensionContext 实例。

第 17 页的 “ 扩展函数 ” 中的序列图显示了启动异步线程 （稍后调度事件）的本机 C 函数。

更多帮助主题

第 83 页的 “FREDispatchStatusEventAsync()”

FREObject 类型

FREObject 类型的变量引用与 ActionScript 类对象或基元类型对应的对象。在本机实现中使用 FREObject 变量来处理 ActionScript 数据。 FREObject 类型主要用于本机函数参数和返回值。

在编写本机函数时，需决定参数的顺序。由于您同时还编写 ActionScript 端，并在 ExtensionContext 实例的 call() 方法中使用该参数顺序，因此，尽管每个本机函数参数都是 FREObject 变量，但您知道它对应的 ActionScript 类型。

类似地，您需要决定本机函数的返回值 （如果有）的 ActionScript 类型。 call() 方法返回此类型的对象。尽管本机函数返回值始终是 FREObject 变量，但您知道它对应的 ActionScript 类型。

扩展 C API 提供用于使用 FREObject 变量所引用的对象的函数。由于这些对象与 ActionScript 数据相对应，因此这些 C API 函数决定您访问 ActionScript 类对象或基元数据变量的方式。所用的 C API 取决于 ActionScript 对象的类型。这些类型包括：

- ActionScript 基元数据类型

- ActionScript 类对象
- ActionScript String 对象
- ActionScript Array 或 Vector 类对象
- ActionScript ByteArray 类对象
- ActionScript BitmapData 类对象

注：您只能从 FREFunction 函数在其中运行的同一线程调用扩展 C API。一种例外情况是用于将事件调度到 ActionScript 端的 C API。可以从任何线程调用该函数 FREDispatchStatusEventAsync()。

确定 FREObject 变量的类型

有时，您不知道 FREObject 变量对应的 ActionScript 对象的类型。若要确定类型，请使用 C API 函数 FREGetObjectType()：

```
FREResult FREGetObjectType( FREObject object, FREObjectType *objectType );
```

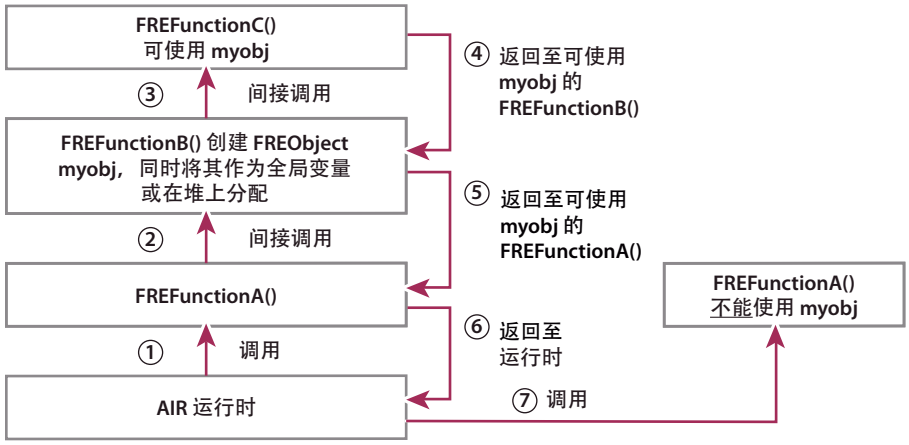
知道类型后，可使用适当的 C API 来处理该值。例如，如果类型是 FRE_TYPE_VECTOR，可第 23 页的“使用 ActionScript Array 和 Vector 对象”中的 C API 来处理该 Vector 对象。

FREObject 有效性

如果您尝试在 C API 调用中使用无效的 FREObject 变量， C API 将返回 FRE_INVALID_OBJECT 返回值。

任何 FREObject 变量都只在调用堆栈上的第一个 FREFunction 函数返回之前有效。调用堆栈上的第一个 FREFunction 函数是运行时由于 ActionScript 端调用 ExtensionContext 实例的 call() 方法而调用的函数。

以下示意图演示了这种行为：



调用堆栈上的 FREObject 有效性

注：FREFunction 函数可以间接调用其他 FREFunction 函数。例如， FREFunctionA() 可以调用 ActionScript 对象的方法。该方法随后可以调用 FREFunctionB()。

因此，在使用 FREObject 变量时，应考虑以下事项：

- 传递给 FREFunction 函数的任何 FREObject 变量都只在调用堆栈上的第一个 FREFunction 函数返回之前有效。
- 任何本机函数使用扩展 C API 创建的任何 FREObject 变量都只在调用堆栈上的第一个 FREFunction 函数返回之前有效。
- 不能在另一线程中使用 FREObject 变量。只能在与本机函数用于接收或创建 FREObject 变量的同一线程中使用该变量。

- 例如，无法在调用 FREFunction 函数之间的这段时间内在全局数据中保存 FREObject 变量。由于在调用堆栈上的第一个 FREFunction 函数返回后，该变量将变得无效，因此保存的变量将没有用处。不过，您可以通过使用方法 FRESetContextActionScriptData() 保存对应的 ActionScript 对象。
- FREObject 变量失效后，对应的 ActionScript 对象仍然可以存在。例如，如果 FREObject 变量是 FREFunction 函数的返回值，其对应的 ActionScript 对象将仍然被引用。但是，一旦 ActionScript 端删除其引用，运行时将立即释放 ActionScript 对象。
- 不能在扩展之间共享 FREObject 变量，

  注：但可以在同一扩展的扩展上下文之间共享 FREObject 变量。不过，无论是哪种情况，FREObject 变量在调用堆栈上的第一个 FREFunction 函数返回到运行时后都变得无效。

使用 ActionScript 基元类型和对象

使用 ActionScript 基元类型

在本机函数中，输入参数可以与基元 ActionScript 类型对应。所有本机函数参数均为 FREObject 类型。因此，若要使用 ActionScript 基元类型输入参数，您应获取 FREObject 参数的 ActionScript 值。可将该值存储在对应的基元 C 数据类型变量中。使用以下 C API 函数：

- FREGetObjectAsInt32()

```
FREResult FREGetObjectAsInt32(FREObject object, int32_t *value);
```

- FREGetObjectAsUint32()

```
FREResult FREGetObjectAsUint32(FREObject object, uint32_t *value);
```

- FREGetObjectAsDouble()

```
FREResult FREGetObjectAsDouble(FREObject object, double *value);
```

- FREGetObjectAsBool()

```
FREResult FREGetObjectAsBool  (FREObject object, bool *value);
```

如果输出参数或返回值与基元 ActionScript 类型对应，则应使用 C API 函数创建 ActionScript 基元。应在 C 数据变量中提供指向 FREObject 对象的指针和基元的值。运行时将创建 ActionScript 基元并设置与其对应的 FREObject 变量。使用以下 C API 函数：

- FRENewObjectFromInt32()

```
FREResult FRENewObjectFromInt32(int32_t value, FREObject *object);
```

- FRENewObjectFromUint32()

```
FREResult FRENewObjectFromUint32(uint32_t value, FREObject *object);
```

- FRENewObjectFromDouble()

```
FREResult FRENewObjectFromDouble(double value, FREObject *object);
```

- FRENewObjectFromBool()

```
FREResult FRENewObjectFromBool  (bool value, FREObject *object);
```

使用 ActionScript String 对象

在本机函数中，输入参数可与 ActionScript String 类对象相对应。所有本机函数参数均为 FREObject 类型。因此，若要使用 ActionScript String 参数，您应获取 FREObject 参数的 ActionScript String 值。可将该值存储在对应的 C 字符串变量中。使用 C API 函数 FREGetObjectAsUTF8()：

```
FREResult FREGetObjectAsUTF8(
            FREObject       object,
            uint32_t*       length,
            const uint8_t** value
);
```

调用 FREGetObjectAsUTF8() 后， ActionScript String 值将位于 value 参数中， length 参数以字节为单位指示值字符串的长度。

如果输出参数或返回值与 ActionScript String 类对象对应，则应使用 C API 创建 ActionScript String 对象。应在 C 字符串变量中提供指向 FREObject 变量的指针和字符串值以及以字节为单位的长度。运行时将创建 ActionScript String 对象并设置与其对应的 FREObject 变量。使用 C API 函数 FRENewObjectFromUTF8()：

```
FREResult FRENewObjectFromUTF8(
            uint32_t        length,
            const uint8_t*  value,
            FREObject*      object
);
```

value 参数字符串值必须采用 UTF-8 编码，并包括 null 终止符。

注：任何 C API 函数的所有字符串参数均采用 UTF-8 编码并包括 null 终止符。

使用 ActionScript 类对象

在本机函数中，输入参数可与 ActionScript 类对象相对应。由于所有本机函数参数均为 FREObject 类型，因此 C API 提供了函数用于处理使用 FREObject 变量的类对象。

使用以下 C API 函数可获取和设置 ActionScript 类对象的属性：

- FREGetObjectProperty()

```
FREResult FREGetObjectProperty(
            FREObject object,
            const uint8_t*   propertyName,
            FREObject*       propertyValue,
            FREObject*       thrownException
);
```

- FRESetObjectProperty()

```
REResult FRESetObjectProperty(
            FREObject       object,
            const uint8_t*  propertyName,
            FREObject       propertyValue,
            FREObject*      thrownException
);
```

使用以下 C API 调用 ActionScript 类对象的方法：

FRECallObjectMethod()

```
FREResult FRECallObjectMethod(
            FREObject        object,
            const uint8_t*   methodName,
            uint32_t         argc,
            FREObject        argv[],
            FREObject*       result,
            FREObject*       thrownException
);
```

如果输出参数或返回值与 ActionScript 类对象对应，则应使用 C API 创建 ActionScript 对象。应提供指向 FREObject 变量的指针以及与 ActionScript 类构造函数的参数对应的 FREObject 变量。运行时将创建 ActionScript 类对象并设置与其对应的 FREObject 变量。使用以下 C API 函数：

FRENewObject()

```
FREResult FRENewObject(
        const uint8_t* className,
        uint32_t       argc,
        FREObject      argv[],
        FREObject*     object,
        FREObject*     thrownException
);
```

注：这些常规 ActionScript 对象处理函数适用于所有 ActionScript 类对象。但是， Array、 Vector、 ByteArray 和 BitmapData 这几个 ActionScript 类属于特例，因为它们每一个都涉及大量数据。因此， C API 提供了其他特定函数用于处理这些特例对象。

使用 ActionScript ByteArray 对象

使用 ActionScript ByteArray 类可高效地在扩展的 ActionScript 端和本机端之间传递大量字节。在本机函数中，输入参数、输出参数或返回值可以与 ActionScript ByteArray 类对象相对应。

与其他 ActionScript 类对象一样，FREObject 变量是 ActionScript ByteArray 对象的本机端表示形式。C API 提供了函数用于处理使用 FREObject 变量的 ByteArray 类对象。使用 FRESetObjectProperty()、 FREGetObjectProperty() 和 FRECallObjectMethod() 可获取并设置 ActionScript ByteArray 对象的属性并调用其方法。

但是，若要操作本机代码中 ByteArray 对象的字节，请使用 C API 函数 FREAcquireByteArray()。此方法访问在 ActionScript 端创建的 ByteArray 对象的字节：

```
FREResult FREAcquireByteArray(
            FREObject     object,
            FREByteArray* byteArrayToSet
);
// The type FREByteArray is defined as:

typedef struct {
            uint32_t length;
            uint8_t* bytes;
} FREByteArray;
```

处理完字节后，可使用 C API FREReleaseByteArray()：

```
FREResult FREReleaseByteArray( FREObject object );
```

注：不要在调用 FREAcquireByteArray() 和 FREReleaseByteArray() 之间调用任何 C API 函数。存在这种限制的原因是其他调用执行的代码会使指向字节数组内容的指针无效。

示例

此示例显示了扩展的 ActionScript 端创建 ByteArray 对象并初始化其字节的过程。然后调用一个本机函数以操作这些字节。

```
// ActionScript side of the extension

var myByteArray:ByteArray = new ByteArray();
myByteArray.writeUTFBytes("Hello, World");
myByteArray.position = 0;
myExtensionContext.call("MyNativeFunction", myByteArray);


// C code
FREObject MyNativeFunction(FREContext ctx, void* funcData, uint32_t argc, FREObject argv[]) {

    FREByteArray byteArray;
    int retVal;

    retVal = FREAcquireByteArray(argv[0], &byteArray);
    uint8_t* nativeString = (uint8_t*) "Hello from C";
    memcpy (byteArray.bytes, nativeString, 12);
    retVal = FREReleaseByteArray(argv[0]);

    return NULL;
}
```

使用 ActionScript Array 和 Vector 对象

使用 ActionScript Vector 和 Array 类可以高效地在扩展的 ActionScript 端和本机端传递数组。在本机函数中，输入参数、输出参数或返回值可以与 ActionScript Array 或 Vector 类对象相对应。

与其他 ActionScript 类对象一样， FREObject 变量是 ActionScript Array 或 Vector 对象的本机端表示形式。 C API 提供了函数用于处理使用 FREObject 变量的 Array 或 Vector 类对象。

使用以下 C API 函数获取和设置 Array 或 Vector 对象的长度：

- FREGetArrayLength()

```
FREResult FREGetArrayLength(
            FREObject  arrayOrVector,
            uint32_t*  length
);
```

- FRESetArrayLength()

```
FREResult FRESetArrayLength(
            FREObject  arrayOrVector,
            uint32_t   length
);
```

使用以下 C API 函数获取和设置 Array 或 Vector 对象的元素：

- FREGetArrayElementAt()

```
FREResult FREGetArrayElementAt(
            FREObject  arrayOrVector,
            uint32_t   index,
            FREObject* value
);
```

- FRESetArrayElementAt()

```
FREResult FRESetArrayElementAt(
        FREObject  arrayOrVector,
        uint32_t   index,
        FREObject  value
);
```

使用 ActionScript BitmapData 对象

使用 ActionScript BitmapData 类在扩展的 ActionScript 端和本机端传递位图。在本机函数中，输入参数、输出参数或返回值可以与 ActionScript BitmapData 类对象相对应。

与其他 ActionScript 类对象一样，FREObject 变量是 ActionScript BitmapData 对象的本机端表示形式。C API 提供了函数用于处理使用 FREObject 变量的 BitmapData 类对象。使用 FRESetObjectProperty()、FREGetObjectProperty() 和 FRECallObjectMethod() 可获取并设置 ActionScript ByteArray 对象的属性并调用其方法。

但是，若要操作本机代码中 BitmapData 对象的位，请使用 C API 函数 FREAcquireBitmapData() 或 FREAcquireBitmapData2()。这些方法访问在 ActionScript 端创建的 BitmapData 对象的位：

```
FREResult FREAcquireBitmapData(
            FREObject       object,
            FREBitmapData*  descriptorToSet
);
// The type FREBitmapData is defined as:

typedef struct {
    uint32_t    width;
    uint32_t    height;
    bool        hasAlpha;
    bool        isPremultiplied;
    uint32_t    lineStride32;
    uint32_t*   bits32;
} FREBitmapData;

//Or:
FREResult FREAcquireBitmapData2(
            FREObject       object,
            FREBitmapData2* descriptorToSet
);
// The type FREBitmapData is defined as:

typedef struct {
    uint32_t    width;
    uint32_t    height;
    bool        hasAlpha;
    bool        isInvertedY;
    bool        isPremultiplied;
    uint32_t    lineStride32;
    uint32_t*   bits32;
} FREBitmapData2;
```

FREBitmapData 或 FREBitmapData2 结构的所有字段均为只读字段。不过，bits32 字段指向实际的位图值，您可以在本机实现中修改这些值。向 AIR 3.1 中的 API 添加了 FREBitmapData2 结构和 FREAquireBitmapData2 函数。FREBitmapData2 包含一个额外字段 isInvertedY，该字段指示存储图像数据行的顺序。

若要指示本机实现已修改位图的全部或部分，请使位图矩形无效。使用 C API 函数 FREInvalidateBitmapDataRect()：

```
FREResult FREInvalidateBitmapDataRect(
        FREObject object,
        uint32_t x,
        uint32_t y,
        uint32_t width,
        uint32_t height
);
```

x 和 y 字段是要失效的矩形的坐标，相对于 0,0 坐标，0,0 是位图的左上角。width 和 height 字段是要失效的矩形的维度（以像素为单位）。

操作完位图后，请使用 C API 函数 FREReleaseBitmapData()：

```
FREResult FREReleaseBitmapData(FREObject object);
```

注：不要在调用 FREAcquireBitmapData() 和 FREReleaseBitmapData() 之间调用除 FREInvalidateBitmapDataRect() 之外的任何 C API 函数。存在这种限制的原因是其他调用执行的代码会使指向位图内容的指针无效。

线程和本机扩展

对本机实现进行编码时，应考虑以下事项：

- 运行时可以通过不同的线程并行调用扩展上下文的 FREFunction 函数。
- 运行时可以通过不同的线程并行调用不同扩展上下文的 FREFunction 函数。

因此，应相应地对本机实现进行编码。例如，如果使用全局数据，则应通过某种形式的锁保护该数据。

注：本机实现可以选择创建单独的线程。如果这样做，应考虑 第 19 页的 “FREObject 有效性 ” 中指定的限制。

第 4 章 : 使用 Java 语言编写本机端代码

Android 设备使用 Java 作为主要应用程序开发语言。如果您要将本机扩展的目标设定为此类设备，应使用本机扩展 Java API 编写扩展的 “ 本机 ” 端代码。您还可以将 AIR 扩展 C API 与 Android 本机开发工具包结合使用，以用于某些特定目的。有关使用 C API 的信息，请参阅第 13 页的 “ 使用 C 语言编写本机端代码 ”。

文件 FlashRuntimeExtensions.jar 中提供了 Java API。可在 AIR SDK 的 lib/android 目录找到此文件。可从 http://www.adobe.com/cn/products/air/sdk/ 获取 AIR SDK。

若要编写 Adobe AIR 本机扩展的 Java 端代码，请执行以下操作：

- 实现 FREExtension 接口。
- 使用一个或多个子类扩展 FREContext 抽象类。
- 为每个能从扩展的 ActionScript 端调用的 Java 函数实现 FREFunction 接口。

有关每个 Java API 函数的详细信息 （比如参数和返回值），请参阅 第 102 页的 “Android Java API 参考 ”。

实现 FREExtension 接口

每个使用 Java API 的扩展都必须实现 FREExtension 接口。此 FREExtension 实例是您扩展中的 Java 代码的初始入口点。在扩展描述符文件的 <initializer> 元素中指定类的完全限定名。 Java 实现只能用于 Android-ARM 平台。请参阅第 63 页的 “ 本机扩展描述符文件 ”。

createContext() 方法是 FREExtension 实现的最重要组成部分。当扩展的 ActionScript 代码调用 ExtensionContext.createExtensionContext() 时，AIR 运行时调用 createContext() 方法。此方法必须返回一个 FREContext 类的实例。 ActionScript createExtensionContext() 方法具有一个字符串参数，该传输传递给 Java createContext() 函数。您可使用此值针对不同用途提供不同上下文。如果您的扩展只使用一个上下文类，可忽略此参数。

FREExtension 接口的其他参数 initialize() 和 dispose() 由运行时自动调用，可用于创建和清除扩展所需的任何永久性资源。但是，并不是每个扩展都需要执行这些函数的操作。

FREExtension 实现类的构造函数不得带有任何参数。

当 ActionScript 代码第一次调用 createExtensionContext() 时， AIR 运行时将实例化 FREExtension 实例。 Java 扩展类的调用顺序如下：

- FREExtension 实现类构造函数
- initialize()
- createContext()

FREExtension 示例

下面的示例演示一个简单的 FREExtension 实现。此示例使用一个扩展上下文。AIR 运行时第一次调用 createContext() 方法时创建上下文引用。保存该引用以供后续使用。

```
package com.example;

import android.util.Log;

import com.adobe.fre.FREContext;
import com.adobe.fre.FREExtension;

public class Extension implements FREExtension {
    private static final String EXT_NAME = "Extension";
    private ExtensionContext context;
    private String tag = EXT_NAME + "ExtensionClass";

    public FREContext createContext(String arg0) {
        Log.i(tag, "Creating context");
        if( context == null) context = new ExtensionContext(EXT_NAME);
        return context;
    }

    public void dispose() {
        Log.i(tag, "Disposing extension");
        // nothing to dispose for this example
    }

    public void initialize() {
        Log.i(tag, "Initialize");
        // nothing to initialize for this example
    }
}
```

ExtensionContext 示例

下面的示例演示扩展中向应用程序代码公开本机扩展功能的 ActionScript 代码。

```
package com.example
{
import flash.external.ExtensionContext;

public class ExampleExtension
{
    private const extensionID:String = "com.example.Extension";
    private var extensionContext:ExtensionContext;

    public function ExampleExtension()
    {
        extensionContext = ExtensionContext.createExtensionContext( extensionID, null );
    }

    public function usefulFunction( value:Boolean ):Boolean
    {
        var retValue:Boolean = false;
        retValue = extensionContext.call( "usefulFunctionKey", value );
        return retValue;
    }
}
}
```

应用程序代码示例

若要使用本机扩展，应用程序需访问扩展提供的 ActionScript 类和方法。（同样，应用程序还可能访问任何其他 ActionScript 库的类和方法。）

```
var exampleExtension:ExampleExtension = new ExampleExtension();

var input:Boolean = true;
var untrue:Boolean = exampleExtension.usefulFunction( input );
```

扩展 FREContext 类

FREContext 对象提供一组 Java 函数和上下文特定状态。您的扩展必须至少提供一个 FREContext 类的具体实现。

FREContext 类定义两个您必须实现的抽象方法：

- getFunctions() — 必须返回一个 Map 对象， AIR 运行时使用该对象查找上下文提供的函数。
- dispose() — 当可以清除上下文资源时，由运行时调用此函数。

释放上下文

扩展的 ActionScript 端可以调用 ExtensionContext 实例的 dispose() 方法。调用 ActionScript dispose() 方法将导致运行时调用 FREContext 类的 Java dispose() 方法。

如果 ActionScript 端不调用 dispose()，当不再存在对 ExtensionContext 实例的引用时，运行时垃圾回收器将释放该实例。那时，运行时将调用 FREContext 类的 Java dispose() 方法。

FREContext 示例

下面的示例演示一个简单的 FREContext 实现。此类创建一个函数映射，其中包含一个函数。

```
package com.example;

import java.util.HashMap;
import java.util.Map;

import android.util.Log;

import com.adobe.fre.FREContext;
import com.adobe.fre.FREFunction;

public class ExtensionContext extends FREContext {
    private static final String CTX_NAME = "ExtensionContext";
    private String tag;

    public ExtensionContext( String extensionName ) {
        tag = extensionName + "." + CTX_NAME;
        Log.i(tag, "Creating context");
    }

    @Override
    public void dispose() {
        Log.i(tag, "Dispose context");
    }

    @Override
    public Map<String, FREFunction> getFunctions() {
        Log.i(tag, "Creating function Map");
        Map<String, FREFunction> functionMap = new HashMap<String, FREFunction>();

        functionMap.put( UsefulFunction.KEY, new UsefulFunction() );
        return functionMap;
    }

    public String getIdentifier() {
        return tag;
    }
}
```

实现 FREFunction 接口

AIR 运行时使用 FREFunction 接口从 ActionScript 代码调用 Java 函数。实现此接口以公开具体功能项。

FREFunction 定义一个方法，即 call()。当扩展中的 ActionScript 代码调用 ExtensionContext 类的 call() 方法时， AIR 运行时调用此方法。若要查找正确的 FREFunction 实例，运行时可在 FREContext 函数映射中查找。ActionScript 代码使用参数的常规 ActionScript 类型和类将参数队列传递给方法。Java 代码将这些值视为 FREObject 实例。同样，Java call() 方法返回一个 FREObject 实例， ActionScript 代码将此返回值视为 ActionScript 类型。

在实现 FREFunction 类时，应决定 call() 方法参数队列中的参数顺序。由于您同时还编写 ActionScript 端，并在 ExtensionContext 实例的 call() 方法中使用该相同参数顺序，因此，尽管每个本机 Java 函数参数都是 FREObject 类型，但应知道其对应的 ActionScript 类型。

同样，需要决定 Java 函数的返回值 （如果有）的 ActionScript 类型。 call() 方法返回此类型的对象。尽管 Java 函数返回值始终是一个 FREObject 实例，但应知道其对应的 ActionScript 类型。

由 FREFunction 对象创建的或传递给它的 FREObject 实例具有有限的有效寿命。在两次函数调用之间，请勿保存对 FREObject 的引用。

FREFunction 示例

下面的示例演示一个简单的 FREFunction 实现。此函数带有一个布尔型参数并返回其取反值。

```
package com.example;

import com.adobe.fre.FREContext;
import com.adobe.fre.FREFunction;
import com.adobe.fre.FREObject;

import android.util.Log;

public class UsefulFunction implements FREFunction {
    public static final String KEY = "usefulFunctionKey";
    private String tag;

    public FREObject call(FREContext arg0, FREObject[] arg1) {
        ExtensionContext ctx = (ExtensionContext) arg0;
        tag = ctx.getIdentifier() + "." + KEY;
        Log.i( tag, "Invoked " + KEY );
        FREObject returnValue = null;

        try {
            FREObject input = arg1[0];
            Boolean value = input.getAsBool();
            returnValue = FREObject.newObject( !value );//Invert the received value

        } catch (Exception e) {
            Log.d(tag, e.getMessage());
            e.printStackTrace();
        }
        return returnValue;
    }
}
```

调度异步事件

Java C 代码可以将异步事件调度回扩展的 ActionScript 代码。例如，扩展方法可启动另一线程来执行某项任务。当另一线程中的任务结束后，该线程调用相应 FREContext 的 dispatchStatusEventAsync() 方法。在 ActionScript 端，与该 Java FREContext 关联的 ActionScript ExtensionContext 实例调度一个 Status 事件。

访问 ActionScript 对象

一个 FREObject 类实例表示一个 ActionScript 类对象或基元类型。在 Java 实现中使用 FREObject 实例操作 ActionScript 数据。 FREObject 的一个主要用途是用于 Java 函数参数和返回值。

FREObject 类提供函数用于访问关联的 ActionScript 类对象或基元值。所用的 Java API 取决于 ActionScript 对象的类型。这些类型可以是：

- ActionScript 基元数据类型
- ActionScript 类对象
- ActionScript String 对象
- ActionScript Array 或 Vector 类对象
- ActionScript ByteArray 类对象

- ActionScript BitmapData 类对象

重要说明：您只能在正在运行 “ 所拥有的 ”FREFunction 函数的同一线程中访问 FREObject。此外，如果您已调用任何 FREByteArray 或 FREBitmapData 对象的 acquire() 方法，则您在调用此原始 FREByteArray 或 FREBitmapData 对象的 release() 方法之前，无法访问任何对象的 ActionScript 方法。如果 “ 获取 ” 了任何对象，调用该对象或其他 FREObject 的 getProperty()、 setProperty() 或 callMethod() 将引发 IllegalStateException 异常。

FREObject 有效性

如果试图在 Java API 调用中使用无效 FREObject， Java API 将引发异常。

任何 FREObject 实例只在调用堆栈上的第一个 FREFunction 函数返回之前有效。调用堆栈上的第一个 FREFunction 函数是运行时由于 ActionScript 端调用 ExtensionContext 实例的 call() 方法而调用的函数。FREObject 也只在运行时用来调用第一个 FREFunction 的线程中有效。

注：FREFunction 函数可以间接调用其他 FREFunction 函数。例如， FREFunctionA() 可以调用 ActionScript 对象的方法。该方法随后可以调用 FREFunctionB()。

因此，在使用 FREObject 时，应考虑以下事项：

- 传递给 FREFunction 实例的任何 FREObject 只在调用堆栈上的第一个 FREFunction 函数返回之前有效。
- 任何 Java 函数创建的任何 FREObject 只在调用堆栈上的第一个 FREFunction 返回之前有效。
- 不能在另一线程中使用 FREObject。只能在与 Java 函数用于接收或创建 FREObject 对象的同一线程中使用该对象。
- 例如，在两次调用 FREFunction 函数之间，无法在全局数据中保存 FREObject 引用。由于在调用堆栈上的第一个 FREFunction 函数返回后，该对象将变得无效，因此保存的引用将没有用处。但是，在两次函数调用之间，可以使用 FREContext 类的 setActionScriptData() 方法保存数据。
- FREObject 失效后，对应的 ActionScript 对象仍然可以存在。例如，如果 FREObject 是 FREFunction 函数的返回值，其对应的 ActionScript 对象将仍然被引用。但是，一旦 ActionScript 端删除其引用，运行时将立即释放 ActionScript 对象。
- 在扩展间无法共享 FREObject。

  注：但可以在同一扩展的扩展上下文之间共享 FREObject。不过即使这样，无论是哪种情况，在调用堆栈上的第一个 FREFunction 函数返回到运行时后， FREObject 仍将变得无效。

使用 ActionScript 基元类型和对象

使用 ActionScript 基元类型

在 Java FREFunction 实现中，输入参数可与 ActionScript 基元类型相对应。从 ActionScript 代码传递到扩展的所有参数均为 FREObject 类型。因此，若要使用 ActionScript 基元类型输入参数，您应获取 FREObject 参数的 ActionScript 值。使用以下 FREObject 函数：

- getAsInt()
- getAsDouble()
- getAsString()
- getAsBool()

如果 FREFunction 的输出参数或返回值对应 ActionScript 基元类型，应使用一个 FREObject 工厂方法创建 ActionScript 基元。使用以下静态 FREObject 函数：

- FREObject.newObject(int value)

- FREObject.newObject(double value)
- FREObject.newObject(String value)
- FREObject.newObject(int boolean)

使用 ActionScript 类对象

在 FREFunction 实现中，输入参数可与 ActionScript 类对象相对应。 FREObject 类提供了以下函数用于访问对象的 ActionScript 定义的属性和方法：

- getProperty(String propertyName)
- setProperty(String propertyName, FREObject value)
- callMethod(String propertyName, FREObject[] methodArgs)

如果输出参数或返回值对应 ActionScript 类对象，应使用静态 FREObject 工厂方法创建 ActionScript 对象：

```
FREObject newObject ( String className, FREObject constructorArgs[] )
```

注：这些常规 ActionScript 对象处理函数适用于所有 ActionScript 类对象。但是， Array、 Vector、 ByteArray 和 BitmapData 这几个 ActionScript 类属于特例，因为它们每一个都涉及大量数据。因此， Java API 提供了附加特定类用于操作这些特殊情况的对象。

使用 ActionScript ByteArray 对象

使用 ActionScript ByteArray 类可高效地在扩展的 ActionScript 端和 Java 端之间传递大量字节。 FREByteArray 类使用以下用于操作字节数组对象的方法扩展 FREObject：

- acquire()
- getLength()
- getBytes()
- release()

若要操作 Java 代码中的 ByteArray 对象的字节，请使用 FREByteArray acquire() 方法，接着再使用 getBytes() 方法。操作完字节后，请使用 FREByteArray release() 方法。

在调用 acquire() 和 release() 之间，请勿调用 （任何对象，而不仅仅是获取的对象的）任何 FREObject 方法。存在这种限制的原因是其他调用执行的代码会产生副作用，会使字节数组对象或其内容无效。

复制到所获取 FREByteArray 对象的 bytes 字段中的字节数请勿超过 length 字段指定的数值。ActionScript 端将忽略任何多余字节。

可使用静态工厂方法 FREByteArray.newByteArray() 新建 FREByteArray 对象。新字节数组为空。若要从 Java 向其添加数据，必须首先设置 ActionScript length 属性：

```
FREByteArray freByteArray = freByteArray = FREByteArray.newByteArray();
freByteArray.setProperty( "length", FREObject.newObject( 12 ) );
freByteArray.acquire();
ByteBuffer bytes = freByteArray.getBytes();
//set the data
freByteArray.release();
```

使用 ActionScript Array 和 Vector 对象

使用 ActionScript Vector 和 Array 类可以高效地在扩展的 ActionScript 端和 Java 端传递数组。

FREArray 类使用以下适用于操作数组和矢量对象的方法扩展 FREObject：

- getLength()
- setLength(long length)
- getObjectAt(long index)
- setObjectAt(long index, FREObject value)

FREArray 还定义工厂方法用于创建数组和矢量：

- FREArray newArray(int numElements)
- FREArray newArray(String classname, int numElements, boolean fixed)

使用 ActionScript BitmapData 对象

使用 ActionScript BitmapData 类在扩展的 ActionScript 端和 Java 端传递位图。 FREBitmapData 类使用以下适用于操作位图数据对象的方法扩展 FREObject：

- acquire()

  在调用以下任何其他方法之前，必须先调用 acquire()：

- getHeight()
- getWidth()
- hasAlpha()
- isInvertedY() (AIR 3.1)
- isPremultiplied()
- getLineStride32()
- getBits()
- invalidateRect()
- release()

若要更改位图数据，首先应调用 acquire()，然后再使用 getBits() 方法获取像素颜色数据。 getLineStride32() 方法指示每条水平线中有多少 32 位数据。尽管此值通常等于位图宽度，但是某些情况下，位图数据可能会被额外、不可见的字节填充。位图中的每个像素都是 ARGB 格式的 32 位值。操作完位图后，应调用 release() 释放位图数据。

在调用 acquire() 和 release() 之间，请勿调用 （任何对象，而不仅仅是获取的对象的）任何 FREObject 方法。存在这种限制的原因是其他调用执行的代码会产生副作用，会使字节数组对象或其内容无效。

若要指示 Java 实现已修改位图的全部或部分，请使用 invalidateRect() 函数使位图矩形无效。 x 和 y 参数是要失效的矩形的坐标，相对于位图左上角的 0,0 坐标而言。 width 和 height 字段是要失效的矩形的维度 （以像素为单位）。

可使用静态工厂方法 FREBitmapData.newBitmapData() 新建 FREBitmapData 对象。

线程和本机扩展

编写 Java 实现代码时，应考虑以下事项：

- 运行时可以通过不同的线程并行调用扩展上下文的 FREFunction 函数。
- 运行时可以通过不同的线程并行调用不同扩展上下文的 FREFunction 函数。

因此，应相应地编写 Java 实现代码。例如，如果使用全局数据，则应通过某种形式的锁保护该数据。

注：Java 实现可以选择创建单独的线程。如果这样做，应考虑 第 31 页的 “FREObject 有效性 ” 中指定的限制。

第 5 章：打包本机扩展

要将您的本机扩展提供给应用程序开发人员，需要将所有相关的文件打包到一个 ANE 文件中。 AIR 应用程序开发人员通过以下方式使用 ANE 文件：

- 将 ANE 文件包括在应用程序的库路径中，方式与开发人员将 SWC 文件包括在库路径中一样。此操作允许应用程序引用扩展的 ActionScript 类。
- 将 ANE 文件与 AIR 应用程序一起打包。

使用 AIR 应用程序构建和打包 ANE 文件的信息，请参阅使用 Adobe AIR 的本机扩展。

创建本机扩展包涉及以下任务：

1. 将扩展的 ActionScript 库构建到 SWC 文件中。
2. 构建扩展的本机库 -- 每个支持的目标平台一个。
3. 为您的扩展创建签名证书。对扩展签名是可选的。
4. 创建扩展描述符文件。
5. 使用 ADT 创建扩展包。

构建本机扩展的 ActionScript 库

将扩展的 ActionScript 端构建到 SWC 文件中。SWC 文件是一个 ActionScript 库，其中包含您的 ActionScript 类和其他资源（如其图像和字符串）的存档文件。

在打包本机扩展时，您需要 SWC 文件和一个单独的 library.swf 文件，后者可从 SWC 文件中提取。SWC 文件提供有关创作和编译的 ActionScript 定义。 library.swf 提供特定平台所使用的 ActionScript 实现。如果扩展的不同目标平台需要不同的 ActionScript 实现，请创建多个 SWC 库并为每个平台分别提取 library.swf 文件。不过，最佳做法是所有 ActionScript 实现都有相同的公共接口。（ANE 包中只能包含一个 SWC 文件。）

SWC 文件包含一个名为 library.swf 的文件。有关详细信息，请参阅 第 43 页的 “ANE 包中的 SWC 文件和 SWF 文件 ”。

使用下列方式之一构建 SWC 文件：

- 使用 Adobe Flash Builder 创建 Flex 库项目。

  在构建 Flex 库项目时， Flash Builder 创建一个 SWC 文件。请参阅创建 Flex 库项目。

  在创建 Flex 库项目时，确保选择该选项以包括 Adobe AIR 库。

  确保将 SWC 编译为正确版本的 SWF 格式。针对 AIR 2.7 使用 SWF 11、针对 AIR 3 使用 SWF 13、针对 AIR 3.1 使用 SWF 14 等等。您可以在项目的属性中设置 SWF 文件格式版本。选择 ActionScript 编译器并输入以下附加编译器参数：

  ```
  -swf-version 17
  ```

  注： 可以使用 Flex SDK bin 目录中的 swfdump 来检查任何 SWF 文件的 SWF 文件格式版本：swfdump myFlexLibraryProjectSWF.swf

- 使用命令行工具 acompc 为 AIR 生成 Flex 库项目。此工具是随 Flex SDK 提供的组件编译器。如果没有使用 Flash Builder，可以直接使用 acompc。请参阅使用组件编译器 compc。

  例如：

```
acompc -source-path $HOME/myExtension/actionScript/src
                                -include-classes sample.extension.MyExtensionClass
sample.extension.MyExtensionHelperClass
                                -swf-version=13
                                -output $HOME/myExtension/output/sample.extension.myExtension.swc
```

注：如果您的 ActionScript 库使用任何外部资源，请使用 ADT 将它们打包到 ANE 文件中。请参阅第 42 页的 “ 创建本机扩展包 ”。

SWF 版本兼容性

编译 ActionScript 库时指定的 SWF 版本是决定扩展是否与 AIR 应用程序兼容的一个因素，另一个因素是扩展描述符命名空间。扩展的 SWF 版本不能超过主应用程序 SWF 文件的 SWF 版本：

兼容的 AIR 应用程序版本	ANE SWF 版本	扩展命名空间
3.0+	10-13	ns.adobe.com/air/extension/2.5
3.1+	14	ns.adobe.com/air/extension/3.1
3.2+	15	ns.adobe.com/air/extension/3.2
3.3+	16	ns.adobe.com/air/extension/3.3
3.4+	17	ns.adobe.com/air/extension/3.4
3.5+	18	ns.adobe.com/air/extension/3.5
3.6+	19	ns.adobe.com/air/extension/3.6
3.7+	20	ns.adobe.com/air/extension/3.7

注：平台选项 (platform.xml) 文件需要 ns.adobe.com/air/extension/3.1 或更高版本的命名空间。如果您使用 -platformoptions 标志来打包 ANE，则必须指定 ns.adobe.com/air/extension/3.1 或更高版本以及高于或等于 14 的 SWC 版本。某些平台选项文件功能需要更高版本的 AIR 命名空间和 SWF。

为本机扩展创建签名证书

您可以选择对本机扩展进行数字签名。对扩展签名是可选的。

使用公认的证书颁发机构 (CA) 颁发的证书对本机扩展进行数字签名，可以向 AIR 应用程序开发人员提供重要保证：

- 他们使用其应用程序打包的本机扩展没有被意外或恶意更改。
- 您是本机扩展的签名者 （发布者）。

注：在打包 AIR 应用程序时，应用程序的用户看到的是 AIR 应用程序的证书，而不是本机扩展的证书 （如果提供）。

从证书颁发机构获取证书。为本机扩展创建数字签名证书与为 AIR 应用程序创建证书相同。请参阅构建 Adobe® AIR® 应用程序中的对 AIR 应用程序签名。

创建扩展描述符文件

每个本机扩展都包含一个扩展描述符文件。此 XML 文件指定有关该扩展的信息，如扩展描述符、名称、版本号以及该扩展可以运行的平台。

创建扩展时，应根据详细架构第 63 页的 “ 本机扩展描述符文件 ”编写扩展描述符文件。

例如：

```
<extension xmlns="http://ns.adobe.com/air/extension/3.5">
                    <id>com.example.MyExtension</id>
                    <versionNumber>0.0.1</versionNumber>
                    <platforms>
                    <platform name="Android-ARM">
                    <applicationDeployment>
                    <nativeLibrary>MyExtension.jar</nativeLibrary>
                    <initializer>com.sample.ext.MyExtension</initializer>
                    </applicationDeployment>
                    </platform>
                    <platform name="iPhone-ARM">
                    <applicationDeployment>
                    <nativeLibrary>MyExtension.a</nativeLibrary>
                    <initializer>MyExtensionIntializer</initializer>
                    </applicationDeployment>
                    </platform>
                    <platform name="default">
                    <applicationDeployment/>
                    </platform>
                    </platforms>
                    </extension>
```

在创建扩展描述符文件时，请考虑以下信息。

扩展 ID

<id> 元素的值与以下情况中使用的值相同：

- ActionScript 对 CreateExtensionContext() 的调用。
- 使用扩展的应用程序的应用程序描述符文件中的 extensionID 元素。

有关命名扩展 ID 的最佳做法，请参阅第 7 页的 “ 扩展 ID”。

版本号

<versionNumber> 元素的值用于指定扩展的版本。版本号的一个重要用途是用于保持设备绑定扩展的向后兼容性。请参阅第 11 页的 “ 本机扩展向后兼容性 ”。

平台

您还可以如第 4 页的 “ 面向多种平台 ” 中所述编写面向多种平台的本机扩展。

如第 4 页的 “ 扩展在运行时的提供方式 ” 中所述，扩展可以是应用程序绑定或设备绑定的，具体取决于平台。

对于每个目标平台，在扩展描述符文件中提供一个 <platform> 元素。 <platform> 元素的 name 属性指定目标平台，如 iPhone-ARM 或 Windows-x86。应用程序绑定扩展还可以指定 default 作为 name 属性的值。此值指示该扩展是一个仅 ActionScript 扩展；该扩展没有本机代码库。

当使用应用程序绑定扩展的 AIR 应用程序运行时， AIR 执行以下操作：

- AIR 加载扩展库，扩展库的扩展描述符文件与设备的平台对应的平台名称关联。
- 如果没有与设备对应的平台名称， AIR 将加载扩展描述符文件与默认平台关联的扩展库。

描述符命名空间

描述符文件的根 <extension> 元素中指定的命名空间决定扩展所需的 AIR SDK 版本。命名空间只是其中一个因素，它与 SWF 版本一起决定能否在 AIR 应用程序中使用扩展。 AIR 应用程序描述符命名空间必须大于或等于扩展描述符命名空间。

扩展命名空间值	兼容的 AIR 版本	ANE SWF 版本
ns.adobe.com/air/extension/2.5	AIR 3+	13
ns.adobe.com/air/extension/3.1	AIR 3.1+	14

扩展命名空间值	兼容的 AIR 版本	ANE SWF 版本
ns.adobe.com/air/extension/3.2	AIR 3.2+	15
ns.adobe.com/air/extension/3.3	AIR 3.3+	16
ns.adobe.com/air/extension/3.4	AIR 3.4+	17
ns.adobe.com/air/extension/3.5	AIR 3.5+	18
ns.adobe.com/air/extension/3.6	AIR 3.6+	19
ns.adobe.com/air/extension/3.7	AIR 3.7+	20

注：平台选项 (platform.xml) 文件需要 ns.adobe.com/air/extension/3.1 或更高版本的命名空间。如果您使用 -platformoptions 标志来打包 ANE，则必须指定 ns.adobe.com/air/extension/3.1 或更高版本以及高于或等于 14 的 SWC 版本。某些平台选项文件功能需要更高版本的 AIR 命名空间和 SWF。

构建本机库

要构建您的扩展的本机库，请使用适用于目标设备的开发环境。例如：

- 在使用 Android SDK 开发时，请使用针对 Eclipse 集成开发环境 (IDE) 的 Android 开发工具插件。
- 在为 iOS 设备和 Mac OS X 设备开发时，请使用 Apple 的 Xcode IDE。
- 在为 Windows 设备开发时，可以使用 Microsoft Visual Studio。

有关使用这些开发环境的本机扩展示例，请参阅 Adobe AIR 的本机扩展。

将扩展的本机端构建到库而非应用程序中。在将您的本机扩展打包到 ANE 文件时，需要指定本机库。

Android 本机库

在使用 Android SDK 时，请作为 JAR 文件提供库。

在使用 Android NDK 时，请提供具有如下文件名的共享库：

```
lib<yourlibraryname>.so
```

需要此共享库命名约定才能正确安装应用程序 APK 包。

创建包含共享库的 ANE 包时，共享库必须存储在以下目录结构中：

```
<Android platform directory>/
                        libs/
                        armeabi/
                        <Android emulator native libraries>
                        armeabi-v7a/
                        <Android device native libraries>
```

iOS 本机库

提供文件扩展名为 .a 的静态库。您可以使用 Xcode IDE 中的 Cocoa Touch 静态库模板构建 .a 文件。使用目标类型 device 创建在设备上运行的本机库；使用目标类型 simulator 创建在 iOS Simulator 上运行的本机库 （AIR 3.3 和更高版本支持 iOS Simulator）。

每个版本的 AIR 绑定一个版本的 iOS SDK。在目标为相应 AIR 版本的本机扩展中，您可以链接到该版本 iOS SDK 中提供的任何公共框架。下表列出了 AIR SDK 版本及其绑定的 iOS 版本，以及对其他功能的支持：

AIR SDK	包含的 iOS SDK	链接到其他库	绑定第三方库
3.0 - 3.2	4.2	否	否
3.3 - 3.4	5.1	是	否
3.5	6.0	是	是

将特定 AIR SDK 版本作为扩展的目标时，不应使用在相应 iOS 版本之后引入的任何 iOS 框架。除了作为目标的 AIR SDK，不应使用任何其他的共享库或第三方框架。

如果不使用与 AIR SDK 绑定的 iOS SDK，一种替代方法是，在 AIR 3.3 及更高版本中您可以链接到外部 iOS SDK。使用 ADT -platformsdk 开关，指定指向外部 iOS SDK 的路径。

默认情况下， AIR 链接到以下 iOS 框架库：

AudioToolbox	CoreLocation	OpenGLES
AVFoundation	CoreMedia	QuartzCore
CFNetwork	CoreVideo	Security
CoreFoundation	Foundation	SystemConfiguration
CoreGraphics	MobileCoreServices	UIKit
GameController	AssetsLibrary	

当链接到其他框架和库时，请在平台选项 XML 文件中指定链接选项。

注：在 AIR 3.4 和更高版本中，您可以使用 ADT -hideAneSymbols yes 选项来消除潜在符号冲突。有关详细信息，请参阅本机扩展选项。

iOS 平台选项 (platform.xml) 文件

您可以使用平台选项 (platform.xml) 文件来指定特定于 iOS 的选项以链接到其他框架和库，或者在您的本机扩展中绑定第三方框架或库。当您在 -platform 标志之后为 iOS 指定 -platformoptions 标志，以此来打包扩展时，平台选项文件即添加到 ANE。稍后，当开发人员创建使用您的扩展的应用程序 IPA 文件时， ADT 会使用 platform.xml 文件中的选项链接到其他库并包含绑定的依赖项。

注：平台选项文件的名称可以任意，不要求将其命名为 “platform.xml”。

您可以将 iOS 平台选项文件用于 iPhone-ARM （设备）和 iPhone-x86 （iOS 模拟器）平台。

platform 选项文件需要 AIR 3.1 或更高版本。

使用 packagedDependencies 功能打包用于 iOS 的 ANE 时，添加

```
<option>-rpath @executable_path/Frameworks</option>
```

在 linkerOptions 标签内 （platformoptions.xml 中）。

iOS 链接器选项

iOS 链接器选项为您提供一种将任意选项传递给链接器的方式。指定选项在传递给链接器时会保持原样。您可以使用这种方式链接到其他框架和库，例如其他 iOS 框架。要指定链接器选项，请在平台选项 xml 文件中使用 <linkerOptions> 标签。在 <linkerOptions> 标签内，指定包含在 <options> 标签对中的每个链接选项，如下例所示：

```
<linkerOptions>
                                <option>-ios_version_min 5.0</option>
                                <option>-framework Accelerate</option>
                                <option>-liconv</option>
                                </linkerOptions>
```

任何以此方式链接的依赖项必须分发给使用您的本机扩展从本机扩展本身分离的开发人员。这对于 iOS 包含的而非 AIR 默认链接的其他库最为有用。您还可以使用此选项让开发人员链接到单独提供给他们的静态库。

<linkerOptions> 标签需要 AIR 3.3 或更高版本。

打包的第三方依赖项

有时，您希望在本机扩展中使用静态库 （例如，本机第三方库），而无需访问该库的源代码，或者无需开发人员访问从您的扩展分离的库。您可以通过将静态库指定为打包的依赖项，将其绑定到您的本机扩展。在 platform 选项 xml 文件中使用 <packagedDependencies> 标签。对于每个要包含在扩展包中的依赖项，指定以 <packagedDependency> 标签对括起的名称或相对路径，如下例所示：

```
<packagedDependencies>
                                <packagedDependency>foo.a</packagedDependency>
                                <packagedDependency>abc/x.framework</packagedDependency>
                                <packagedDependency>lib.o</packagedDependency>
                                </packagedDependencies>
```

打包的依赖项应当是具有以下扩展之一的静态库：.a、.framework 或 .o。该库应支持在设备上使用 ARMv7 体系结构，在 iOS 模拟器上使用 i386 体系结构。

打包本机扩展时，您必须将依赖项的名称指定为 -platformoptions 标志的参数。在 platform.xml 文件的文件名之后、任何后续的 -package 标志之前列出依赖项，如下例所示：

```
adt -package <signing options> -target ane MyExtension.ane MyExt.xml -swc MyExtension.swc
                                -platform iPhone-ARM -platformoptions platformiOSARM.xml
                                foo.a abc/x.framework lib.o -C platform/ios .
                                -platform iPhone-x86 -platformoptions platformiOSx86.xml
                                -C platform/iosSimulator
                                -platform default -C platform/default library.swf
```

<packagedDependencies> 标签需要 AIR 3.5 或更高版本。

使用专用的嵌入框架

要创建具有 Xcode6 或更高版本的 iOS ANE，可使用专用的框架：

1 编辑 platform.xml，添加

```
<option>-rpath @executable_path/Frameworks</option>
```

在 linkerOptions 标签内。

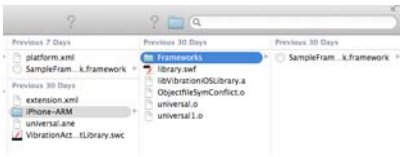
例如：

```
</description>
                                        <linkerOptions>
                                        <option>-ios_version_min 5.1.1</option>
                                        <option>-rpath @executable_path/Frameworks</option>
                                        </linkerOptions>
                                        <packagedDependencies>

<packagedDependency>SampleFramework.framework</packagedDependency>
                                        </packagedDependencies>
```

2 在现有的 iPhone-ARM 文件夹中创建一个名为 Frameworks 的文件夹。

3 将专用框架复制到 Frameworks 文件夹，并将该框架与 ANE 打包在一起。



复制专用框架

platform 选项 **(platform.xml)** 文件示例

以下列表显示了一个平台选项 (platform.xml) 文件结构的示例：

```
<platform xmlns="http://ns.adobe.com/air/extension/3.5">
                                <description>An optional description.</description>
                                <copyright>2011 (optional)</copyright>
                                <sdkVersion>5.0.0</sdkVersion>
                                <linkerOptions>
                                <option>-ios_version_min 5.0</option>
                                <option>-framework Accelerate</option>
                                <option>-liconv</option>
                                </linkerOptions>
                                <packagedDependencies>
                                <packagedDependency>foo.a</packagedDependency>
                                <packagedDependency>abc/x.framework</packagedDependency>
                                <packagedDependency>lib.o</packagedDependency>
                                </packagedDependencies>
                                </platform>
```

要在本机扩展包中包含此平台选项文件，您可以使用如下 ADT 命令：

```
adt -package <signing options> -target ane MyExtension.ane MyExt.xml -swc MyExtension.swc
                                -platform iPhone-ARM -platformoptions platformiOSARM.xml
                                foo.a abc/x.framework lib.o -C platform/ios .
                                -platform iPhone-x86 -platformoptions platformiOSx86.xml
                                -C platform/iosSimulator
                                -platform default -C platform/default library.swf
```

Mac OS X 本机库

对于 Mac OS X 设备，请提供一个 .framework 库。在构建库时，确保 Xcode 项目的基本 SDK 设置为 Mac OS X 10.5 SDK。

在所有情况下，在编译 Mac OS X 框架以用作扩展时，均应设置以下选项以便正确解析 AIR 框架的依赖项：

- 向 LD_RUNPATH_SEARCH_PATHS 中添加 @executable_path/../runtimes/air/mac、 @executable_path/../Frameworks 和 /Library/Frameworks。
- 使用弱框架链接和平面命名空间选项。

这些选项设置加在一起将允许应用程序首先加载 AIR 框架的正确副本，然后扩展就可以依靠已加载的副本。

Windows 本机库

对于 Windows 设备，请作为 DLL 文件提供库。将 lib/windows 目录的 AIR SDK 目录中的库 FlashExtensions.lib 动态链接到您的 DLL。此外，如果您的本机代码库使用 Microsoft 的任何 C 运行时库，请链接到 C 运行时库的多线程、静态版本。要指定此类型的链接，请使用 /MT 编译器选项。

创建本机扩展包

要将您的本机扩展提供给应用程序开发人员，需要将所有相关的文件打包到一个 ANE 文件中。ANE 文件是一个存档文件，它包含：

- 扩展的 ActionScript 库
- 扩展的本机代码库
- 扩展描述符文件
- 扩展的证书
- 扩展的资源，如图像。

使用 AIR 开发人员工具 (ADT) 创建 ANE 文件。 ADT 的完整文档位于 AIR 开发人员工具中。

打包扩展的 ADT 示例

下例说明如何使用 ADT 打包 ANE 文件。本示例打包 ANE 文件，以便与在以下设备上运行的应用程序一起使用：

- Android 设备。
- Android x86 设备。
- iOS 设备。
- iOS Simulator。
- 使用默认的仅 ActionScript 实现的其他设备。

```
adt -package <signing options> -target ane MyExtension.ane MyExt.xml -swc MyExtension.swc     -platform
Android-ARM -C platform/Android .
                        -platform Android-x86 -C platform/Android-x86 .
                        -platform iPhone-ARM -platformoptions platform.xml
                        abc/x.framework lib.o -C platform/ios .
                        -platform iPhone-x86 -C platform/iosSimulator
                        -platform default -C platform/default library.swf
```

在本示例中，使用以下命令行选项创建 ANE 包：

- <signing options>

  您可以选择对 ANE 文件签名。有关详细信息，请参阅第 36 页的 “ 为本机扩展创建签名证书 ”。

- -target ane

  -target 标志指定要创建哪种类型的包。使用 ane 目标打包本机扩展。

- MyExtension.ane

  指定要创建的包文件的名称。使用 .ane 文件扩展名。

- MyExt.xml

  指定扩展描述符文件。该文件指定扩展 ID 和支持的平台。AIR 使用此信息定位并加载应用程序的扩展。在本示例中，扩展描述符是：

```
<extension xmlns="http://ns.adobe.com/air/extension/3.1">
                                <id>com.sample.ext.MyExtension</id>
                                <versionNumber>0.0.1</versionNumber>
                                <platforms>
                                <platform name="Android-ARM">
                                <applicationDeployment>
                                <nativeLibrary>MyExtension.jar</nativeLibrary>
                                <initializer>com.sample.ext.MyExtension</initializer>
                                </applicationDeployment>
                                </platform>
                                <platform name="Android-x86">
                                <applicationDeployment>
                                <nativeLibrary>MyExtension.jar</nativeLibrary>
                                <initializer>com.sample.ext.MyExtension</initializer>
                                </applicationDeployment>
                                </platform>
                                <platform name="iPhone-ARM">
                                <applicationDeployment>
                                <nativeLibrary>MyExtension.a</nativeLibrary>
                                <initializer>InitMyExtension></initializer>
                                </applicationDeployment>
                                </platform>
                                <platform name="iPhone-x86">
                                <applicationDeployment>
                                <nativeLibrary>MyExtension.a</nativeLibrary>
                                <initializer>InitMyExtension></initializer>
                                </applicationDeployment>
                                </platform>
                                <platform name="default">
                                <applicationDeployment/>
                                </platform>
                                </platforms>
                                </extension>
```

有关详细信息，请参阅第 63 页的 “ 本机扩展描述符文件 ”。

- MyExtension.swc

  指定包含扩展的 ActionScript 端的 SWC 文件。

- -platform Android-ARM -C platform/Android . -platform Android-x86 -C platform/Android-x86 .-platform iPhone-ARM -platformoptions platform.xml-Cplatform/iOS.

  -platform 标志指定此 ANE 文件支持的平台。该名称后面的选项指定在何处查找特定于平台的库和资源。在本示例中，Android-ARM 平台的 -C 选项指示将相对目录 platform/Android 作为工作目录。后面的目录和文件都是相对于新工作目录。

  因此，在本示例中，相对目录 platform/Android 包含所有 Android 本机代码库和资源。它还包含特定于 Android 平台的 library.swf 文件和任何其他特定于 Android 平台的 SWF 文件。

  iPhone-ARM 平台的 -platformoptions 标志是一个可选项，用于指定特定于平台的选项。这些选项包含用于链接到 iOS 框架 （而不是默认项）以及将第三方静态库绑定到本机扩展的选项。请参阅 第 38 页的 “iOS 本机库 ”。

- -platform default -C platform/default library.swf

  当 -platform 选项指定 default 平台时，请不要指定任何本机代码文件。仅指定一个 library.swf 文件，和其他 SWF 文件（如果有）。

ANE 包中的 SWC 文件和 SWF 文件

您在 ADT 打包命令的 -swc 选项中指定一个 SWC 文件。此 SWC 文件是您的 ActionScript 库。它包含一个名为 library.swf 的文件。ADT 将 library.swf 从 SWC 文件放入 ANE 文件。当 AIR 应用程序使用本机扩展时，它在其库路径中包括该扩展的 ANE 文件，以便该应用程序可以编译。事实上，该应用程序根据 library.swf 中的公共接口进行编译。

有时，您需要为每个目标平台分别创建一个不同的 ActionScript 实现。在此情况下，应在使用 ADT 打包命令之前，为每个平台编译一个 SWC 文件并将每个 SWC 文件中的 library.swf 文件放在适当的平台目录中。您可以使用提取工具（如 WinZip）从 SWC 文件提取 library.swf。

考虑扩展的公共接口在所有平台上都相同的情况。这种情况下，在 ADT 命令的 -swc 选项中使用哪个平台特定 SWC 文件无关紧要。无关紧要的原因是 SWC 文件中的 SWF 文件只用于应用程序编译。您放入平台目录中的 SWF 文件是本机扩展在运行时使用的文件。

请注意有关 library.swf 文件的以下情况：

- ADT 需要您为每个平台提供一个名为 library.swf 的主 SWF 文件。在创建 SWC 文件时， library.swf 就是 SWF 文件的名称。
- 每个平台的 library.swf 文件可以不相同。
- 如果 ActionScript 端没有平台依赖项，则每个平台的 library.swf 文件可以相同。
- 每个平台的 library.swf 可以加载您在特定于平台的目录中包括的其他 SWF 文件。这些其他 SWF 文件可以是任何名称。

应用程序打包的 ANE 文件规则

利用 ADT 打包使用本机扩展的应用程序。在打包使用扩展的应用程序时，ADT 会验证 ANE 文件中指定的平台与应用程序打包目标是否匹配。例如，平台 Android-ARM 匹配 Android apk 包。

此外， ADT 将 default 平台与任何 目标包匹配。default 平台指定扩展的仅 ActionScript 版本。假定有一个使用应用程序绑定扩展的 AIR 应用程序。如果扩展的任何指定平台都不与设备对应， AIR 仅加载 default 平台扩展的 ActionScript 库。

例如，假定有一个指定 iPhone-ARM、 Android-ARM 和 default 平台的应用程序绑定扩展。当使用该扩展的应用程序运行于 Windows 平台时，它将使用该扩展的 default 平台库。

因此，当您创建用于应用程序绑定的 ANE 文件时，请在打包使用 ANE 文件的应用程序时，考虑 ADT 使用的以下规则：

- 要创建 Android 应用程序包，ANE 文件必须包括 Android-ARM 平台。或者，ANE 文件必须包括默认平台和至少一个其他平台。
- 要创建 iOS 应用程序包， ANE 文件必须包括 iPhone-ARM 平台。或者， ANE 文件必须包括默认平台和至少一个其他平台。
- 要创建 iOS Simulator 应用程序包，ANE 文件必须包括 iPhone-x86 平台。或者，ANE 文件必须包括默认平台和至少一个其他平台。
- 要创建 Mac OS X 应用程序包， ANE 文件必须包括 MacOS-x86-64 平台。或者， ANE 文件必须包括默认平台和至少一个其他平台。
- 要创建 Windows 应用程序包，ANE 文件必须包括 Windows-x86 平台。或者，ANE 文件必须包括默认平台和至少一个其他平台。

一个应用程序只能绑定一个平台实现。但是，如果使用 ADL 实用程序测试包含扩展的应用程序，则会在运行时选择实现。根据测试平台和 ANE 包的不同，这种在运行时进行的选择可能导致行为差异。例如，如果 ANE 包含 Android-ARM、Windows-x86 和 default 平台的实现，测试时使用的实现将有所不同，具体取决于测试计算机运行的是 Windows 还是 OS X。在 Windows 上，将使用 Windows-x86 平台实现 （即使是使用移动配置文件进行测试）；在 OS X 上，将使用 default 实现。

在 ANE 包中包括其他 Android 共享 .so 库

假定有一个面向平台 Android-ARM 的本机扩展。扩展本机端的主要库为以下任一项：

- .so 库 （如果使用 Android NDK）
- JAR 文件 （如果使用 Android SDK）

不过，本机端有时需要比扩展的主要 .so 库或 JAR 文件更多的本机库 （.so 库）。

例如：

- 您正在使用 Java API 开发本机扩展。但是，您需要使用 JNI （Java 本机接口）从 Java 代码访问本机 .so 库。
- 您正在使用 C API 开发本机扩展。但是，您需要将代码划分到多个共享库中。根据扩展的逻辑，您需要在运行时加载适当的共享 .so 库。

在创建 ANE 包时，请使用以下目录结构：

```
<Android platform directory>/
                         libs/
                         armeabi/
                         <Android emulator native libraries>
                         armeabi-v7a/
                         <Android device native libraries>
```

在使用 ADT 创建 ANE 包时，请设置 -platform 选项以指定 Android 平台目录内容：

```
-platform Android-ARM -C <Android platform directory> .
```

当应用程序开发人员使用 ADT 将 ANE 包包括在 APK 包中时， APK 包包括以下库：

- libs/armeabi-v7a 中的库 （如果 ADT 目标为 apk 或 apk-captive-runtime）。
- libs/armeabi 中的库 （如果 ADT 目标为 apk-emulator、 apk-debug 或 apk-profile）。

注：对于 iPhone-ARM 平台，不能在 ANE 文件中包括共享库。有关详细信息，请参阅第 38 页的 “ 构建本机库 ”。

将资源包括在本机扩展包中

您的扩展的 ActionScript 端和本机代码端有时使用外部资源，如图像。

在 ActionScript 端，独立于平台的 SWC 文件可以包括它需要的资源。如果依赖于平台的 SWF 文件需要资源，请将资源包括在您在 ADT 命令中指定的依赖于平台的目录结构中。

在本机代码端，还可以将资源包括在依赖于平台的目录结构中。将资源放在相对于本机代码库的子目录中本机代码期望的位置。 ADT 在打包您的 ANE 时将保留此目录结构。

使用 Android 和 iOS 设备上的资源有一些额外要求。

Android 设备上的资源

对于 Android-ARM 平台，将资源放在与包含本机代码库的目录对应的名为 res 的子目录中。使用 developer.android.com 上的提供资源中所述的资源填充目录。当 ADT 打包 ANE 文件时，它将资源放在生成的 ANE 包的 Android-ARM/res 目录中。

当访问扩展的 Java 本机代码库中的资源时，请使用 FREContext 类中的 getResourceID() 方法。不要使用标准 Android 资源 ID 机制访问资源。有关 getResourceID() 方法的详细信息，请参阅第 114 页的 “ 方法详细信息 ”。

getResourceId() 方法提取作为资源名的 String 参数。为避免资源名在一个应用程序的扩展之间发生冲突，请在扩展中为每个资源文件提供一个唯一名称。例如，通过在资源名前加扩展 ID 创建资源名，如 com.sample.ext.MyExtension.myImage1.png。

使用 R.* 机制访问本机资源

在以前的版本中，只能使用 getResourceID() API 访问 Android 本机扩展中的本机 Android 资源，而不能对 ANE 使用 R.* 机制。从 AIR 4.0 开始，将可以使用 R.* 机制访问资源。使用 R.* 机制时，应确保使用平台描述符 platform.xml，其中已定义所有依赖项：

```
<?xml version="1.0"?>
                            <xs:schema
                            xmlns:xs="http://www.w3.org/2001/XMLSchema"
                            targetNamespace="http://ns.adobe.com/air/extension/4.0"
                            xmlns="http://ns.adobe.com/air/extension/4.0"
                            elementFormDefault="qualified">
                            <xs:element name="platform">
                            <xs:complexType>
                            <xs:all>
                            <xs:element name="description"   type="LocalizableType" minOccurs="0"/>
                            <xs:element name="copyright"     type="xs:string"       minOccurs="0"/>
                            <xs:element name="packagedDependencies" minOccurs="0">
                            <xs:complexType>
                            <xs:all>
                            <xs:element name="packagedDependency" type="name" minOccurs="0"
maxOccurs="unbounded"/>
                            </xs:all>
                            </xs:complexType>
                            </xs:element>
                            <xs:element name="packagedResources" minOccurs="0">
                            <xs:complexType>
                            <xs:all>
                            <xs:element name="packagedResource" minOccurs="0" maxOccurs="unbounded"/>
                            <xs:complexType>
                            <xs:all>
                            <xs:element name="packageName" type="name" minOccurs="0"/>
                            <xs:element name="folderName" type="name" minOccurs="0"/>
                            </xs:all>
                            </xs:complexType>
                            </xs:all>
                            </xs:element>
                            </xs:complexType>
                            </xs:element>
                            </xs:all>
                            </xs:complexType>
                            </xs:element>
                            <xs:simpleType name="name">
                            <xs:restriction base="xs:string">
                            <xs:pattern value="[A-Za-z0-9\-\.]{1,255}"/>
                            </xs:restriction>
                            </xs:simpleType>
                            <xs:complexType name="LocalizableType" mixed="true">
                            <xs:sequence>
                            <xs:element name="text" minOccurs="0" maxOccurs="unbounded">
                            <xs:complexType>
                            <xs:simpleContent>
                            <xs:extension base="xs:string">
                            <xs:attribute name="lang" type="xs:language" use="required"/>
                            </xs:extension>
                            </xs:simpleContent>
                            </xs:complexType>
                            </xs:element>
                            </xs:sequence>
                            </xs:complexType>
                            </xs:schema>
```

以下是 platform.xml 中依赖项的一个示例：

```
<packagedDependencies>
                        <packagedDependency>android-support-v4.jar</packagedDependency>
                        <packagedDependency>google-play-services.jar</packagedDependency>
                        </packagedDependencies>
                        <packagedResources>
                        <packagedResource>
                        <packageName>com.myane.sampleasextension</packageName>
                        <folderName>ane-res</folderName>
                        </packagedResource>
                        <packagedResource>
                        <packageName>com.google.android.gms</packageName>
                        <folderName>google-play-services-res</folderName>
                        </packagedResource>
                        </packagedResources>
```

其中：

- packagedDependencies 用于提供 ANE 所依赖的所有 JAR 文件的名称。
- packagedResources 定义 ANE 或任何其他 JAR 文件使用的资源。
- folderName 定义资源文件夹的名称。
- packageName 定义使用这些资源的 JAR 文件的包名称。
- packagedDependencies 和 packagedResources 将从扩展命名空间 4.0 开始提供。

Android-ARM 文件夹包含所有 ANE JAR 文件和资源以及第三方 JAR 文件。以下是一个 ANE 打包命令示例：

```
bin/adt -package -target ane sample.ane extension.xml -swc sampleane.swc -platform Android-ARM -
platformoptions platform.xml -C Android-ARM .
```

使用 R.* 资源访问机制时，您不需要将第三方 JAR 文件和资源与 ANE JAR 文件和资源进行合并。 ADT 会在内部合并这些 JAR 文件和资源。所有依赖项和资源仍需要打包在 ANE 中。

注意：

- ANE 项目应是一个库项目， R.* 资源访问机制才会有效。
- 对资源文件夹的名称没有限制，可以 “res” 或任何其他有效字符串开头。
- 如果打包 ANE 时未使用 -platformoptions 开关，则只能通过 getResourceId() 机制进行资源访问。

iOS 设备上的资源

资源位置
在使用 ADT 为 iOS 设备创建 ANE 文件之前，请将非本地化资源放在包含本机代码库的目录。将本地化资源放在下一节中所述的子目录中。

不过， iOS 应用程序绑定在应用程序绑定的顶级目录包含其资源。这些资源包括每个扩展的特定于平台的部分使用的所有资源。当 AIR 应用程序开发人员将 ANE 文件与应用程序打包在一起时， ADT 将执行以下操作：

1 检查 ANE 包的 iPhone-ARM 目录。

2 将该目录中的所有文件视为资源文件，但 library.swf 和扩展描述符文件除外。

3 将资源文件移动到应用程序的顶级目录。

由于多个扩展的资源文件都移动到同一位置，因此可能存在资源文件名冲突。如果存在冲突， ADT 不会打包应用程序并报告错误。因此，请为扩展中的每个资源文件提供一个唯一名称。例如，通过在资源名前加扩展 ID 创建资源名，如 com.sample.ext.MyExtension.myImage1.png。

注：资源位于应用程序的顶级目录 -- 不在扩展目录中。因此，要访问资源，请使用 ActionScript 属性 File.applicationDirectory。不要使用 ActionScript API ExtensionContext.getExtensionDirectory() 导航到扩展目录来查找资源。资源不在这里。

本地化资源

您的扩展的本机代码库 （但不是 ActionScript 端）可以使用本地化资源。要使用本地化资源，请执行以下操作：

- 在您使用 ADT 创建 ANE 文件时指定的特定于平台的目录中，将本地化资源放在特定于语言的子目录中。按以下方式命名这些子目录：

  `language[_region].lproj`

  根据 iOS 约定设置 ***language*** 和 ***region*** 的值。请参阅 iOS 开发人员库中的语言和区域设置目标。

- 使用 developer.apple.com/library/ios/navigation 中所述的格式将本地化字符串放入 .strings 文件。不过，不要将任何文件命名为 Localizable.strings，因为该名称是应用程序绑定中使用的默认名称。

以下目录结构是一个目录示例，该目录将包含的所有特定于 iOS 平台的文件都打包在 ANE 文件中：

```
platform/
                              iOS/
                              library.swf
                              myNativeLibrary.a
                              myNonLocalizedImage1.jpg
                              de.lproj/
                              MyGermanLocalizable.strings
                              MyGermanLocalizedImage1.png
                              en_us.lproj/
                              MyAmericanLocalizable.strings
                              MyAmericanLocalizedImage1.png
                              en_gb.lproj/
                              MyBritishLocalizable.strings
                              MyBritishLocalizedImage1.png
```

第 6 章 : 构建和安装 AIR for TV 本机扩展

要为 AIR for TV 设备开发本机扩展，您必须有权访问 AIR for TV 扩展开发工具包 (EDK)。 EDK 分发给在其产品中包括 AIR for TV 的设备制造商和单片机制造商。

有关使用 “AIR for TV” 的制造商的详细信息，请参阅 Adobe AIR for TV 快速入门 (PDF)

开发 AIR for TV 扩展的任务概述

当您为 AIR for TV 设备开发本机扩展时，请完成下列迭代式任务：

1 编写本机实现。

有关详细信息，请参阅第 13 页的 “ 使用 C 语言编写本机端代码 ”。

2 编写实际的 ActionScript 实现。

有关详细信息，请参阅第 6 页的 “ 编写 ActionScript 端代码 ”。请确保考虑第 11 页的 “ 本机扩展向后兼容性 ”。

3 编写存根 ActionScript 实现，以及根据需要编写模拟器 ActionScript 实现。

有关详细信息，请参阅第 54 页的 “ 设备绑定扩展和存根扩展 ” 以及第 55 页的 “ 检查扩展支持 ”。

4 创建证书以便对本机扩展进行签名。此步骤为可选步骤。

有关详细信息，请参阅第 36 页的 “ 为本机扩展创建签名证书 ”。

5 使用 AIR for TV 生成实用程序生成设备绑定扩展和存根扩展。该过程创建一个 ZIP 文件，以便在设备上安装设备绑定扩展。该过程还创建存根扩展的 ANE 文件，以便用来生成和测试 AIR 应用程序。

有关详细信息，请参阅第 56 页的 “ 构建 AIR for TV 本机扩展 ”。

6 如果有模拟器 ActionScript 实现可供使用，可使用 AIR for TV 生成实用程序生成模拟器扩展。该过程创建一个 ANE 文件，以便用来构建和测试 AIR 应用程序。

有关详细信息，请参阅第 56 页的 “ 构建 AIR for TV 本机扩展 ”。

7 向 ZIP 文件和 ANE 文件添加所需的任何资源，例如图像。

有关详细信息，请参阅第 61 页的 “ 将资源添加到您的 AIR for TV 本机扩展 ”。

8 在台式计算机上测试模拟器扩展。

有关详细信息，请参阅调试 AIR for TV 应用程序。

9 在设备上安装设备绑定扩展。

有关详细信息，请参阅第 62 页的 “ 在 AIR for TV 设备上安装设备绑定扩展 ”。

10 在设备上测试设备绑定扩展。

有关详细信息，请参阅第 62 页的 “ 在 AIR for TV 设备上运行 AIR 应用程序 ” 和调试 AIR for TV 应用程序。

11 向应用程序开发人员交付存根和 / 或模拟器 ANE 文件。

AIR for TV 扩展示例

Adobe® AIR® for TV 提供了多个本机扩展示例。本机实现是使用 C++ 编写的，并且使用了 AIR for TV 扩展开发工具包 (EDK)。EDK 分发给在其产品中包括 AIR for TV 的设备制造商和单片机制造商。有关 AIR for TV EDK 和如何创建 AIR for TV 扩展的详细信息，请参阅第 56 页的 “ 构建 AIR for TV 本机扩展 ”。

作为 AIR for TV 扩展开发人员，您可以：

- 复制这些示例，将其作为您的扩展的起点。
- 请参见这些示例中的代码样例，了解如何使用各种 C API 扩展函数以及 ActionScript ExtensionContext 类。
- 复制其中一个示例的生成文件，作为创建您的扩展的生成文件的起点。

HelloWorld 示例

HelloWorld 示例位于 AIR for TV 分发包的下列目录：

```
<AIR for TV installation directory>/products/stagecraft/source/ae/edk/helloworld
```

HelloWorld 示例是一个简单的扩展，其目的是阐明基本扩展行为。它完成下列工作：

- 使用 ExtensionContext.call() 方法将一个字符串从 ActionScript 端传递给本机实现。
- 将一个字符串从本机实现返回给 ActionScript 端。
- 在本机实现中启动一个线程，以便向 ActionScript 端发送异步事件。

下表介绍了每个文件及其相对于 helloworld/ 目录的位置：

文件	说明
HelloWorld.as 所在目录： as/device/src/tv/adobe/extension/example/	定义 HelloWorld 类的实际（而非存根）扩展的 ActionScript 端。它完成下列工作： • 创建 ExtensionContext 实例。 • 定义该扩展的 ActionScript API：Hello() 和 StartCount()。 • 侦听 ExtensionContext 实例上的事件，并且将这些事件重新调度给 HelloWorld 实例的侦听器。
HelloWorld.as 所在目录： as/distributable/src/tv/adobe/extension/example/	定义 HelloWorld 类的存根扩展的 ActionScript。这一仅包含 ActionScript 的存根定义了该扩展的 ActionScript API：Hello() 和 StartCount()。不过，存根实现不调用本机函数。
HelloWorldExtensionClient.as 所在目录： client/src	使用该扩展的 AIR 应用程序。AIR 应用程序是扩展的客户端。它完成下列工作： • 创建 HelloWorld 类的实例。 • 侦听 HelloWorld 实例上的事件。 • 调用 HelloWorld 实例的 Hello() 和 StartCount() API。
HelloWorldExtensionClient-app.xml 所在目录： client/src	AIR 应用程序描述符文件。包括 extensionID 值为 tv.adobe.extension.example.HelloWorld 的 <extensions> 元素。

文件	说明
HelloWorld.h 所在目录： native/	HelloWorld 类的 C++ 头文件。
HelloWorld.cpp 所在目录： native/	HelloWorld 类的 C++ 实现文件。该实现完成下列工作： • 定义 FREFunction Hello()，该函数将它的字符串参数写入控制台。它还返回字符串 “Hello from extensionland”。 • 定义 FREFunction StartCount()，该函数启动一个异步线程，以便每 500 毫秒向 ExtensionContext 实例发送一个事件。
HelloWorldExtension.cpp 所在目录： native/	包含下列 C API 扩展函数的实现： • FREInitializer() • FREContextInitializer() • FREContextFinalizer() • FREFinalizer()
PlatformEDKExtension_HelloWorld.mk	HelloWorld 扩展的生成文件。
ExtensionUtil.h 所在目录： <AIR for TV 安装目录>/products/stagecraft/source/ae/edk	包含便于编写 C 或 C++ 实现代码的宏。
ExtensionBridge.cpp 所在目录： <AIR for TV 安装目录>/products/stagecraft/source/ae/edk	AIR for TV 扩展模块实现。当您生成您的本机实现时，必须在您的内部版本中包括此源文件。
phonyEdkAneCert.p12 所在目录： <AIR for TV 安装目录>/products/stagecraft/source/ae/edk	一个虚假证书，生成实用程序使用该证书将 HelloWorld 扩展打包为 ANE 文件。
extension.mk 所在目录： <AIR for TV 安装目录>/products/stagecraft/source/ae/edk	扩展模块的生成文件。请不要修改此文件。

Process 示例

Process 示例位于 AIR for TV 分发包的以下目录中：

```
<AIR for TV installation directory>/products/stagecraft/source/ae/edk/process
```

Process 扩展允许 AIR 应用程序操纵 Linux 进程，包括下列功能：

- 衍生 Linux 进程。该进程执行 AIR 应用程序指定的 Linux 命令。
- 获取该进程的进程标识符。

- 检查该进程是否已完成。
- 接收指示该进程已完成的事件。
- 从已完成的进程获取返回码。
- 接收指示该进程已经写入 stdout 或 stderr 的事件。
- 从 stdout 和 stderr 检索输出字符串。
- 将字符串写入 stdin。
- 向进程发送中断信号。
- 终止进程。

下表介绍了每个文件及其相对于 process/ 目录的位置：

文件	说明
Process.as 所在目录： as/device/src/tv/adobe/extension/process/example/	定义 Process 类的实际（而非存根）扩展的 ActionScript 端。它完成下列工作： • 创建 ExtensionContext 实例。 • 定义扩展的 ActionScript API。 • 侦听 ExtensionContext 实例上的事件，并且将这些事件重新调度给 Process 实例的侦听器。
ProcessEvent.as 所在目录： as/device/src/tv/adobe/extension/process/example/	定义 ProcessEvent 类，该类派生自 Event 类。 AIR 应用程序 ActionScript 侦听这些 ProcessEvent 通知。
Process.as 所在目录： as/distributable/src/tv/adobe/extension/process/example/	定义 Process 类的存根扩展的 ActionScript。这一仅包含 ActionScript 的存根定义了该扩展的 ActionScript API。不过，存根实现不调用本机函数。
ProcessExtensionClient.as 所在目录： client/simple/src	使用该扩展的 AIR 应用程序。AIR 应用程序是扩展的客户端。它提供了一个说明 AIR 应用程序如何使用 Process 扩展 API 的示例。
ProcessExtensionClient-app.xml 所在目录： client/simple/src	AIR 应用程序描述符文件。包括 extensionID 值为 tv.adobe.extension.process.Process 的 <extensions> 元素。
Process.h 所在目录： native/	抽象 Process 类的 C++ 头文件。
ProcessLinux.h 所在目录： native/	具体 ProcessLinux 类的 C++ 头文件。 ProcessLinux 类派生自 Process 类。它声明 Process 类的 Linux 实现的私有方法和数据。

文件	说明
ProcessLinux.cpp 所在目录： native/	ProcessLinux 类的 C++ 实现文件。该实现包括下列功能： • 定义 FREFunction 函数。这些函数使用 Linux 系统调用执行某些操作 （例如，创立和执行 Linux 进程）以及与 stdin、 stdout 和 stderr 交互。 • 监控衍生进程的状态。该实现创建一个线程以达到此目的。该线程使用 C 扩展 API FREDispatchStatusEventAsync() 来报告事件。 • 定义帮助器函数，以便创建用于从 FREFunction 函数向 ActionScript 端返回信息的 FREObject 变量。这些帮助器函数使用 C API 扩展函数，如 FRENewObjectFromBool()、FRENewObjectFromUTF8() 和 FRENewObjectFromUint32()。
ProcessExtension.cpp 所在目录： native/	包含下列 C API 扩展函数的实现： • FREInitializer() • FREContextInitializer() • FREContextFinalizer() • FREFinalizer()
PlatformEDKExtension_Process.mk	Process 扩展的生成文件。
ExtensionUtil.h 所在目录： <AIR for TV 安装目录>/products/stagecraft/source/ae/edk	包含便于编写 C 或 C++ 实现代码的宏。
ExtensionBridge.cpp 所在目录： <AIR for TV 安装目录>/products/stagecraft/source/ae/edk	AIR for TV 扩展模块实现。当您生成您的本机实现时，必须在您的内部版本中包括此源文件。
phonyEdkAneCert 所在目录： <AIR for TV 安装目录>/products/stagecraft/source/ae/edk	一个虚假证书，生成实用程序使用该证书将 Process 扩展打包为 ANE 文件。
extension.mk 所在目录： <AIR for TV 安装目录>/products/stagecraft/source/ae/edk	扩展模块的生成文件。请不要修改此文件。

在 x86Desktop 平台上构建扩展示例

若要在 x86Desktop 平台上构建 Hello World 或 Process 扩展示例，请先为 x86Desktop 平台生成 AIR for TV 分发包。有关说明，请参阅 Adobe AIR for TV 快速入门 (PDF) 中的 Linux 快速入门。

注：确保已安装 AIR 3 SDK、开源 Flex® SDK 和 Java™ 运行时，并在 PATH 环境变量中包含 bin 目录。为 AIR for TV 构建本机扩展依赖这些库。

接下来，将构建 Hello World 或 Process 扩展示例添加到 x86Desktop 构建中。请执行以下操作：

1 更改到目录：

```
<AIR for TV installation directory>/products/stagecraft/build/linux/platforms/x86Desktop
```

2 链接到要构建的扩展的 .mk 文件。例如：

```
ln -s <AIR for TV installation
directory>/products/stagecraft/source/ae/edk/helloworld/PlatformEDKExtension_HelloWorld.mk
```

3 更改到目录：

```
 <AIR for TV installation directory>/products/stagecraft/build/linux
```

4 生成分发包，其中包含扩展示例：

```
make
```

也可以只构建扩展示例。例如：

```
make PlatformEDKExtension_HelloWorld.mk
```

生成实用程序为每个扩展示例创建两个文件。它将这些文件放入下列目录之一，具体取决于您为 SC_BUILD_MODE 指定了 debug 还是 release：

```
<AIR for TV installation directory>/build/stagecraft/linux/x86Desktop/debug/bin
<AIR for TV installation directory>/build/stagecraft/linux/x86Desktop/release/bin
```

生成实用程序为扩展示例创建的文件包括：

- 一个 ZIP 文件，包含设备捆绑扩展。
- 一个 ANE 文件，包含存根或模拟器扩展。

设备绑定扩展和存根扩展

当您为 Adobe® AIR® for TV 设备编写本机扩展时，您需要创建该扩展的两个变体：

- 设备绑定扩展，也称为实际扩展。
- 存根扩展。

此外，您还可以根据情况提供第三个变体：模拟器扩展。

设备绑定扩展

设备绑定扩展是设备上安装的变体。 ActionScript 端调用本机实现的函数。您生成此实际 ActionScript 实现以及本机实现，并创建一个 ZIP 文件。设备制造商将此文件解压缩到设备上的特定目录中。

存根扩展

存根本机扩展具有与实际 ActionScript 实现相同的 ActionScript 接口，但是 ActionScript 方法不做任何事情。存根扩展是仅包含 ActionScript 的扩展；它没有本机实现。当您生成存根 ActionScript 实现时，您将创建一个 ANE 文件。

AIR 应用程序开发人员使用此 ANE 文件达到三个目的：

- 编译使用本机扩展的 AIR 应用程序。
- 在台式计算机而不是目标设备上运行 AIR 应用程序。
- 包括在 AIR 应用程序软件包中。

模拟器扩展

可选的第三种变体是模拟器扩展。此实现也具有与实际 ActionScript 实现相同的 ActionScript 接口。不过，它的 ActionScript 方法在 ActionScript 中模拟扩展的行为。像存根扩展一样，模拟器扩展是仅包含 ActionScript 的扩展；它没有本机实现。当您生成模拟器 ActionScript 实现时，您将创建一个 ANE 文件。

AIR 应用程序开发人员可以使用模拟器扩展 ANE 文件编译他们的应用程序。与使用存根扩展进行测试相比，他们可以使用此 ANE 文件在台式计算机上更加彻底地测试应用程序。他们还可以将模拟器扩展包括在 AIR 应用程序软件包中。

注：您可以创建一个模拟器扩展，作为存根扩展的替代品或辅助品。

设备绑定扩展、存根扩展和模拟器扩展的使用

AIR 应用程序开发人员使用存根和模拟器扩展完成下列工作：

- 使用存根扩展或模拟器扩展编译 AIR 应用程序。
- 使用存根扩展或模拟器扩展在台式计算机上测试应用程序。
- 将存根扩展或模拟器扩展打包到它们的可分发 AIR 应用程序中。

  注： 如果您为 AIR 应用程序开发人员同时提供了存根和模拟器扩展，请告诉他们将哪个扩展与他们的可分发应用程序一起打包。

当 AIR 应用程序在设备上运行时， AIR for TV 将执行下列操作：

1. 在设备上寻找相应的设备绑定 （实际）扩展。
2. 如果该扩展存在，则 AIR for TV 会加载它以供 AIR 应用程序使用。
3. 如果该扩展不存在， AIR for TV 将改为加载与应用程序一起打包的存根或模拟器扩展。

检查扩展支持

一种最佳做法是在本机扩展和 AIR for TV 应用程序之间提供握手。握手告诉应用程序有关该设备上扩展的可用性信息。利用此信息， AIR for TV 应用程序能够决定采取哪一条逻辑路径。

决定在您的 ActionScript 扩展类中为此握手提供哪些公共接口。然后，指示 AIR for TV 应用程序开发人员如何使用这些接口。

例如，假定有一个本机扩展，该扩展具有：

- 一个存根扩展，与 AIR for TV 应用程序一起打包。
- 一个用于在台式计算机上测试应用程序的模拟器扩展。
- 用于在目标设备上安装的设备绑定实际扩展。

在每个变体的 ActionScript 端中，使用下列实现定义一个名为 isSupported() 的方法：

- 在与本机实现交互的实际 ActionScript 实现中，实现该方法以返回 true。

  执行调用的 AIR 应用程序正在具有设备绑定扩展的设备上运行。因此，应用程序知道它可以继续使用该扩展。

- 在存根 ActionScript 实现中，实现该方法以返回 false。

  执行调用的 AIR 应用程序正在具有非设备绑定扩展的设备上运行。在此情况下， AIR for TV 使用与 AIR 应用程序一起打包的存根扩展。 false 返回值通知应用程序不要调用该扩展的任何其他方法。应用程序可以做出相应的逻辑决策，如正常退出。

- 在模拟器 ActionScript 实现中，实现该方法以返回 true。

执行调用的 AIR 应用程序正在台式计算机上运行以达到测试目的。因此， true 返回值通知应用程序它可以继续使用该扩展。

不过，根据您的扩展的不同，您可以指示应用程序开发人员将其应用程序与您的模拟器扩展一起打包。那么，如果应用程序正在不具有设备绑定扩展的设备上运行，则应用程序将使用模拟器版本的扩展继续运行。

构建 AIR for TV 本机扩展

当您构建 AIR for TV 本机扩展时，您将构建两个版本的扩展：

- 设备绑定扩展。
- 存根或模拟器扩展。

设备绑定扩展包括：

- 通常使用 C 或 C++ 编写的本机实现。
- 调用该本机实现的函数的实际 ActionScript 实现。

存根或模拟器扩展是仅包含 ActionScript 的实现。

有关实际、存根和模拟器 ActionScript 实现的详细信息，请参阅第 54 页的 “ 设备绑定扩展和存根扩展 ”。

为扩展创建签名证书

您可以选择对本机扩展进行数字签名。对扩展签名是可选的。

默认情况下， AIR for TV 生成实用程序使用虚假证书。该虚假证书仅适用于测试。有关创建证书颁发机构证书的信息，请参阅第 36 页的 “ 为本机扩展创建签名证书 ”。

编写本机实现

对于 AIR for TV 而言，您的扩展的本机实现是 AIR for TV 模块。有关 AIR for TV 模块的一般信息 （包括有关生成模块的详细信息），请参阅优化和集成 Adobe AIR for TV (PDF)。

AIR for TV 分发包提供了扩展开发工具包 (EDK)，用于编写和生成您的扩展的本机实现。

EDK 包括下列部分：

- C 扩展 API 头文件：

  ```
  <AIR for TV installation directory>/products/stagecraft/include/ae/edk/FlashRuntimeExtensions.h
  ```

  该头文件声明本机实现使用的 C 类型和函数。

- 一个位于以下源文件中的扩展模块实现：

  ```
  <AIR for TV installation directory>/products/stagecraft/source/ae/edk/ExtensionBridge.cpp
  ```

  请不要修改此扩展模块实现。当您生成您的本机实现时，必须在您的内部版本中包括此源文件。

- 生成您的设备绑定扩展所需的生成文件支持。有关详细信息，请参阅第 57 页的“创建 .mk 文件”和第 59 页的“运行生成实用程序 ”。

注: AIR for TV EDK 要求将 FREInitializer() 方法命名为 Initializer()，将 FREFinalizer() 方法命名为 Finalizer()。有关这些方法的详细信息，请参阅第 13 页的 “ 扩展初始化 ”和第 17 页的 “ 扩展终止化 ”。

将 ActionScript 和本机代码放入目录结构中

设备绑定扩展是特定于硬件平台的。当您开发设备绑定扩展时，请将您的文件放入您的平台的子目录中。此子目录位于以下目录中：

```
<AIR for TV installation directory>/products/stagecraft/thirdparty-private/<yourCompany>/stagecraft-platforms/<yourPlatform>/edk
```

例如，公司 A 使用以下子目录来进行面向其平台 B 的开发工作：

```
<AIR for TV installation directory>/products/stagecraft/thirdparty-private/CompanyA/stagecraft-platforms/PlatformB/edk
```

请将您的 C 实现的头文件和源文件放入 < 您的平台 >/edk 目录或其子目录中。例如，将您的扩展 .cpp 和 .h 文件在放入以下目录中：

```
<AIR for TV installation directory>/products/stagecraft/thirdparty-private/CompanyA/stagecraft-platforms/PlatformB/edk/myExtension/native
```

类似地，将您的实际 ActionScript 实现的 .as 文件放入 < 您的平台 >/edk 目录或其子目录中。例如：

```
<AIR for TV installation directory>/products/stagecraft/thirdparty-private/CompanyA/stagecraft-platforms/PlatformB/edk/myExtension/as/real
```

另外，请将您的存根或模拟器 ActionScript 实现的 .as 文件放入 < 您的平台 >/edk 目录或其子目录中。例如：

```
<AIR for TV installation directory>/products/stagecraft/thirdparty-private/CompanyA/stagecraft-platforms/PlatformB/edk/myExtension/as/stub
<AIR for TV installation directory>/products/stagecraft/thirdparty-private/CompanyA/stagecraft-platforms/PlatformB/edk/myExtension/as/simulator
```

注：AIR for TV 提供的扩展示例位于目录 <***AIR for TV*** 安装目录 >/products/stagecraft/source/edk 中。请不要将您的扩展文件放入此目录中。

创建 .mk 文件

与其他 AIR for TV 模块一样，要生成您的扩展模块，您首先需要创建 .mk 文件。 .mk 文件的主要用途是指定要生成的源文件。

要创建 .mk 文件，请执行下列操作：

1 复制以下目录中的 PlatformEDKExtension_HelloWorld.mk 文件或 PlatformEDKExtension_Process.mk 文件：

```
<AIR for TV installation directory>/products/stagecraft/source/ae/edk/helloworld/
```

或

```
<AIR for TV installation directory>/products/stagecraft/source/ae/edk/process/
```

将其复制到：

```
<AIR for TV installation directory>/products/stagecraft/thirdparty-private/<yourCompany>/stagecraft-platforms/<yourPlatform>
```

此目录与包含您的平台的 Makefile.config 文件的目录相同。请参阅优化和集成 Adobe AIR for TV (PDF) 中 “ 编码、生成和测试 ” 一章中的 “ 创建平台 Makefile.config”。

2 将 .mk 文件重命名为 PlatformEDKExtension_< 您的扩展名称 >.mk。AIR for TV 生成实用程序自动通过此命名约定发现 .mk 文件。

请始终使用 PlatformEDKExtension_ 作为 .mk 文件名称的开头。

3 编辑 .mk 文件中标有 “REQUIRED” 的部分。

进行下列必要的修改：

- 将 SC_EDK_EXTENSION_NAME 设置为扩展名称。将该变量设置为 PlatformEDKExtension_< 您的扩展名称 >.mk 中 < 您的扩展名称 > 的值。
- 将 SC_EDK_EXTENSION_PACKAGE 设置为扩展软件包名称。将该值设置为在您的扩展的 ActionScript 端使用的软件包名称。

  生成实用程序将该值用作扩展的扩展描述符文件中 <id> 元素的值。它还使用此值和 .ane 扩展文件名命名生成的 ANE 文件。

  有关扩展描述符文件的详细信息，请参阅第 63 页的 “ 本机扩展描述符文件 ”。
- 将 SC_EDK_EXTENSION_VERSION 设置为扩展的版本号。

  生成实用程序将该值用作扩展的扩展描述符文件中 <versionNumber> 元素的值。
- 设置 SC_MODULE_SOURCE_DIR、SC_MODULE_SOURCE_FILES 和 SC_ADDITIONAL_MODULE_OBJ_SUBDIRS 以指定 AIR for TV 提供的本机实现文件。

  注：请不要从该列表中删除 ExtensionBridge.cpp。请删除 HelloWorld 或 Process 扩展实现文件。通常，不要将您的扩展的源文件添加到该列表中。

  例如：

  ```
  SC_MODULE_SOURCE_DIR := $(SC_SOURCE_DIR_EDK)
  SC_MODULE_SOURCE_FILES := ExtensionBridge.cpp
  ```
- 设置 SC_PLATFORM_SOURCE_DIR 和 SC_PLATFORM_SOURCE_FILES 以指定您的扩展的本机实现文件。例如：

  ```
  SC_PLATFORM_SOURCE_DIR:= $(SC_PLATFORM_MAKEFILE_DIR)/edk/myExtension/native
  SC_PLATFORM_SOURCE_FILES := \
                  MyExtension.cpp \
                  helper\MyHelperClass1.cpp \
                  helper\MyHelperClass2.cpp
  ```
- 将 SC_EDK_AS_SOURCE_DIR 设置为包含您的扩展的实际 （而非存根）实现的 ActionScript 文件的目录。例如：

  ```
  SC_EDK_AS_SOURCE_DIR := $(SC_PLATFORM_MAKEFILE_DIR)/edk/myExtension/as/real
  ```

  注：该目录是您的 ActionScript 软件包的基本目录。例如，假定有一个名为 tv.adobe.extension.example 的 ActionScript 软件包。目录 tv、 adobe、 extension 和 example 是 SC_EDK_AS_SOURCE_DIR 的后续子目录。
- 设置 SC_EDK_AS_CLASSES 以列出实际 ActionScript 实现定义的每个 ActionScript 类。例如：

  ```
  SC_EDK_AS_CLASSES := MyExtension \
                       MyHelperClass1 \
                       MyHelperClass2
  ```
- 将 SC_EDK_AS_SOURCE_DIR_AUTHORING 设置为包含您的扩展的存根或模拟器实现的 ActionScript 文件的目录。例如：

  ```
  SC_EDK_AS_SOURCE_DIR_AUTHORING := $(SC_PLATFORM_MAKEFILE_DIR)/edk/myExtension/as/stub
  ```

  注：该目录是您的 ActionScript 软件包的基本目录。例如，假定有一个名为 tv.adobe.extension.example 的 ActionScript 软件包。目录 tv、 adobe、 extension 和 example 是 SC_EDK_AS_SOURCE_DIR_AUTHORING 的后续子目录。
- 设置 SC_EDK_AS_CLASSES_AUTHORING 以列出存根或模拟器 ActionScript 实现定义的每个 ActionScript 类。例如：

  ```
  SC_EDK_AS_CLASSES_AUTHORING := MyExtension \
                       MyHelperClass1 \
                       MyHelperClass2
  ```

安装第三方库

生成 AIR for TV 需要某些第三方库。有关这些库的详细信息，请参阅 Adobe AIR for TV 快速入门 (PDF) 中的 “ 安装第三方软件 ”。

如果您仅生成您的扩展模块，而不是生成所有 AIR for TV，则需要的库有：

- AIR 3 SDK

  从 http://www.adobe.com/cn/products/air/sdk/ 中选择下载 Mac OS X。

  创建一个目录以包含 .tbz2 文件的内容。例如：

  ```
  /usr/AIRSDK
  ```

  将 .tbz2 文件的内容解压到此目录中。

  ```
  tar jxf AdobeAIRSDK.tbz2
  ```

  设置 PATH 环境变量以包括 AIR SDK bin 目录。在本示例中，该 bin 目录是 /usr/AIRSDK/bin。

- Open Source Flex® SDK。

  从 http://opensource.adobe.com/wiki/display/flexsdk/Downloads 下载 Open Source Flex SDK 最新发行版的 ZIP 文件。

  创建一个目录以包含 ZIP 文件的内容。例如：

  ```
  /usr/flexSDK
  ```

  将 ZIP 文件的内容解压到此目录中。

  ```
  unzip flex_sdk_4.5.1.21328_mpl.zip
  ```

  设置 PATH 环境变量以包括 Flex SDK bin 目录。在本示例中，该 bin 目录是 /usr/flexSDK/bin。

- Java™ 运行时。 Flex SDK 需要最新的 Java 运行时。如果您的开发系统还不具有 Java 运行时，请访问 http://www.java.com/en/download/manual.jsp 获取下载包和安装说明。

  设置 PATH 环境变量以包括 Java bin 目录。

运行生成实用程序

可在以下位置查看有关如何使用 AIR for TV 生成实用程序的详细信息：

- Adobe AIR for TV 快速入门 (PDF) 中的 “ 安装和生成源分发包 ” 一章。
- 优化和集成 Adobe AIR for TV (PDF) 中的 “ 编码、生成和测试 ”。请按照 “ 创建平台 Makefile.config” 一节中的说明设置各种生成变量。

具体说来，在生成扩展时，生成实用程序在 Makefile.config 中使用下列生成变量：

- SC_ZIP
- SC_UNZIP
- SC_PLATFORM_NAME
- SC_PLATFORM_ARCH

在创建您的平台的 Makefile.config 文件和您的扩展的 .mk 文件后，您可以使用生成实用程序执行下列操作：

- 生成 AIR for TV 的所有组件。
- 仅生成您的扩展模块。

要生成 AIR for TV 的所有组件，请执行下列操作：

1 确保已设置环境变量 SC_BUILD_MODE 和 SC_PLATFORM。

2 如果您使用自己创建的证书对您的扩展进行签名，请设置环境变量 SC_EDK_ANE_CERT_FILE 和 SC_EDK_ANE_CERT_PASSWD。

将 SC_EDK_ANE_CERT_FILE 设置为您的证书的相对或绝对路径。相对路径相对于生成目录 <AIR for TV 安装目录>/stagecraft/build/linux。

将 SC_EDK_ANE_CERT_PASSWD 设置为该证书的密码。

如果您不设置这些环境变量，生成实用程序将使用默认的虚假证书，并且显示警告消息。该虚假证书仅适用于测试。

3 更改到目录：

```
<AIR for TV installation directory>/products/stagecraft/build/linux
```

4 输入以下命令：

```
make
```

要仅生成您的扩展模块，请执行下列操作：

1 确保已设置环境变量 SC_BUILD_MODE 和 SC_PLATFORM。

2 如果您使用自己创建的证书对您的扩展进行签名，请如前面的步骤所述，设置环境变量 SC_EDK_ANE_CERT_FILE 和 SC_EDK_ANE_CERT_PASSWD。

3 更改到目录 stagecraft/build/linux。

4 输入以下命令：

```
make PlatformEDKExtension_<your extension name>
```

您可以用以下命令删除以前为您的扩展生成的所有对象：

```
make clean-PlatformEDKExtension_<your extension name>
```

您可以用以下命令删除以前为您的扩展生成的所有对象，然后重新生成这些对象：

```
make rebuild-PlatformEDKExtension_<your extension name>
```

重要说明：如果您的生成计算机使用了防火墙，则生成实用程序有时会失败。防火墙可能禁止访问 ADT 在将本机扩展打包为 ANE 文件时使用的时间戳服务器。这一失败会产生以下错误输出：

```
Could not generate timestamp: Connection timed out
```

要避免该失败，请修改生成实用程序使用的 ADT 命令。编辑以下目录中的文件 extension.mk：

```
<AIR for TV installation directory>/stagecraft/source/ae/edk/
```

查找以下行：

```
$(SC_EXEC_CMD) $(SC_ADT) -package \
```

将参数 -tsa none 添加至该命令，如下所示：

```
$(SC_EXEC_CMD) $(SC_ADT) -package -tsa none\
```

生成实用程序扩展输出

生成实用程序为您的扩展创建两个文件。它将这些文件放入下列目录之一，具体取决于您为 SC_BUILD_MODE 指定了 debug 还是 release：

```
<AIR for TV installation directory>/build/stagecraft/linux/<yourPlatform>/debug/bin
<AIR for TV installation directory>/build/stagecraft/linux/<yourPlatform>/release/bin
```

生成实用程序为您的扩展创建的文件有：

- 一个 ZIP 文件，包含要在设备上部署的设备绑定扩展。

- 一个 ANE 文件，包含存根或模拟器扩展。AIR 应用程序开发人员使用该 ANE 文件来生成他们的应用程序。他们还使用它在使用 ADL 的台式计算机上测试他们的应用程序。他们还将该 ANE 文件与他们的应用程序一起打包为 AIRN 软件包。

同时生成存根和模拟器扩展

有时，除了实际扩展以外，您还想同时生成存根和模拟器扩展。通常，您指示 AIR 应用程序开发人员执行以下操作：

- 使用模拟器扩展在台式计算机上进行测试。
- 将存根扩展与他们的应用程序一起打包为 AIRN 软件包。

要同时生成存根和模拟器扩展，请执行以下操作：

1. 创建存根扩展及其 .mk 文件。确保您可以生成存根扩展和实际扩展。
2. 为您的模拟器实现创建一个目录，该目录需要是您的存根实现目录的兄弟目录。例如：

   ```
   <AIR for TV installation directory>/products/stagecraft/thirdparty-private/CompanyA/stagecraft-platforms/PlatformB/edk/myExtension/as/stub
   <AIR for TV installation directory>/products/stagecraft/thirdparty-private/CompanyA/stagecraft-platforms/PlatformB/edk/myExtension/as/simulator
   ```

3. 制作您的扩展的 .mk 文件的副本。
4. 在该副本中，编辑 SC_EDK_AS_SOURCE_DIR_AUTHORING 和 SC_EDK_AS_CLASSES_AUTHORING 的值。对这些值进行适当的设置，以反映您的模拟器实现目录和类。
5. 重命名您的扩展的原始 .mk 文件，以保证它的安全。然后，将该副本重命名为您的扩展的 .mk 文件名：PlatformEDKExtension_< 您的扩展名称 >.mk。
6. 将您的平台的 bin 目录中的存根 ANE 文件移至某个安全的位置。否则，下一步操作会覆盖该文件。
7. 运行生成实用程序以生成实际扩展和您的模拟器扩展。

将资源添加到您的 AIR for TV 本机扩展

一些 AIR for TV 本机扩展需要资源，如图像文件。如果您的扩展使用资源，请执行下列操作：

1. 从生成实用程序创建的 ZIP 文件中提取文件，并保留目录结构。
2. 将扩展需要的资源目录和文件添加到该目录结构中。
3. 将更新后的目录结构重新压缩为 ZIP 文件，并且使用与生成实用程序创建的 ZIP 文件相同的名称来命名该文件。
4. 从生成实用程序创建的 ANE 文件中提取文件，并保留目录结构。您可以使用您用来处理 ZIP 文件的同一工具。
5. 将扩展需要的资源目录和文件添加到该目录结构中。
6. 将更新后的目录结构重新压缩为 ZIP 文件。使用与生成实用程序创建的 ANE 文件相同的名称来重命名该文件。确保您将文件扩展名更改为 .ane。

您的资源在扩展目录结构内的准确子目录结构取决于您的扩展在何处寻找这些资源。扩展的 ActionScript 端可以使用 ExtensionContext.getExtensionDirectory() 来查找扩展的目录。有关详细信息，请参阅第 10 页的 “ 访问本机扩展的目录 ”。

分发 AIR for TV 本机扩展

在 AIR for TV 设备上安装设备绑定扩展

在生成设备绑定扩展之后，请将其安装到设备上。

生成实用程序创建一个 ZIP 文件，其中包含要安装到设备上的所有文件。请将该文件解压缩到设备上的以下位置：

```
/opt/adobe/stagecraft/extensions
```

注：/opt/adobe/stagecraft/extensions 目录还可以是指向设备文件系统中其他目录的 Linux 符号软链接。有关详细信息，请参阅优化和集成 Adobe AIR for TV (PDF) 中的 “ 文件系统用法 ” 一章。

将存根或模拟器扩展与 AIR 应用程序一起打包

为使 AIR 应用程序能够在设备上使用设备绑定扩展，请将存根或模拟器扩展与 AIR 应用程序一起打包。使用 ADT 可将存根或模拟器扩展的 ANE 文件与 AIR 应用程序一起打包，以创建 AIRN 软件包文件。AIRN 软件包文件与 AIR 软件包文件类似，但 AIRN 软件包文件包括扩展 ANE 文件。

有关详细信息，请参阅将 AIR for TV 应用程序打包。

在 AIR for TV 设备上运行 AIR 应用程序

有关如何在 AIR for TV 设备上安装和运行 AIR 应用程序的信息，请参阅 Adobe AIR for TV 快速入门 (PDF)。当您运行 stagecraft 二进制可执行文件时，请确保您使用以下命令行选项：

```
--profile extendedTV
```

要使 AIR 应用程序能够使用它的本机扩展，必须填写该选项。

当 AIR for TV 运行包含存根或模拟器扩展的应用程序时，AIR for TV 将在该设备上寻找相应的设备绑定扩展。如果该扩展存在， AIR 应用程序就会使用它。如果设备上未安装设备绑定扩展， AIR 应用程序将使用存根或模拟器扩展。

第 7 章：本机扩展描述符文件

扩展描述符文件描述本机扩展包的内容。

扩展描述符示例
以下扩展描述符文档描述以下各项的本机扩展：

- Android 设备
- 其他平台的默认 ActionScript 实现

```
<extension xmlns="http://ns.adobe.com/air/extension/3.1">
    <id>com.example.MyExtension</id>
    <versionNumber>0.0.1</versionNumber>
    <platforms>
        <platform name="Android-ARM">
            <applicationDeployment>
                <nativeLibrary>MyExtension.jar</nativeLibrary>
                <initializer>com.sample.ext.MyExtension</initializer>
            </applicationDeployment>
        <platform name="default">
            <applicationDeployment/>
        </platform>
    </platforms>
</extension>
```

扩展描述符文件结构

扩展描述符文件是具有以下结构的 XML 文档：

```
<extension xmlns="http://ns.adobe.com/air/extension/2.5">
    <id>...</id>
    <versionNumber>...</versionNumber>
    <name>
        <text xml:lang="language_code">...</text>
    </name>
    <description>
        <text xml:lang="language_code">...</text>
    </description>
    <platforms>
        <platform name="device">
            <applicationDeployment>
                <nativeLibrary>...</nativeLibrary>
                <initializer>...</initializer>
                <finalizer>...</finalizer>
            </applicationDeployment>
        </platform>
        <platform name="device">
            <deviceDeployment/>
        <platform name="default">
            <applicationDeployment/>
    </platforms>
</extension>
```

本机扩展描述符元素

下面的元素列表描述了 AIR 应用程序描述符文件的各个合法元素。

applicationDeployment

声明扩展包中附带的并随应用程序一起部署的本机代码库。

每个 platform 元素必须包含 applicationDeployment 元素或 deviceDeployment 元素，但不能同时包含两者。

父元素：platform。

子元素：

- finalizer
- initializer
- nativeLibrary

内容
标识本机代码库以及初始化和终止化函数。当平台名称为 default 时， applicationDeployment 元素没有子元素，因为 default 平台没有本机代码库。

示例

```
<applicationDeployment>
    <nativeLibrary>myExtension.so</nativeLibrary>
    <initializer>com.example.extension.Initializer</initializer>
    <finalizer>com.example.extension.Finalizer</finalizer>
</applicationDeployment>
```

copyright

可选

扩展的版权声明。

父元素：extension

子元素：无

内容
包含版权信息的字符串。

示例

```
<copyright>© 2010, Examples, Inc. All rights reserved.</copyright>
```

description

可选

扩展的描述。

父元素：extension

子元素：text

内容
使用简单文本节点或多个 text 元素。

使用多个 text 元素，可在 description 元素中指定多种语言。每个文本元素的 xml:lang 属性用于指定语言代码，有关具体定义，请参阅 RFC4646 (http://www.ietf.org/rfc/rfc4646.txt)。

示例
使用简单文本节点的说明：

```
<description>This is a sample native extension for Adobe AIR.</description>
```

对英语、法语和西班牙语使用本地化文本元素的描述：

```
<description>
    <text xml:lang="en">This is a example.</text>
    <text xml:lang="fr">C'est un exemple.</text>
    <text xml:lang="es">Esto es un ejemplo.</text>
</description>
```

deviceDeployment

声明一个本机扩展，在设备上为其单独部署代码库且该代码库不包含在此扩展包中。

并非所有平台都支持设备部署。

每个 platform 元素必须包含 applicationDeployment 元素或 deviceDeployment 元素，但不能同时包含两者。

父元素：platform

子元素：无。

内容
无。 deviceDeployment 元素必须为空。

示例

```
<deviceDeployment/>
```

extension

必需

扩展描述符文档的根元素。

父元素：无。

子元素：

- copyright
- description
- id
- name
- platforms
- versionNumber

内容
标识支持的平台和每个平台的代码库。

extension 元素包含一个名为 xmlns 的命名空间属性。将 xmlns 值设置为以下值之一：

```
xmlns="http://ns.adobe.com/air/extension/3.1"
xmlns="http://ns.adobe.com/air/extension/2.5"
```

命名空间是决定程序兼容性的一个因素，它与 SWF 版本一起决定 AIR SDK 和 ANE 文件之间的兼容性。打包 AIR 应用程序所使用的 AIR SDK 版本必须等于或高于扩展命名空间。因此，AIR 3 应用程序可以使用命名空间为 2.5 的扩展，但不能使用命名空间为 3.1 的扩展。

示例

```
<extension xmlns="http://ns.adobe.com/air/extension/2.5">
    <id>com.example.MyExtension</id>
    <versionNumber>1.0.1</versionNumber>
    <platforms>
        <platform name="Polyphonic-MIPS">
            <deviceDeployment/>
        </platform>
        <platform name="NeoTech-ARM">
            <deviceDeployment/>
        </platform>
        <platform name="Philsung-x86">
            <deviceDeployment/>
        </platform>
        <platform name="default">
            <applicationDeployment/>
        </platform>
    </platforms>
</extension>
```

finalizer

可选

本机库中定义的终止化函数。

父元素：applicationDeployment

子元素：无。

内容
如果扩展使用其本机库中的 C API，则为终结器函数的名称。

如果扩展使用 Java API，则此元素包含实现 FREExtension 接口的类的名称。

此值可包含以下字符：A - Z、 a - z、 0 - 9、句点 (.) 和短划线 (-)。

示例

```
<finalizer>...</finalizer>
```

id

必需

扩展的 ID。

父元素：extension

子元素：无。

内容
指定扩展的 ID。

此值可包含以下字符：A - Z、 a - z、 0 - 9、句点 (.) 和短划线 (-)。

示例
```
<id>com.example.MyExtension</id>
```

initializer

可选

本机库中定义的初始化函数。如果使用了 nativeLibrary 元素，则需要 initializer 元素。

父元素：applicationDeployment

子元素：无。

内容
如果扩展使用其本机库中的 C API，则为初始化函数的名称。

如果扩展使用 Java API，则此元素包含实现 FREExtension 接口的类的名称。

此值可包含以下字符：A - Z、 a - z、 0 - 9、句点 (.) 和短划线 (-)。

示例
```
<initializer>...</initializer>
```

name

可选

扩展的名称。

父元素：extension

子元素：text

内容
如果指定单个文本节点 （而非多个 <text> 元素），则无论系统语言为哪种语言， AIR 应用程序安装程序都将使用此名称。

每个文本元素的 xml:lang 属性用于指定语言代码，有关具体定义，请参阅 RFC4646 (http://www.ietf.org/rfc/rfc4646.txt)。

示例
以下示例使用简单文本节点定义名称。

```
<name>Test Extension</name>
```

以下示例使用 <text> 元素节点指定三种语言 （英语、法语和西班牙语）的名称：

```
<name>
    <text xml:lang="en">Hello AIR</text>
    <text xml:lang="fr">Bonjour AIR</text>
    <text xml:lang="es">Hola AIR</text>
</name>
```

nativeLibrary

可选

平台的扩展包中附带的本机库文件。请考虑以下示例：

- 如果扩展仅包含 ActionScript 代码，则不需要 nativeLibrary 元素。
- 如果未使用 nativeLibrary 元素，则也无法使用 initializer 和 finalizer 元素。
- 如果使用 nativeLibrary 元素，则还需要 initializer 元素。

父元素：applicationDeployment

子元素：无。

内容
扩展包中附带的本机库的文件名。

此值可包含以下字符：A - Z、 a - z、 0 - 9、句点 (.) 和短划线 (-)。

示例

```
<nativeLibrary>extensioncode.so</nativeLibrary>
```

platform

必需

指定扩展在特定平台上的本机代码库。

父元素：platforms

子元素：以下元素之一 （仅限一个）：

- applicationDeployment
- deviceDeployment

内容
name 属性指定平台的名称。扩展开发人员可以使用专用的 default 平台名称来包括一个 ActionScript 库，用于在不支持的平台上模拟本机代码行为。模拟的行为可用于支持调试，并为多平台应用程序提供回退行为。

对 name 属性使用以下值：

- Android-ARM，对于 Android 设备。
- default
- iPhone-ARM，对于 iOS 设备。
- iPhone-x86，对于 iOS Simulator。
- MacOS-x86-64，对于 Mac OS X 设备。
- QNX-ARM，对于 Blackberry Tablet OS 设备。

- Windows-x86，对于 Windows 设备。

注：设备绑定扩展使用设备制造商定义的 name 属性值。

子元素指定本机代码库的部署方式。应用程序部署是指代码库随使用它的每个 AIR 应用程序一起部署。代码库必须包含在扩展包中。设备部署是指将代码库单独部署到平台，而不包含在扩展包中。这两种部署类型相互排斥；只能包括一个部署元素。

示例

```
<platform name="Philsung-x86">
    <deviceDeployment/>
</platform>
<platform name="default">
    <applicationDeployment/>
</platform>
```

platforms

必需

指定此扩展支持的平台。

父元素：extension

子元素：platform

内容

每个支持的平台对应一个 platform 元素。另外，还可以指定一个包含 ActionScript 实现的专用的 default 平台，以便在不支持特定代码库的平台上使用。

示例

```
<platforms>
    <platform name="Android-ARM">
        <applicationDeployment>
            <nativeLibrary>MyExtension.jar</nativeLibrary>
            <initializer>com.sample.ext.MyExtension</initializer>
            <finalizer>com.sample.ext.MyExtension</finalizer>
        </applicationDeployment>
    </platform>
    <platform name="iPhone-ARM">
        <applicationDeployment>
            <nativeLibrary>MyExtension.a</nativeLibrary>
            <initializer>InitMyExtension></initializer>
        </applicationDeployment>
    <platform name="Philsung-x86">
        <deviceDeployment/>
    </platform>
    <platform name="default">
        <applicationDeployment/>
    </platform>
</platforms>
```

text

可选

指定本地化字符串。

文本元素的 xml:lang 属性用于指定语言代码，有关具体定义，请参阅 RFC4646 (http://www.ietf.org/rfc/rfc4646.txt)。

AIR 使用具有与用户操作系统的用户界面语言最匹配的 xml:lang 属性值的 text 元素。

例如，考虑一种安装，text 元素将针对 en （英语）区域设置的值包括在该安装中。如果操作系统将 en （英语）标识为用户界面语言，则 AIR 将使用此 en 名称。如果系统用户界面语言为 en-US （美式英语），则该应用程序也使用此 en 名称。但是，如果用户界面语言为 en-US，而应用程序描述符文件同时定义了 en-US 名称和 en-GB 名称，则 AIR 应用程序安装程序将使用相应的 en-US 值。

如果应用程序定义的任何 text 元素与系统用户界面语言均不匹配，则 AIR 将使用在扩展描述符文件中定义的第一个 name 值。

父元素：

- name
- description

子元素：无

内容
指定区域设置和本地化文本字符串的 xml:lang 属性。

示例

```
<text xml:lang="fr">Bonjour AIR</text>
```

versionNumber

必需

扩展版本号。

父元素：extension

子元素：无

内容
版本号可以包含按时期分隔的三个整数组成的序列。每个整数必须是介于 0 到 999 （两者都包括）之间的数字。

示例

```
<versionNumber>1.0.657</versionNumber>

<versionNumber>10</versionNumber>

<versionNumber>0.01</versionNumber>
```

第 8 章：本机 C API 参考

有关使用本机 C API 的本机扩展示例，请参阅 Adobe AIR 的本机扩展。

Typedef

FREContext

AIR 3.0 和更高版本

```
typedef void* FREContext;
```

ActionScript 端通过调用 ExtensionContext.createExtensionContext() 来创建扩展上下文。 AIR 运行时会为每个扩展上下文创建一个相应的 FREContext 变量。

运行时在本机实现中将 FREContext 变量传递给下面的每个函数：

- 上下文初始化函数， FREContextInitializer()。
- 上下文终结器函数， FREContextFinalizer()。
- 每个扩展函数， FREFunction()。

在下列情况下使用此 FREContext 变量：

- 将某个事件调度到 ActionScript ExtensionContext 实例时。请参阅 FREDispatchStatusEventAsync()。
- 获取或设置本机上下文数据时。请参阅 第 86 页的 “FREGetContextNativeData()” 和 第 100 页的 “FRESetContextNativeData()”。
- 获取或设置 ActionScript 上下文数据时。请参阅 第 86 页的 “FREGetContextActionScriptData()” 和 第 99 页的 “FRESetContextActionScriptData()”。

FREObject

AIR 3.0 和更高版本

```
typedef void* FREObject;
```

从本机 C 实现访问 ActionScript 类对象或基元数据类型变量时，使用 FREObject 变量。运行时将 FREObject 变量与相应的 ActionScript 对象关联。

FREObject 变量用于使用签名 FREFunction() 实现的本机函数中。本机函数：

- 将 FREObject 变量作为参数接收。
- 返回 FREObject 变量。

当使用本机扩展 C API 函数执行以下操作时，也会使用 FREObject 变量：

- 创建 ActionScript 类对象或 ActionScript 基元数据类型。
- 获取 ActionScript 类对象或 ActionScript 基元数据类型的值。
- 创建 ActionScript String 对象。
- 获取 ActionScript String 对象的值。

- 获取或设置 ActionScript 对象的属性。
- 调用 ActionScript 对象的方法。
- 访问 ActionScript BitmapData 对象的位。
- 访问 ActionScript ByteArray 对象的字节。
- 获取或设置 ActionScript Array 或 Vector 对象的长度。
- 获取或设置 ActionScript Array 或 Vector 对象的元素。
- 当运行时在本机扩展 C API 函数调用中引发异常时，获取 ActionScript Error 对象。
- 在 ActionScript 上下文数据中设置或获取 ActionScript 对象。

结构 typedef

FREBitmapData

AIR 3.0

当在 ActionScript BitmapData 类对象中获取和操作位时，使用 FREBitmapData 或 FREBitmapData2 结构。FREBitmapData 结构定义如下：

```
typedef struct {
    uint32_t    width;
    uint32_t    height;
    uint32_t    hasAlpha;
    uint32_t    isPremultiplied;
    uint32_t    lineStride32;
    uint32_t*   bits32;
} FREBitmapData;
```

FREBitmapData 的字段含义如下：

width uint32_t，以像素为单位指定位图的宽度。此值对应于 ActionScript BitmapData 类对象的 width 属性。该字段为只读字段。

height uint32_t，以像素为单位指定位图的高度。此值对应于 ActionScript BitmapData 类对象的 height 属性。该字段为只读字段。

hasAlpha 一个 uint32_t 值，指示位图是否支持每个像素具有不同的透明度。此值对应于 ActionScript BitmapData 类对象的 transparent 属性。如果值为非零，则像素格式为 ARGB32。如果值为零，则像素格式为 _RGB32。此值为 big endian 还是 little endian 取决于主机设备。该字段为只读字段。

isPremultiplied 一个 uint32_t 值，指示是否将位图像素作为预乘颜色值进行存储。非零值意味着值会预乘。该字段为只读字段。有关预乘颜色值的详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考中的 “BitmapData.getPixel()”。

lineStride32 uint32_t，指定每个扫描线的 uint32_t 值数量。此值通常与 width 参数相同。该字段为只读字段。

bits32 指向 uint32_t 的指针。此值为一个 uint32_t 值数组。每个值都是位图的一个像素。

注：您可以在本机实现中更改的唯一一个 FREBitmapData 结构字段是 bits32 字段。 bits32 字段包含实际位图值。请将 FREBitmapData 结构中的所有其他字段都视为只读字段。

更多帮助主题

第 73 页的 “FREBitmapData2”

第 80 页的 “FREAcquireBitmapData()”

第 92 页的 “FREInvalidateBitmapDataRect()”

FREBitmapData2

AIR 3.1 和更高版本

FREBitmapData2 结构会向 FREBitmapData 结构中添加 isInvertedY 字段。在其他方面，这两个结构是相同的。FREBitmapData2 结构定义如下：

```
typedef struct {
    uint32_t    width;
    uint32_t    height;
    uint32_t    hasAlpha;
    uint32_t    isPremultiplied;
    uint32_t    lineStride32;
    uint32_t    isInvertedY
    uint32_t*   bits32;
} FREBitmapData2;
```

FREBitmapData2 的字段含义如下：

width uint32_t，以像素为单位指定位图的宽度。此值对应于 ActionScript BitmapData 类对象的 width 属性。该字段为只读字段。

height uint32_t，以像素为单位指定位图的高度。此值对应于 ActionScript BitmapData 类对象的 height 属性。该字段为只读字段。

hasAlpha 一个 uint32_t 值，指示位图是否支持每个像素具有不同的透明度。此值对应于 ActionScript BitmapData 类对象的 transparent 属性。如果值为非零，则像素格式为 ARGB32。如果值为零，则像素格式为 _RGB32。此值为 big endian 还是 little endian 取决于主机设备。该字段为只读字段。

isPremultiplied 一个 uint32_t 值，指示是否将位图像素作为预乘颜色值进行存储。非零值意味着值会预乘。该字段为只读字段。有关预乘颜色值的详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考中的 “BitmapData.getPixel()”。

lineStride32 uint32_t，指定每个扫描线的 uint32_t 值数量。此值通常与 width 参数相同。该字段为只读字段。

isInvertedY 是一个 uint32_t，用于指定存储图像位图数据行的顺序。非零值表示在图像数据中首先显示图像的最后一行（就是说，bits32 数组中的第一个值是图像最后一行的第一个像素。值为零表示在图像数据中首先显示图像的第一行。该字段为只读字段。

bits32 指向 uint32_t 的指针。此值为一个 uint32_t 值数组。每个值都是位图的一个像素。

注：您可以在本机实现中更改的唯一一个 FREBitmapData2 结构字段是 bits32 字段。 bits32 字段包含实际位图值。请将 FREBitmapData2 结构中的所有其他字段都视为只读字段。

更多帮助主题

第 72 页的 “FREBitmapData”

第 81 页的 “FREAcquireBitmapData2()”

第 92 页的 “FREInvalidateBitmapDataRect()”

FREByteArray

AIR 3.0 和更高版本

当在 ActionScript ByteArray 类对象中获取和操作字节时，使用 FREByteArray 结构。该结构的定义如下：

```
typedef struct {
    uint32_t     length;
    uint8_t*     bytes;
} FREByteArray;
```

FREByteArray 的字段含义如下：

length uint32_t，bytes 数组中的字节数。

bytes uint8_t*，一个指向 ActionScript ByteArray 对象中的字节的指针。

更多帮助主题

第 81 页的 “FREAcquireByteArray()”

FRENamedFunction

AIR 3.0 和更高版本

使用 FRENamedFunction 将您编写的 FREFunction 与一个名称关联。当使用 ExtensionContext 实例的 call() 方法调用本机函数时，在您的 ActionScript 代码中使用该名称。该结构的定义如下：

```
typedef struct FRENamedFunction_{
    const uint8_t*   name;
    void*            functionData;
    FREFunction      function;
} FRENamedFunction;
```

FRENamedFunction 的字段含义如下：

name const uint8_t* 指针，此指针指向 ActionScript 端用于调用关联的 C 函数的字符串。即，字符串值为 ActionScript ExtensionContext call() 方法在其 functionName 参数中使用的名称。字符串使用 UTF-8 编码，并以 null 字符终止。

functionData void* 指针，此指针指向您希望与此 FREFunction 函数相关联的任何数据。当运行时调用 FREFunction 函数时，它会向该函数传递此数据指针。

function FRENamedFunction，运行时与 name 字段给定的字符串关联的函数。使用 FREFunction() 的签名定义此函数。

枚举

FREObjectType

AIR 3.0 和更高版本

一个 FREObject 变量对应一个 ActionScript 类对象或基元类型。 FREObjectType 枚举定义这些 ActionScript 类类型和基元类型的值。C API 函数 FREGetObjectType() 返回一个最能准确描述 FREObject 变量的对应 ActionScript 类对象或基元类型的 FREObjectType 枚举值。

```
enum FREObjectType {
    FRE_TYPE_OBJECT             = 0,
    FRE_TYPE_NUMBER             = 1,
    FRE_TYPE_STRING             = 2,
    FRE_TYPE_BYTEARRAY          = 3,
    FRE_TYPE_ARRAY              = 4,
    FRE_TYPE_VECTOR             = 5,
    FRE_TYPE_BITMAPDATA         = 6,
    FRE_TYPE_BOOLEAN            = 7,
    FRE_TYPE_NULL               = 8,
    FREObjectType_ENUMPADDING   = 0xfffff
};
```

各枚举值的含义如下：

FRE_TYPE_OBJECT FREObject 变量对应于一个除 String 对象、 ByteArray 对象、 Array 对象、 Vector 对象或 BitmapData 对象以外的 ActionScript 类对象。

FRE_TYPE_NUMBER FREObject 变量对应于一个 ActionScript Number 变量。

FRE_TYPE_STRING FREObject 变量对应于一个 ActionScript String 对象。

FRE_TYPE_BYTEARRAY FREObject 变量对应于一个 ActionScript ByteArray 对象。

FRE_TYPE_ARRAY FREObject 变量对应于一个 ActionScript Array 对象。

FRE_TYPE_VECTOR FREObject 变量对应于一个 ActionScript Vector 对象。

FRE_TYPE_BITMAPDATA FREObject 变量对应于一个 ActionScript BitmapData 对象。

FRE_TYPE_BOOLEAN FREObject 变量对应于一个 ActionScript Boolean 变量。

FRE_TYPE_NULL FREObject 变量对应于 ActionScript 值 Null 或 undefined。

FREObjectType_ENUMPADDING 这是最后一个枚举值，用于保证枚举值的大小始终为 4 个字节。

更多帮助主题

第 18 页的 “FREObject 类型 ”

FREResult

AIR 3.0 和更高版本

FREResult 枚举定义您调用的本机扩展 C API 函数的返回值。

```
enum FREResult {
    FRE_OK                  = 0,
    FRE_NO_SUCH_NAME        = 1,
    FRE_INVALID_OBJECT      = 2,
    FRE_TYPE_MISMATCH       = 3,
    FRE_ACTIONSCRIPT_ERROR  = 4,
    FRE_INVALID_ARGUMENT    = 5,
    FRE_READ_ONLY           = 6,
    FRE_WRONG_THREAD        = 7,
    FRE_ILLEGAL_STATE       = 8,
    FRE_INSUFFICIENT_MEMORY = 9,
    FREResult_ENUMPADDING   = 0xffff
};
```

各枚举值的含义如下：

FRE_OK 函数已成功。

FRE_ACTIONSCRIPT_ERROR 发生 ActionScript 错误，并且引发了异常。可导致此错误的 C API 函数允许您指定一个 FREObject 以接收异常的相关信息。

FRE_ILLEGAL_STATE 当对本机扩展 C API 函数执行调用时，扩展上下文对于该调用来说处于非法状态。以下情况下会出现此返回值。上下文获取了对 ActionScript BitmapData 或 ByteArray 类对象的访问。唯一的例外是，在释放 BitmapData 或 ByteArray 对象之前，上下文无法调用任何其他 C API 函数。一个例外情况是，上下文可以在调用了 FREAcquireBitmapData() 或 FREAcquireBitmapData2() 之后调用 FREInvalidateBitmapDataRect()。

FRE_INSUFFICIENT_MEMORY 运行时无法分配足够的内存以更改 Array 或 Vector 对象的大小。

FRE_INVALID_ARGUMENT 一个指针参数为 NULL。

FRE_INVALID_OBJECT 一个 FREObject 参数无效。有关无效 FREObject 变量的示例，请参阅 第 19 页的 “FREObject 有效性 ”。

FRE_NO_SUCH_NAME 作为参数传递的类、属性或方法的名称与 ActionScript 类名称、属性或方法不匹配。

FRE_READ_ONLY 该函数尝试修改 ActionScript 对象的只读属性。

FRE_TYPE_MISMATCH FREObject 参数不表示调用的函数所需的 ActionScript 类的对象。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

FREResult_ENUMPADDING 这是最后一个枚举值，用于保证枚举值的大小始终为 4 个字节。

实现的函数

扩展 C API 提供了您在扩展的本机 C 实现中实现的函数的签名。

FREContextFinalizer()

AIR 3.0 和更高版本

签名

```
typedef void (*FREContextFinalizer)(
        FREContext ctx,
);
```

参数

ctx 表示此扩展上下文的 FREContext 变量。请参阅 第 71 页的 “FREContext”。

返回

无。

说明

运行时在释放此扩展上下文的 ExtensionContext 实例时调用此函数。以下情况导致运行时释放示例：

- ActionScript 端调用 ExtensionContext 实例的 dispose() 方法。
- 运行时的垃圾回收器未检测到对 ExtensionContext 实例的引用。
- 正在关闭 AIR 应用程序。

实现此函数可以清理特定于此扩展上下文的资源。使用 ctx 参数可以获取并清理与本机上下文数据和 ActionScript 上下文数据关联的资源。请参阅第 15 页的 “ 上下文特定数据 ”。

运行时在调用了此函数后，不会再调用此扩展上下文的任何其他函数。

更多帮助主题

第 14 页的 “ 扩展上下文初始化 ”

第 16 页的 “ 扩展上下文终止化 ”

FREContextInitializer()

AIR 3.0 和更高版本

签名

```
typedef void (*FREContextInitializer)(
        void*                   extData,
        const uint8_t*          ctxType,
        FREContext              ctx,
        uint32_t*               numFunctionsToSet,
        const FRENamedFunction** functionsToSet
);
```

参数

extData 指向本机扩展的扩展数据的指针。函数 FREInitializer() 会创建扩展数据。

ctxType 标识上下文类型的字符串。可根据扩展的需要定义此字符串。上下文类型可表示扩展的 ActionScript 端与本机端之间的任何约定含义。如果您的扩展不使用上下文类型，此值可以是 Null。此值为 UTF-8 编码的字符串，以 null 字符终止。

ctx FREContext 变量。运行时创建此值并将其传递给 FREContextInitializer()。请参阅 第 71 页的 “FREContext”。

numFunctionsToSet 指向 unint32_t 的指针。将 numFunctionsToSet 设置为包含 functionsToSet 参数中的函数数量的 unint32_t 变量。

functionsToSet 指向 FRNamedFunction 元素数组的指针。每个元素都包含一个指向本机函数的指针，以及 ActionScript 端在 ExtensionContext 实例的 call() 方法中使用的字符串。请参阅 第 74 页的 “FRENamedFunction”。

返回

无。

说明

当 ActionScript 端调用 ExtensionContext.createExtensionContext() 时，运行时会调用此方法。实现此方法可执行以下操作：

- 初始化扩展上下文。初始化可能取决于 ctxType 参数中传递的上下文类型。
- 保存 ctx 参数的值，以便其可用于本机实现对 FREDispatchStatusEventAsync() 的调用。
- 使用 ctx 参数初始化上下文特定数据。此数据包括上下文特定的本机数据和上下文特定的 ActionScript 数据。
- 设置 FRENamedFunction 对象的数组。返回一个指向 functionsToSet 参数中的数组的指针。返回一个指向 numFunctionsToSet 参数中的数组元素数量的指针。

FREContextInitializer() 方法的行为取决于 ctxType 参数。 ActionScript 端可将上下文类型传递给 ExtensionContext.createExtensionContext()。然后，运行时将值传递给 FREContextInitializer()。此函数通常使用上下文类型选择本机实现中 ActionScript 端可以调用的方法集。每个上下文类型都对应一个不同的方法集。

更多帮助主题

第 7 页的 “ 上下文类型 ”

第 14 页的 “ 扩展上下文初始化 ”

FREFinalizer()

AIR 3.0 和更高版本

签名

```
typedef void (*FREFinalizer)(
        void* extData,
);
```

参数

extData 指向扩展的扩展数据的指针。

返回

无。

说明

运行时在卸载扩展时调用此函数。但是，运行时不保证其将卸载扩展或调用 FREFinalizer()。

实现此函数可清理扩展资源。

更多帮助主题

第 79 页的 “FREInitializer()”

第 17 页的 “扩展终止化 ”

FREFunction()

AIR 3.0 和更高版本

签名

```
typedef FREObject (*FREFunction)(
        FREContext   ctx,
        void*        functionData,
        uint32_t     argc,
        FREObject    argv[]
);
```

参数

ctx 表示此扩展上下文的 FREContext 变量。请参阅 第 71 页的 “FREContext”。

functionData void* 指针，此指针指向您在其 FRENamedFunction 结构中与此 FREFunction 函数关联的数据。FREContextInitializer() 实现在输出参数中将一组 FRENamedFunction 结构传递给运行时。当运行时调用 FREFunction 函数时，它会向该函数传递此数据指针。请参阅 第 74 页的 “FRENamedFunction”。

argc argv 参数中元素的数量。

argv FREObject 变量数组。这些指针对应于在 ActionScript 调用中的函数名称后面传递给 ExtensionContext 实例的 call() 方法的参数。

返回

FREObject 变量。默认返回值是类型为 FRE_INVALID_OBJECT 的 FREObject。

说明
对扩展中的每个本机函数使用 FREFunction 签名实现函数。函数名称对应于 FREContextInitializer() 函数的 functionsToSet 参数中返回的数组中的 FRENamedFunction 元素的 function 字段。

当 ActionScript 端调用 ExtensionContext 实例的 call() 方法时，运行时调用此 FREFunction 函数。 call() 的第一个参数与 FRENamedFunction 元素的 name 字段相同。 call() 的后续参数对应于 argv 数组中的 FREObject 变量。

定义 FREFunction 可返回 FREObject 变量。 FREObject 变量的类型对应于 call() 方法返回的 ActionScript 类型。如果 FREFunction 无返回值，则默认返回值是类型为 FRE_INVALID_OBJECT 的 FREObject 变量。默认返回值会导致 call() 在 ActionScript 端返回 null。

使用 ctx 参数可执行以下操作：

- 获取和设置与扩展关联的数据。此数据可以是本机上下文数据，也可以是 ActionScript 上下文数据。请参阅第 15 页的 “ 上下文特定数据 ”。
- 在 ActionScript 端将一个异步事件调度给 ExtensionContext 实例。请参阅 第 83 页的 “FREDispatchStatusEventAsync()”。

使用第 80 页的 “ 所使用的函数 ” 中的函数处理 FREObject 参数和返回值 （如果有）。

更多帮助主题
第 77 页的 “FREContextInitializer()”

第 18 页的 “FREObject 类型 ”

FREInitializer()

AIR 3.0 和更高版本

签名

```
typedef void (*FREInitializer)(
        void**                  extDataToSet,
        FREContextInitializer*  ctxInitializerToSet,
        FREContextFinalizer*    contextFinalizerToSet
);
```

参数
extDataToSet 指向指向本机扩展的扩展数据的指针的指针。创建一个数据结构以保存扩展特定数据。例如，从堆分配数据，或提供全局数据。将 extDataToSet 设置为指向所分配数据的指针。

ctxInitializerToSet 指向指向 FREContextInitializer() 函数的指针的指针。将 ctxInitializerToSet 设置为所定义的 FREContextInitializer() 函数。

ctxFinalizerToSet 指向指向 FREContextFinalizer() 函数的指针的指针。将 ctxFinalizerToSet 设置为所定义的 FREContextFinalizer() 函数。您可将此指针设置为 NULL。

返回
无。

说明
运行时在加载扩展时调用一次此方法。实现此函数可执行您的扩展所需的任何初始化任务。然后设置输出参数。

更多帮助主题

第 77 页的 “FREContextInitializer()”

第 76 页的 “FREContextFinalizer()”

所使用的函数

本机扩展 C API 提供的函数可用于访问和操作 ActionScript 对象和基元数据。

FREAcquireBitmapData()

AIR 3.0 和更高版本

用法

```
FREResult FREAcquireBitmapData (
            FREObject        object,
            FREBitmapData*   descriptorToSet
);
```

参数

object FREObject。此 FREObject 参数表示一个 ActionScript BitmapData 类对象。

descriptorToSet 指向类型为 FREBitmapData 的变量的指针。当本机 C 实现调用此方法时，运行时设置此结构的字段。请参阅 第 72 页的 “FREBitmapData”。

返回

FREResult，可能的返回值包括 （但不限于）下列值:

FRE_OK 函数已成功。设置了 FREBitmapData 参数。 ActionScript BitmapData 对象可供您操作。

FRE_ILLEGAL_STATE 扩展上下文已获取一个 ActionScript BitmapData 对象或 ByteArray 对象。在释放该 BitmapData 或 ByteArray 对象之前，上下文无法调用此方法。

FRE_INVALID_ARGUMENT descriptorToSet 参数为 NULL。

FRE_INVALID_OBJECT FREObject object 参数无效。

FRE_TYPE_MISMATCH FREObject object 参数不表示 ActionScript BitmapData 类对象。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明

调用此函数可获取 ActionScript BitmapData 类对象的位图。成功调用此函数之后，在您调用 FREReleaseBitmapData() 之前，将无法成功调用任何其他 C API 函数。存在这种限制的原因是其他调用执行的代码会使指向位图内容的指针无效。

调用了此函数后，便可操作 BitmapData 对象的位图。 descriptorToSet 参数中提供了位图以及有关位图的其他信息。要通知运行时位图或位图的子矩形已更改，请调用 FREInvalidateBitmapDataRect()。已完成位图处理后，调用 FREReleaseBitmapData()。

更多帮助主题

第 97 页的 “FREReleaseBitmapData()”

第 92 页的 “FREInvalidateBitmapDataRect()”

FREAcquireBitmapData2()

AIR 3.1 和更高版本

用法

```
FREResult FREAcquireBitmapData2 (
            FREObject       object,
            FREBitmapData2* descriptorToSet
);
```

参数

object FREObject。此 FREObject 参数表示一个 ActionScript BitmapData 类对象。

descriptorToSet 指向类型为 FREBitmapData2 的变量的指针。当本机 C 实现调用此方法时，运行时设置此结构的字段。请参阅 第 73 页的 “FREBitmapData2”。

返回

FREResult，可能的返回值包括 （但不限于）下列值：

FRE_OK 函数已成功。设置了 FREBitmapData2 参数。 ActionScript BitmapData 对象可供您操作。

FRE_ILLEGAL_STATE 扩展上下文已获取一个 ActionScript BitmapData 对象或 ByteArray 对象。在释放该 BitmapData 或 ByteArray 对象之前，上下文无法调用此方法。

FRE_INVALID_ARGUMENT descriptorToSet 参数为 NULL。

FRE_INVALID_OBJECT FREObject object 参数无效。

FRE_TYPE_MISMATCH FREObject object 参数不表示 ActionScript BitmapData 类对象。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明

调用此函数可获取 ActionScript BitmapData 类对象的位图。成功调用此函数之后，在您调用 FREReleaseBitmapData() 之前，将无法成功调用任何其他 C API 函数。存在这种限制的原因是其他调用执行的代码会使指向位图内容的指针无效。

调用了此函数后，便可操作 BitmapData 对象的位图。 descriptorToSet 参数中提供了位图以及有关位图的其他信息。要通知运行时位图或位图的子矩形已更改，请调用 FREInvalidateBitmapDataRect()。已完成位图处理后，调用 FREReleaseBitmapData()。

更多帮助主题

第 97 页的 “FREReleaseBitmapData()”

第 92 页的 “FREInvalidateBitmapDataRect()”

FREAcquireByteArray()

AIR 3.0 和更高版本

用法

```
FREResult FREAcquireByteArray (
            FREObject       object,
            FREByteArray*   byteArrayToSet
);
```

参数

object FREObject。此 FREObject 参数表示一个 ActionScript ByteArray 类对象。

byteArrayToSet 指向类型为 FREByteArray 的变量的指针。当本机 C 实现调用此方法时，运行时设置此结构的字段。请参阅 第 74 页的 “FREByteArray”。

返回

FREResult，可能的返回值包括 （但不限于）下列值：

FRE_OK 函数已成功。设置了 FREByteArray 参数。 ActionScript ByteArray 对象可供您操作。

FRE_ILLEGAL_STATE 扩展上下文已获取一个 ActionScript BitmapData 对象或 ByteArray 对象。在释放该 BitmapData 或 ByteArray 对象之前，上下文无法调用此方法。

FRE_INVALID_ARGUMENT byteArrayToSet 参数为 NULL。

FRE_INVALID_OBJECT FREObject object 参数无效。

FRE_TYPE_MISMATCH FREObject object 参数不表示 ActionScript ByteArray 类对象。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明

调用此函数可获取 ActionScript ByteArray 类对象的字节。成功调用此函数之后，在您调用 FREReleaseByteArray() 之前，将无法成功调用任何其他 C API 函数。存在这种限制的原因是其他调用执行的代码会使指向字节数组内容的指针无效。

调用了此函数后，便可操作 ByteArray 对象的字节。 byteArrayToSet 参数中提供了字节以及字节的数量。已完成位图处理后，调用 FREReleaseByteArray()。

更多帮助主题

第 97 页的 “FREReleaseByteArray()”

FRECallObjectMethod()

AIR 3.0 和更高版本

用法

```
FREResult FRECallObjectMethod(
            FREObject        object,
            const uint8_t*   methodName,
            uint32_t         argc,
            FREObject        argv[],
            FREObject*       result,
            FREObject*       thrownException
);
```

参数

object FREObject，表示对其调用方法的 ActionScript 类对象。

methodName uint8_t 数组。此数组是一个表示所调用方法的名称的字符串。字符串使用 UTF-8 编码，并以 null 字符终止。

argc uint32_t，值是传递给方法的参数数量。此参数是 argv 数组参数的长度。当要调用的方法没有任何参数时，值可以是 0。

argv[] FREObject 数组，每个 FREObject 元素都对应于作为参数传递给所调用方法的 ActionScript 类或基元类型。当要调用的方法没有任何参数时，值可以是 NULL。

result 指向 FREObject 的指针。此 FREObject 变量用于接收所调用方法的返回值。FREObject 变量表示所调用方法返回的 ActionScript 类或基元类型。

thrownException 指向 FREObject 的指针。如果调用此方法会导致运行时引发 ActionScript 异常，则此 FREObject 变量表示 ActionScript Error （或 Error 子类）对象。如果未发生错误，则运行时会将此 FREObject 变量设置为无效。即，thrownException FREObject 变量的 FREGetObjectType() 会返回 FRE_INVALID_OBJECT。如果您不希望处理异常信息，则此指针可以是 NULL。

返回

FREResult，可能的返回值包括 （但不限于）下列值：

FRE_OK 函数已成功。 ActionScript 方法返回值，但未引发异常。

FRE_ACTIONSCRIPT_ERROR 发生 ActionScript 错误。运行时将 thrownException 参数设置为表示 ActionScript Error 类或子类对象。

FRE_ILLEGAL_STATE 扩展上下文已获取一个 ActionScript BitmapData 对象或 ByteArray 对象。在释放 BitmapData 或 ByteArray 对象之前，上下文无法调用此方法。

FRE_INVALID_ARGUMENT method 或 result 参数为 NULL，或者 argc 大于 0 但 argv 为 NULL。

FRE_INVALID_OBJECT FREObject 参数或 argv FREObject 元素无效。

FRE_NO_SUCH_NAME methodName 参数与 object 参数表示的 ActionScript 类对象的方法不匹配。也存在导致返回此值的另一种原因 （不太可能）。尤其是，考虑某个 ActionScript 类具有两个名称相同的方法，但这两个名称位于不同的 ActionScript 命名空间中这一不常见的情况。

FRE_TYPE_MISMATCH FREObject 参数不表示 ActionScript 类对象。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明

调用此函数可调用 ActionScript 类对象的方法。

更多帮助主题

第 18 页的 “FREObject 类型 ”

FREDispatchStatusEventAsync()

AIR 3.0 和更高版本

用法

```
FREResult FREDispatchStatusEventAsync(
            FREContext      ctx,
            const uint8_t*  code,
            const uint8_t*  level
);
```

参数

ctx FREContext，此值为扩展上下文在其上下文初始化函数中接收到的 FREContext 变量。

code 指向 uint8_t 的指针。运行时将 StatusEvent 对象的 code 属性设置为此值。字符串使用 UTF-8 编码，并以 null 字符终止。

level 指向 uint8_t 的指针。此参数是一个 UTF8 编码的字符串，以 null 终止。运行时将 StatusEvent 对象的 level 属性设置为此值。

返回
FREResult，可能的返回值包括 （但不限于）下列值：

FRE_OK 函数已成功。

FRE_INVALID_ARGUMENT ctx、 code 或 level 参数为 NULL。如果 ctx 无效，运行时也会返回此值。

说明
调用此函数可调度 ActionScript StatusEvent 事件。事件的目标是运行时将其与 ctx 参数指定的上下文关联的 ActionScript ExtensionContext 实例。

通常，此函数调度的事件为异步事件。例如，扩展方法可启动另一线程来执行某项任务。当其他线程中的任务完成后，该线程会调用 FREDispatchStatusEventAsync() 以通知 ActionScript ExtensionContext 实例。

注：FREDispatchStatusEventAsync() 函数是您可从本机实现的任意线程中调用的唯一 C API。

除非其中的一个参数无效，否则 FREDispatchStatusEventAsync() 将返回 FRE_OK。但是，返回 FRE_OK 并不表示已调度了事件。在下列情况下，运行时不会调度事件：

- 运行时已释放 ExtensionContext 实例。
- 运行时正在释放 ExtensionContext 实例。
- ExtensionContext 实例没有任何引用。应由运行时垃圾回收器来释放该实例。

将 code 和 level 参数设置为任何以 null 终止且采用 UTF8 编码的字符串值。这些值可以为所需的任何值，但必须将其与扩展的 ActionScript 端保持一致。

示例

更多帮助主题
第 71 页的 “FREContext”

FREGetArrayElementAt()

AIR 3.0 和更高版本

用法

```
FREResult FREGetArrayElementAt (
            FREObject    arrayOrVector,
            uint32_t     index,
            FREObject*   value
);
```

参数
arrayOrVector FREObject，表示一个 ActionScript Array 或 Vector 类对象。

index uint32_t，包含要获取的 Array 或 Vector 元素的索引。 Array 或 Vector 对象的第一个元素的索引为 0。

value 指向 FREObject 的指针。此方法会设置此参数所指向的 FREObject 变量。该方法将 FREObject 变量设置为对应于位于所请求索引处的 Array 或 Vector 元素。

返回
FREResult，可能的返回值包括 （但不限于）下列值：

FRE_OK 函数已成功。 value 参数被设置为所请求的 Array 或 Vector 元素。

FRE_ILLEGAL_STATE 扩展上下文已获取一个 ActionScript BitmapData 对象或 ByteArray 对象。在释放 BitmapData 或 ByteArray 对象之前，上下文无法调用此方法。

FRE_INVALID_ARGUMENT arrayOrVector 参数对应于一个 ActionScript Vector 对象，但 index 大于最后一个元素的索引。如果 value 参数为 NULL，也会返回此值。

FRE_INVALID_OBJECT arrayOrVector FREObject 参数无效。

FRE_TYPE_MISMATCH arrayOrVector FREObject 参数不表示 ActionScript Array 或 Vector 类对象。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明
调用此函数可获取位于 ActionScript Array 或 Vector 类对象的指定索引处的 ActionScript 类对象或基元值。FREObject 参数 arrayOrVector 表示 Array 或 Vector 对象。运行时会设置 value 参数所指向的 FREObject 变量。它将 FREObject 变量设置为对应于相应的 Array 或 Vector 元素。

如果某个 ActionScript Array 对象在请求的索引位置没有值，运行时会将 FREObject value 参数设置为无效，但返回 FRE_OK。

更多帮助主题
第 98 页的 “FRESetArrayElementAt()”
第 99 页的 “FRESetArrayLength()”

FREGetArrayLength()

AIR 3.0 和更高版本

用法

```
FREResult FREGetArrayLength (
            FREObject    arrayOrVector,
            uint32_t*    length
);
```

参数
arrayOrVector FREObject，表示一个 ActionScript Array 或 Vector 类对象。

length 指向 uint32_t 的指针。此方法会使用 Array 或 Vector 类对象的长度设置此参数所指向的 uint32_t 变量。

返回
FREResult，可能的返回值包括 （但不限于）下列值：

FRE_OK 函数已成功。此方法会设置 length 参数指向的 uint32_t 变量。它会将变量设置为 Array 或 Vector 对象的长度。

FRE_ILLEGAL_STATE 扩展上下文已获取一个 ActionScript BitmapData 对象或 ByteArray 对象。在释放 BitmapData 或 ByteArray 对象之前，上下文无法调用此方法。

FRE_INVALID_ARGUMENT length 参数为 NULL。

FRE_INVALID_OBJECT arrayOrVector FREObject 参数无效。

FRE_TYPE_MISMATCH arrayOrVector FREObject 参数不表示 ActionScript Array 或 Vector 类对象。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明
调用此函数可获取 Array 或 Vector 类对象的长度。FREObject 参数 arrayOrVector 表示 Array 或 Vector 对象。运行时会返回 length 参数指向的 uint32_t 变量中的长度。

更多帮助主题
第 99 页的 “FRESetArrayLength()”
第 98 页的 “FRESetArrayElementAt()”

FREGetContextActionScriptData()

AIR 3.0 和更高版本

用法
```
FREResult FREGetContextActionScriptData( FREContext ctx, FREObject *actionScriptData);
```

参数
ctx FREContext 变量。运行时将此值传递给 FREContextInitializer()。请参阅 第 77 页的 “FREContextInitializer()”。

actionScriptData 指向 FREObject 变量的指针。当 FREGetContextActionScriptData() 成功时，运行时会设置此参数。值对应于先前随 FRESetContextActionScriptData() 保存的 ActionScript 对象。

返回
FREResult，可能的返回值包括 （但不限于）下列值:

FRE_OK 函数已成功。 actionScriptData 参数对应于先前随 FRESetContextActionScriptData() 保存的 ActionScript 对象。

FRE_INVALID_ARGUMENT actionScriptData 参数为 NULL。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明
调用此函数可获取扩展上下文的 ActionScript 数据。

更多帮助主题
第 99 页的 “FRESetContextActionScriptData()”
第 15 页的 “ 上下文特定数据 ”

FREGetContextNativeData()

AIR 3.0 和更高版本

用法
```
FREResult FREGetContextNativeData( FREContext ctx, void** nativeData );
```

参数
ctx FREContext 变量。运行时将此值传递给 FREContextInitializer()。请参阅 第 77 页的 “FREContextInitializer()”。

nativeData 指向指向本机数据的指针的指针。

返回
FREResult，可能的返回值包括 （但不限于）下列值：

FRE_OK 函数已成功。 nativeData 参数被设置为指向上下文的本机数据。

FRE_INVALID_ARGUMENT nativeData 参数为 NULL。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明
调用此函数可获取扩展上下文的本机数据。

更多帮助主题
第 100 页的 “FRESetContextNativeData()”
第 15 页的 “ 上下文特定数据 ”

FREGetObjectAsBool()

AIR 3.0 和更高版本

用法
```
FREResult FREGetObjectAsBool  ( FREObject object, unint32_t *value );
```

参数
object FREObject。

value 指向 uint32_t 的指针。函数会将此值设置为对应于 FREObject 变量表示的 Boolean ActionScript 变量的值。对应于 true 的非零值。对应于 false 的零值。

返回
FREResult，可能的返回值包括 （但不限于）下列值：

FRE_OK 函数已成功并且正确设置了 value 参数。

FRE_TYPE_MISMATCH FREObject 参数不包含 Boolean ActionScript 值。

FRE_INVALID_OBJECT FREObject 参数无效。

FRE_INVALID_ARGUMENT 值参数为 NULL。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明
调用此函数可将 C uint32_t 变量的值设置为 ActionScript Boolean 变量的值。

更多帮助主题
第 94 页的 “FRENewObjectFromBool()”
第 18 页的 “FREObject 类型 ”

FREGetObjectAsDouble()

AIR 3.0 和更高版本

用法

```
FREResult FREGetObjectAsDouble ( FREObject object, double *value );
```

参数

object FREObject。

value 指向双精度值的指针。函数会将此值设置为对应于 FREObject 变量表示的 Boolean、int 或 Number ActionScript 变量。

返回

FREResult，可能的返回值包括 （但不限于）下列值:

FRE_OK 函数已成功并且正确设置了 value 参数。

FRE_TYPE_MISMATCH FREObject 参数不包含 Boolean、 int 或 Number ActionScript 值。

FRE_INVALID_OBJECT FREObject 参数无效。

FRE_INVALID_ARGUMENT 值参数为 NULL。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明

调用此函数可将 C 双精度变量的值设置为 ActionScript Boolean、 int 或 Number 变量的值。

更多帮助主题

第 94 页的 “FRENewObjectFromDouble()”

第 18 页的 “FREObject 类型 ”

FREGetObjectAsInt32()

AIR 3.0 和更高版本

用法

```
FREResult FREGetObjectAsInt32 ( FREObject object, int32_t *value );
```

参数

object FREObject。

value 指向 int32_t 的指针。函数会将此值设置为对应于 FREObject 变量表示的 Boolean 或 int ActionScript 变量的值。

返回

FREResult，可能的返回值包括 （但不限于）下列值:

FRE_OK 函数已成功并且正确设置了 value 参数。

FRE_TYPE_MISMATCH FREObject 参数不包含 Boolean 或 int ActionScript 值。

FRE_INVALID_OBJECT FREObject 参数无效。

FRE_INVALID_ARGUMENT 值参数为 NULL。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明
调用此函数可将 C int32_t 变量的值设置为 int 或 Boolean ActionScript 变量的值。

更多帮助主题
第 95 页的 “FRENewObjectFromInt32()”
第 18 页的 “FREObject 类型 ”

FREGetObjectAsUint32()

AIR 3.0 和更高版本

用法
```
FREResult FREGetObjectAsUint32 ( FREObject object, uint32_t *value );
```

参数
object FREObject。
value 指向 uint32_t 的指针。函数会将此值设置为对应于 FREObject 变量表示的 Boolean 或 int ActionScript 变量的值。

返回
FREResult，可能的返回值包括 （但不限于）下列值:
FRE_OK 函数已成功并且正确设置了 value 参数。
FRE_TYPE_MISMATCH FREObject 参数不包含 Boolean 或 int ActionScript 值。int ActionScript 值为负也会导致返回此值。
FRE_INVALID_OBJECT FREObject 参数无效。
FRE_INVALID_ARGUMENT 值参数为 NULL。
FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明
调用此函数可将 C uint32_t 变量的值设置为 ActionScript Boolean 或 int 变量的值。

更多帮助主题
第 95 页的 “FRENewObjectFromUint32()”
第 18 页的 “FREObject 类型 ”

FREGetObjectAsUTF8()

AIR 3.0 和更高版本

用法
```
FREResult FREGetObjectAsUTF8(FREObject object, uint32_t* length, const uint8_t** value);
```

参数
object FREObject。

length 指向 uint32_t 的指针。length 的值为 value 数组中的字节数。长度包括 null 终止符。长度对应于 FREObject 变量表示的 String ActionScript 变量的长度。

value 指向 uint8_t 数组的指针。函数会使用 FREObject 变量表示的 String ActionScript 变量的字符填充数组。字符串采用 UTF-8 编码，并以 null 字符终止。

返回
FREResult，可能的返回值包括 （但不限于）下列值：

FRE_OK 函数已成功且正确设置了 value 和 length 参数。

FRE_TYPE_MISMATCH FREObject 参数不包含 String ActionScript 值。

FRE_INVALID_OBJECT FREObject 参数无效。

FRE_INVALID_ARGUMENT value 或 length 参数为 NULL。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明
调用此函数可将 uint8_t 数组的值设置为 ActionScript String 对象的字符串值。

考虑下面与 value 参数中返回的字符串有关的情况：

- 您无法更改字符串。
- 仅当运行时调用的本机扩展函数返回后，字符串才有效。
- 如果调用任何其他 C API 函数，则字符串变为无效。

因此，如果稍后要操作或访问字符串，请立即将其复制到您自己的数组中。

更多帮助主题
第 96 页的 “FRENewObjectFromUTF8()”
第 18 页的 “FREObject 类型 ”

FREGetObjectProperty()

AIR 3.0 和更高版本

用法

```
FREResult FREGetObjectProperty (
                FREObject        object,
                const uint8_t*   propertyName,
                FREObject*       propertyValue,
                FREObject*       thrownException
);
```

参数
object FREObject，表示从其中提取属性值的 ActionScript 类对象。

propertyName uint8_t 数组。此数组包含一个表示属性名称的字符串。字符串使用 UTF-8 编码，并以 null 字符终止。

propertyValue 指向 FREObject 的指针。此方法会将此 FREObject 参数设置为表示作为所请求属性的 ActionScript 对象。

thrownException 指向 FREObject 的指针。如果调用此方法会导致运行时引发 ActionScript 异常，则此 FREObject 变量表示 ActionScript Error （或 Error 子类）对象。如果未发生错误，则运行时会将此 FREObject 变量设置为无效。即，thrownException FREObject 变量的 FREGetObjectType() 会返回 FRE_INVALID_OBJECT。如果您不希望处理异常信息，则此指针可以是 NULL。

返回
FREResult，可能的返回值包括 （但不限于）下列值:

FRE_OK 函数已成功并且正确设置了 propertyValue 参数。

FRE_ACTIONSCRIPT_ERROR 发生 ActionScript 错误。运行时将 thrownException 参数设置为表示 ActionScript Error 类或子类对象。

FRE_ILLEGAL_STATE 扩展上下文已获取一个 ActionScript BitmapData 对象或 ByteArray 对象。在释放 BitmapData 或 ByteArray 对象之前，上下文无法调用此方法。

FRE_INVALID_ARGUMENT propertyName 或 propertyValue 参数为 NULL。

FRE_INVALID_OBJECT FREObject 参数无效。

FRE_NO_SUCH_NAME propertyName 参数与 object 参数表示的 ActionScript 类对象的属性不匹配。也存在导致返回此值的另一种原因 （不太可能）。尤其是，考虑某个 ActionScript 类具有两个名称相同的属性，但这两个名称位于不同的 ActionScript 命名空间中这一不常见的情况。

FRE_TYPE_MISMATCH FREObject 参数不表示 ActionScript 类对象。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明
调用此函数可将 FREObject 变量设置为对应于 object 参数指定的 ActionScript 类对象的公共属性的数据。

更多帮助主题
第 101 页的 “FRESetObjectProperty()”
第 18 页的 “FREObject 类型 ”

FREGetObjectType()

AIR 3.0 和更高版本

用法

```
FREResult FREGetObjectType( FREObject object, FREObjectType *objectType );
```

参数
object FREObject 变量。

objectType 指向 FREObjectType 变量的指针。FREGetObjectType() 会将此变量设置为其中一个 FREObjectType 枚举值。

返回
FREResult，可能的返回值包括 （但不限于）下列值:

FRE_OK 函数已成功并且正确设置了 objectType 参数。

FRE_INVALID_ARGUMENT objectType 参数为 NULL。

FRE_INVALID_OBJECT FREObject 参数无效。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明
调用此函数可获取最能准确描述 FREObject 变量的相应 ActionScript 类对象或基元类型的 FREObjectType 枚举值。

更多帮助主题
第 74 页的 “FREObjectType”
第 18 页的 “FREObject 类型 ”

FREInvalidateBitmapDataRect()

AIR 3.0 和更高版本

用法

```
FREResult FREInvalidateBitmapDataRect(
            FREObject object,
            uint32_t x,
            uint32_t y,
            uint32_t width,
            uint32_t height
);
```

参数
object FREObject，表示先前获取的 ActionScript BitmapData 类对象。

x uint32_t，此值为 x 坐标 （以像素为单位）。该值表示相对于位图的左上角的坐标。它与 y 参数一起表示要失效的矩形的左上角。

y uint32_t，此值为 y 坐标 （以像素为单位）。该值表示相对于位图的左上角的坐标。它与 x 参数一起表示要失效的矩形的左上角。

width uint32_t，此值是要失效的矩形的宽度，以像素为单位。

height uint32_t，此值是要失效的矩形的高度，以像素为单位。

返回
FREResult，可能的返回值包括 （但不限于）下列值:

FRE_OK 函数已成功。指定的矩形已失效。

FRE_ILLEGAL_STATE 扩展上下文未获取 ActionScript BitmapData 对象。

FRE_INVALID_OBJECT FREObject object 参数无效。

FRE_TYPE_MISMATCH FREObject object 参数不表示 ActionScript BitmapData 类对象。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明
调用此函数可使 ActionScript BitmapData 类对象矩形失效。在调用此函数之前，调用 FREAcquireBitmapData() 或 FREAcquireBitmapData2()。在完成位图操作和失效操作后，调用 FREReleaseBitmapData()。

使 BitmapData 对象矩形失效表示运行时将需要重绘矩形。

更多帮助主题

第 80 页的 “FREAcquireBitmapData()”

第 81 页的 “FREAcquireBitmapData2()”

第 97 页的 “FREReleaseBitmapData()”

FRENewObject()

AIR 3.0 和更高版本

用法

```
FREResult FRENewObject(
            const uint8_t*   className,
            uint32_t         argc,
            FREObject        argv[],
            FREObject*       object,
            FREObject*       thrownException
);
```

参数

className uint8_t 数组。一个作为要创建一个对象的 ActionScript 类的名称的字符串。字符串使用 UTF-8 编码，并以 null 字符终止。

argc uint32_t，传递给 ActionScript 类的构造函数的参数的数量。此参数是 argv 数组参数的长度。当构造函数没有任何参数时，值可以是 0。

argv[] FREObject 数组，每个 FREObject 元素都对应于作为参数传递给构造函数的 ActionScript 类或基元类型。当构造函数没有任何参数时，值可以是 NULL。

object 指向 FREObject 的指针。当此方法成功返回时，此 FREObject 变量表示新的 ActionScript 类对象。

thrownException 指向 FREObject 的指针。如果调用此方法会导致运行时引发 ActionScript 异常，则此 FREObject 变量表示 ActionScript Error （或 Error 子类）对象。如果未发生错误，则运行时会将此 FREObject 变量设置为无效。即，thrownException FREObject 变量的 FREGetObjectType() 会返回 FRE_INVALID_OBJECT。如果您不希望处理异常信息，则此指针可以是 NULL。

返回

FREResult，可能的返回值包括 （但不限于）下列值：

FRE_OK 函数已成功。 object 参数表示新的 ActionScript 类对象。

FRE_ACTIONSCRIPT_ERROR 发生 ActionScript 错误。运行时将 thrownException 参数设置为表示 ActionScript Error 类或子类对象。

FRE_ILLEGAL_STATE 扩展上下文已获取一个 ActionScript BitmapData 对象或 ByteArray 对象。在释放 BitmapData 或 ByteArray 对象之前，上下文无法调用此方法。

FRE_INVALID_ARGUMENT className 或 object 参数为 NULL，或者 argc 大于 0 但 argv 为 NULL 或为空。

FRE_NO_SUCH_NAME className 参数与 ActionScript 类名称不匹配。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明

调用此函数可创建 ActionScript 类的对象。运行时调用的构造函数取决于您在 argv 数组中传递的参数。

更多帮助主题

第 18 页的 “FREObject 类型 ”

FRENewObjectFromBool()

AIR 3.0 和更高版本

用法

```
FREResult FRENewObjectFromBool ( uint32_t value, FREObject* object);
```

参数

value 新 ActionScript Boolean 实例的 uint32_t 值。

object 指向 FREObject 的指针，指向表示 Boolean ActionScript 变量的数据。对应于 true 的非零值。对应于 false 的零值。

返回

FREResult，可能的返回值包括 （但不限于）下列值:

FRE_OK 函数已成功并且正确设置了 object 参数。

FRE_INVALID_ARGUMENT FREObject 参数为 NULL。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明

调用此函数可使用 value 参数创建一个 ActionScript Boolean 实例。运行时会将 FREObject 变量设置为对应于新的 ActionScript 实例的数据。

更多帮助主题

第 87 页的 “FREGetObjectAsBool()”

第 18 页的 “FREObject 类型 ”

FRENewObjectFromDouble()

AIR 3.0 和更高版本

用法

```
FREResult FRENewObjectFromDouble ( double value, FREObject* object);
```

参数

value 新 ActionScript Number 实例的双精度值。

object 指向 FREObject 的指针，指向表示 Number ActionScript 变量的数据。

返回

FREResult，可能的返回值包括 （但不限于）下列值:

FRE_OK 函数已成功并且正确设置了 object 参数。

FRE_INVALID_ARGUMENT FREObject 参数为 NULL。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明
调用此函数可使用 value 参数创建一个 ActionScript Number 实例。运行时会将 FREObject 变量设置为对应于新的 ActionScript 实例的数据。

更多帮助主题
第 88 页的 “FREGetObjectAsDouble()”
第 18 页的 “FREObject 类型 ”

FRENewObjectFromInt32()

AIR 3.0 和更高版本

用法
```
FREResult FRENewObjectFromInt32 ( int32_t value, FREObject* object);
```

参数
value int32_t，新 ActionScript int 实例的值。
object 指向 FREObject 的指针，表示 int ActionScript 实例。

返回
FREResult，可能的返回值包括 （但不限于）下列值:
FRE_OK 函数已成功并且正确设置了 object 参数。
FRE_INVALID_ARGUMENT FREObject 参数为 NULL。
FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明
调用此函数可使用 value 参数创建一个 ActionScript int 实例。运行时会将 FREObject 变量设置为对应于新的 ActionScript 实例。

更多帮助主题
第 88 页的 “FREGetObjectAsInt32()”
第 18 页的 “FREObject 类型 ”

FRENewObjectFromUint32()

AIR 3.0 和更高版本

用法
```
FREResult FRENewObjectFromUint32 ( uint32_t value, FREObject* object);
```

参数
value uint32_t，新 ActionScript int 实例的值，值大于或等于 0。
object 指向 FREObject 的指针，表示 int ActionScript 实例。

返回
FREResult，可能的返回值包括 （但不限于）下列值：

FRE_OK 函数已成功并且正确设置了 object 参数。

FRE_INVALID_ARGUMENT FREObject 参数为 NULL。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明
调用此函数可使用 value 参数创建一个 ActionScript int 实例。运行时会将 FREObject 变量设置为对应于新的 ActionScript int 实例。

更多帮助主题
第 89 页的 “FREGetObjectAsUint32()”
第 18 页的 “FREObject 类型 ”

FRENewObjectFromUTF8()

AIR 3.0 和更高版本

用法

```
FREResult FRENewObjectFromUTF8(uint32_t length, const uint8_t* value, FREObject* object);
```

参数
length uint32_t，value 参数中的字符串的长度，包括 null 终止符。

value 一个 uint8_t 元素数组。此数组是新 ActionScript String 对象的值。FREObject 变量表示一个 String ActionScript 变量。字符串使用 UTF-8 编码，并以 null 字符终止。

object 指向 FREObject 的指针，指向表示 String ActionScript 对象的数据。

返回
FREResult，可能的返回值包括 （但不限于）下列值：

FRE_OK 函数已成功并且正确设置了 object 参数。

FRE_INVALID_ARGUMENT object 或 value 参数为 NULL。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明
调用此函数可使用 value 中指定的字符串值创建一个 ActionScript String 对象。此方法会将 FREObject 输出参数 object 设置为对应于新的 ActionScript String 实例。

更多帮助主题
第 89 页的 “FREGetObjectAsUTF8()”
第 18 页的 “FREObject 类型 ”

FREReleaseBitmapData()

AIR 3.0 和更高版本

用法

```
FREResult FREReleaseBitmapData (FREObject object);
```

参数

object FREObject，对应于一个 ActionScript BitmapData 类对象。

返回

FREResult，可能的返回值包括 （但不限于）下列值:

FRE_OK 函数已成功。 ActionScript BitmapData 对象不再可供您操作。

FRE_ILLEGAL_STATE 扩展上下文未获取 ActionScript BitmapData 对象。

FRE_INVALID_OBJECT FREObject object 参数无效。

FRE_TYPE_MISMATCH FREObject object 参数不表示 ActionScript BitmapData 类对象。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明

调用此函数可释放 ActionScript BitmapData 类对象。在调用此函数之前，调用 FREAcquireBitmapData() 或 FREAcquireBitmapData2() 和 FREInvalidateBitmapDataRect()。在调用了此函数之后，您无法再操作位图，但可以再次调用其他本机扩展 C API 函数。

更多帮助主题

第 80 页的 “FREAcquireBitmapData()”

第 81 页的 “FREAcquireBitmapData2()”

第 92 页的 “FREInvalidateBitmapDataRect()”

FREReleaseByteArray()

AIR 3.0 和更高版本

用法

```
FREResult FREReleaseByteArray (FREObject object);
```

参数

object FREObject，对应于一个 ActionScript ByteArray 类对象。

返回

FREResult，可能的返回值包括 （但不限于）下列值:

FRE_OK 函数已成功。 ActionScript ByteArray 对象不再可供您操作。

FRE_ILLEGAL_STATE 扩展上下文未获取 ActionScript ByteArray 对象。

FRE_INVALID_OBJECT FREObject object 参数无效。

FRE_TYPE_MISMATCH FREObject object 参数不对应于 ActionScript ByteArray 对象。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明
调用此函数可释放 ActionScript ByteArray 类对象。在调用此函数之前，调用 FREAcquireByteArray()。在调用了此函数之后，您无法再操作 ByteArray 字节，但可以再次调用其他本机扩展 C API 函数。

更多帮助主题
第 81 页的 “FREAcquireByteArray()”

FRESetArrayElementAt()

AIR 3.0 和更高版本

用法

```
FREResult FRESetArrayElementAt (
            FREObject    arrayOrVector,
            uint32_t     index,
            FREObject    value
);
```

参数
arrayOrVector FREObject，指向表示 ActionScript Array 或 Vector 类对象的数据。

index uint32_t，包含要设置的 Array 或 Vector 元素的索引。 Array 或 Vector 对象的第一个元素的索引为 0。

value FREObject。此方法会将 index 指定的 Array 或 Vector 元素设置为 FREObject value 参数表示的 ActionScript 对象。

返回
FREResult，可能的返回值包括 （但不限于）下列值:

FRE_OK 函数已成功。 Array 或 Vector 元素被设置为 value FREOjbect 参数。

FRE_ILLEGAL_STATE 扩展上下文已获取一个 ActionScript BitmapData 对象或 ByteArray 对象。在释放 BitmapData 或 ByteArray 对象之前，上下文无法调用此方法。

FRE_INVALID_OBJECT arrayOrVector 或 value FREObject 参数无效。

FRE_TYPE_MISMATCH arrayOrVector FREObject 参数不指向表示 ActionScript Array 或 Vector 类对象的数据。此返回值还意味着 arrayOrVector 参数表示一个 Vector 对象，而 value 参数不是该 Vector 对象的正确类型。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明
调用此函数可设置位于 ActionScript Array 或 Vector 类对象的指定索引处的 ActionScript 类对象或基元值。FREObject 参数 arrayOrVector 对应于 Array 或 Vector 对象。 FREObject 参数 value 对应于数组元素值。

更多帮助主题
第 84 页的 “FREGetArrayElementAt()”

第 85 页的 “FREGetArrayLength()”

FRESetArrayLength()

AIR 3.0 和更高版本

用法

```
FREResult FRESetArrayLength (
            FREObject    arrayOrVector,
            uint32_t     length
);
```

参数

arrayOrVector FREObject，表示一个 ActionScript Array 或 Vector 类对象。

length uint32_t，此方法将 Array 或 Vector 类对象的长度设置为此参数的值。

返回

FREResult，可能的返回值包括 （但不限于）下列值：

FRE_OK 函数已成功。运行时已更改了 Array 或 Vector 对象的大小。

FRE_ILLEGAL_STATE 扩展上下文已获取一个 ActionScript BitmapData 对象或 ByteArray 对象。在释放 BitmapData 或 ByteArray 对象之前，上下文无法调用此方法。

FRE_INSUFFICIENT_MEMORY 运行时无法分配足够的内存以更改 Array 或 Vector 对象的大小。

FRE_INVALID_ARGUMENT length 参数大于 2^{32}。

FRE_INVALID_OBJECT arrayOrVector FREObject 参数无效。

FRE_READ_ONLY arrayOrVector FREObject 参数表示具有固定大小的 ActionScript Vector 对象。（其 fixed 属性为 true。）

FRE_TYPE_MISMATCH arrayOrVector FREObject 参数不表示 ActionScript Array 或 Vector 类对象。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明

调用此函数可设置 ActionScript Array 或 Vector 类对象的长度。FREObject 参数 arrayOrVector 对应于 Array 或 Vector 对象。运行时根据 length 参数指定的值更改 Array 或 Vector 对象的大小。

更多帮助主题

第 84 页的 “FREGetArrayElementAt()”

第 85 页的 “FREGetArrayLength()”

FRESetContextActionScriptData()

AIR 3.0 和更高版本

用法

```
FREResult FRESetContextActionScriptData( FREContext ctx, FREObject actionScriptData);
```

参数

ctx FREContext 变量。运行时将此值传递给 FREContextInitializer()。请参阅 第 77 页的 “FREContextInitializer()”。

actionScriptData FREObject 变量。

返回
FREResult，可能的返回值包括 （但不限于）下列值：

FRE_OK 函数已成功。

FRE_INVALID_OBJECT actionScriptData 参数为无效 FREObject 变量。

FRE_INVALID_ARGUMENT 参数无效。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明
调用此函数可设置扩展上下文的 ActionScript 数据。

更多帮助主题
第 86 页的 “FREGetContextActionScriptData()”
第 15 页的 “ 上下文特定数据 ”

FRESetContextNativeData()

AIR 3.0 和更高版本

用法

```
FREResult FRESetContextNativeData( FREContext ctx, void* nativeData );
```

参数
ctx FREContext 变量。运行时将此值传递给 FREContextInitializer()。请参阅 第 77 页的 “FREContextInitializer()”。

nativeData 指向本机数据的指针。

返回
FREResult，可能的返回值包括 （但不限于）下列值：

FRE_OK 函数已成功。

FRE_INVALID_ARGUMENT nativeData 参数为 NULL。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明
调用此函数可设置扩展上下文的本机数据。

更多帮助主题
第 86 页的 “FREGetContextNativeData()”
第 15 页的 “ 上下文特定数据 ”

FRESetObjectProperty()

AIR 3.0 和更高版本

用法

```
FREResult FRESetObjectProperty (
                FREObject        object,
                const uint8_t*   propertyName,
                FREObject        propertyValue,
                FREObject*       thrownException
);
```

参数

object FREObject，表示要设置属性的 ActionScript 类对象。

propertyName uint8_t 数组。此数组包含一个表示要设置的属性名称的字符串。字符串使用 UTF-8 编码，并以 null 字符终止。

propertyValue FREObject，表示要设置的属性值。

thrownException 指向 FREObject 的指针。如果调用此方法会导致运行时引发 ActionScript 异常，则此 FREObject 变量表示 ActionScript Error （或 Error 子类）对象。如果未发生错误，则运行时会将此 FREObject 变量设置为无效。即，thrownException FREObject 变量的 FREGetObjectType() 会返回 FRE_INVALID_OBJECT。如果您不希望处理异常信息，则此指针可以是 NULL。

返回

FREResult，可能的返回值包括 （但不限于）下列值:

FRE_OK 函数已成功并且正确设置了 ActionScript 类对象的属性。

FRE_ACTIONSCRIPT_ERROR 发生 ActionScript 错误。运行时将 thrownException 参数设置为表示 ActionScript Error 类或子类对象。

FRE_ILLEGAL_STATE 扩展上下文已获取一个 ActionScript BitmapData 对象或 ByteArray 对象。在释放 BitmapData 或 ByteArray 对象之前，上下文无法调用此方法。

FRE_INVALID_ARGUMENT propertyName 参数为 NULL。

FRE_INVALID_OBJECT object 或 propertyValue 参数为无效 FREObject 变量。

FRE_NO_SUCH_NAME propertyName 参数与 object 参数表示的 ActionScript 类对象的属性不匹配。也存在导致返回此值的另一种原因 （不太可能）。尤其是，考虑某个 ActionScript 类具有两个名称相同的属性，但这两个名称位于不同的 ActionScript 命名空间中这一不常见的情况。

FRE_READ_ONLY 要设置的属性为只读属性。

FRE_TYPE_MISMATCH FREObject object 参数不表示 ActionScript 类对象。

FRE_WRONG_THREAD 从运行时在其中具有对本机扩展函数的未决调用的线程以外的线程调用了方法。

说明

调用此函数可设置 FREObject 变量表示的 ActionScript 类对象的公共属性的值。通过 propertyName 参数传递要设置的属性名称。通过 propertyValue 参数传递新的属性值。

更多帮助主题

第 90 页的 “FREGetObjectProperty()”

第 18 页的 “FREObject 类型 ”

第 9 章 : Android Java API 参考

有关使用 Android Java API 的本机扩展示例，请参阅 Adobe AIR 的本机扩展。

接口

用于 AIR 的 Java API 本机扩展定义两个接口，这两个接口都必须由所有扩展实现。

FREExtension

包： com.adobe.fre

运行时版本 AIR 3

FREExtension 接口定义由 AIR 运行时使用的接口，以实例化本机扩展中的 Java 代码。

方法

方法	说明
void initialize()	扩展初始化期间，由运行时调用。
FREContext createContext(String contextType)	创建 FREContext 对象。
void dispose()	当扩展遭到破坏时由运行时调用。

所有本机扩展库都必须实现 FREExtension 接口。实现类的完全限定名称必须存在于扩展描述符文件的 <initializer> 元素中。

例如，如果扩展 Java 代码打包在名为 ExampleExtension.jar 的 JAR 文件中，并且实现 FREExtension 的类为 com.example.ExampleExtension，则应使用以下扩展描述符条目：

```
<platform name="Android-ARM">
    <applicationDeployment>
        <nativeLibrary>ExampleExtension.jar</nativeLibrary>
        <initializer>com.example.ExampleExtension</initializer>
    </applicationDeployment>
</platform>
```

在使用 Java 扩展接口时，不需要 <finalizer> 元素。

方法详细信息

createContext

```
FREContext createContext( String contextType )
```

创建 FREContext 对象。

当扩展调用 ActionScript ExtensionContext.createExtensionContext() 方法后， AIR 运行时调用 Java createContext() 方法。运行时使用返回的上下文进行后续方法调用。

此函数通常使用 contextType 参数来选择本机实现中 ActionScript 端可以调用的方法集。每个上下文类型都对应一个不同的方法集。您的扩展可以创建单个上下文、提供相同功能集的多个上下文实例 （但具有特定于实例的状态）或者提供不同功能的多个上下文实例。

参数：

contextType 标识上下文类型的字符串。可根据扩展的需要定义此字符串。上下文类型可表示扩展的 ActionScript 端与本机端之间的任何约定含义。如果您的扩展不使用上下文类型，则此值可以是 Null。此值为 UTF-8 编码的字符串，以 null 字符终止。

返回：

FREContext 扩展上下文对象。

示例：

以下示例返回基于 contextType 参数的上下文对象。如果 contextType 为字符串 “TypeA”，则函数在每次调用时返回一个唯一的 FREContext 实例。对于 contextType 的其他值，该函数仅创建一个 FREContext 实例，并将其存储在私有变量 bContext 中。

```
private FREContext bContext;
public FREContext createContext( String contextType ) {
    FREContext theContext = null;
    if( contextType == "TypeA" )
    {
        theContext = new TypeAContext();
    }
    else
    {
        if( bContext == null ) bContext = new TypeBContext();
        theContext = bContext;
    }
    return theContext;
}
```

dispose

```
void dispose()
```

使用 dispose() 方法，清除由此 FREExtension 实现创建的所有资源。当关联的 ActionScript ExtensionContext 对象被丢弃或符合垃圾回收的条件时， AIR 运行时调用此方法。

initialize

```
void initialize()
```

扩展初始化期间，由 AIR 运行时调用。

类示例

以下示例演示创建单个 FREContext 对象的简单 FREExtension 示例。该示例还展示如何使用 Android Log 类来将信息性消息记录到 Android 系统日志中 （您可以使用 adb logcat 命令查看这些日志）。

```
package com.example;

import android.util.Log;
import com.adobe.fre.FREContext;
import com.adobe.fre.FREExtension;

public class DataExchangeExtension implements FREExtension {

    private static final String EXT_NAME = "DataExchangeExtension";
    private String tag = EXT_NAME + "ExtensionClass";
    private DataExchangeContext context;

    public FREContext createContext(String arg0) {
        Log.i(tag, "Creating context");
        if( context == null) context = new DataExchangeContext();
        return context;
    }

    public void dispose() {
        Log.i(tag, "Disposing extension");
        // nothing to dispose for this example
    }

    public void initialize() {
        Log.i(tag, "Initialize");
        // nothing to initialize for this example
    }
}
```

FREFunction

包：com.adobe.fre

运行时版本 AIR 3

FREFunction 接口定义运行时使用的用于调用本机扩展中定义的 Java 函数的接口。

方法

方法	说明
FREObject call(FREContext ctx, FREObject[] args)	当扩展的 ActionScript 端调用此函数时，由 AIR 运行时调用。

为扩展中的可以由本机扩展库的 ActionScript 部分调用的每个 Java 函数实现 FREFunction 接口。将类添加到由 FREContext 实例的提供函数的 getFunctions() 方法返回的 Java Map 对象。

当执行 ActionScript 部分中的 ExtensionContext 实例的 call() 方法时，运行时调用 call() 方法。

将 FREFunction 添加到上下文的函数映射中时，要指定一个字符串值作为密钥。当从 ActionScript 调用该函数时，使用同一值。

方法详细信息

调用

```
FREObject call( FREContext ctx, FREObject[] args )
```

当从 ActionScript 调用此函数时，由运行时调用。实现或启动由 call() 方法中的 FREFunction 实例提供的功能。

参数:

ctx 表示此扩展上下文的 FREContext 变量。

使用 ctx 参数可执行以下操作:

- 获取和设置与扩展上下文关联的数据。
- 使用 FREContext dispatchStatusEventAsync() 方法将异步事件调度到 ActionScript 端的 ExtensionContext 实例。

args 作为 FREObject 变量的数组传递到函数的参数。数组中的对象是用于调用 java 函数的 ActionScript ExtensionContext call() 方法中的参数。

返回:

FREObject 结果。在扩展的 Java 和 ActionScript 部分间共享的所有对象均封装在一个 FREObject 或其中一个子类中。

类示例

以下实例演示一个函数获取字符串参数并返回一个带有反转字符的新字符串。 call() 函数使用 ctx 参数从通过其调用函数的上下文中获取一个标识符字符串。

```
package com.example;

import com.adobe.fre.FREContext;
import com.adobe.fre.FREFunction;
import com.adobe.fre.FREInvalidObjectException;
import com.adobe.fre.FREObject;
import com.adobe.fre.FRETypeMismatchException;
import com.adobe.fre.FREWrongThreadException;
import android.util.Log;

public class ReverseStringFunction implements FREFunction {
    public static final String NAME = "reverseString";
    private String tag;

    public FREObject call(FREContext arg0, FREObject[] arg1) {
        DataExchangeContext ctx = (DataExchangeContext) arg0;
        tag = ctx.getIdentifier() + "." + NAME;
        Log.i( tag, "Invoked " + NAME );
        FREObject returnValue = null;

        try {
            String value = arg1[0].getAsString();
            value = reverse( value );
            returnValue = FREObject.newObject( value );//Invert the received value

        } catch (IllegalStateException e) {
            Log.d(tag, e.getMessage());
            e.printStackTrace();
        } catch (FRETypeMismatchException e) {
            Log.d(tag, e.getMessage());
```

```
            e.printStackTrace();
        } catch (FREInvalidObjectException e) {
            Log.d(tag, e.getMessage());
            e.printStackTrace();
        } catch (FREWrongThreadException e) {
            Log.d(tag, e.getMessage());
            e.printStackTrace();
        }

        return returnValue;
    }

    private String reverse( String source ) {
        int i, len = source.length();
        StringBuffer dest = new StringBuffer(len);
        for (i = (len - 1); i >= 0; i--)
            dest.append(source.charAt(i));
            return dest.toString();
    }
}
```

类

用于 AIR 的 Java API 本机扩展定义一个抽象类 FREContext，该类可以允许定义扩展上下文。 API 还定义几个实际类（FREObject 及其子类），这些类表示在本机扩展的 Java 与 ActionScript 端共享的对象。扩展必须实现至少一个 FREContext 的实际子类。

FREArray

包： com.adobe.fre

继承 FREObject

运行时版本 AIR 3

FREArray 类表示一个 ActionScript 数组或一个 Vector 对象。

方法

方法	说明
public static FREArray newArray (String classname, int numElements, boolean fixed)	创建一个 ActionScript Vector 数组对象。
public static FREArray newArray (int numElements)	创建一个 ActionScript Array 对象。
public long getLength()	获取数组中的元素数。
public void setLength(long length)	改变数组长度。
public FREObject getObjectAt(long index)	获取指定索引处的对象。
public void setObjectAt(long index, FREObject value)	将指定索引处的对象推入数组。

您可以通过使用 FREArray 类中定义的方法以及 FREObject 类（FREArray 的超类）中定义的方法，使用 FREArray 对象。使用 FREObject getProperty() 和 setProperty() 方法来访问或修改数组和 Vector 类的 ActionScript 定义的属性。使用 callMethod() 来调用 ActionScript 定义的方法。

方法详细信息

newArray

```
public static FREArray newArray (String classname, int numElements, boolean fixed)
```

创建一个 ActionScript Vector 数组对象。

参数：

classname Vector 数组成员的 ActionScript 类的完全限定名称。

numElements 为数组分配的元素的个数。

fixed 如果值为 true，则向量长度不能更改。

返回：

FREArray 与 ActionScript Vector 数组对象关联的 FREArray 对象。

示例：

```
FREArray vector = FREArray.newArray( "flash.geom.Matrix3D", 4, true );
```

newArray

```
public static FREArray newArray (int numElements)
```

创建一个 ActionScript Array 对象。

参数：

numElements 为数组分配的元素的个数。未定义的元素。

返回：

FREArray 与 ActionScript Array 对象关联的 FREArray 对象。

示例：

```
FREArray array = FREArray.newArray( 4 );
```

getLength

```
public long getLength()
```

获取数组中的元素数。

返回：

long 数组的长度。

示例：

```
long length = asArray.getLength();
```

setLength

```
public void setLength( long length )
```

更改此数组的长度。如果新长度短于当前长度，则该数组会被截断。

参数：

length 数组的新长度。

示例：

```
asArray.setLength( 4 );
```

getObjectAt

```
public FREObject getObjectAt( long index )
```

从数组中获取一个元素。

参数：

index 要检索的元素的位置。（从 0 开始）

返回：

FREObject 与数组中的 ActionScript 对象关联的 FREObject 实例。

示例：

```
FREObject element = asArray.getObjectAt( 2 );
```

setObjectAt

```
public void setObjectAt( long index, FREObject value )
```

将指定索引处的对象推入数组。

参数：

index 数组中将对象推入的位置。（从 0 开始）

value 包含要插入的原始值或 ActionScript 对象的 FREObject。

示例：

```
FREObject stringElement = FREObject.newObject("String element value");
FREArray asVector = FREArray.newArray( "String", 1, false );
asVector.setObjectAt( 0, stringElement );
```

FREByteArray

包：com.adobe.fre

继承 FREObject

运行时版本 AIR 3

FREByteArray 类表示一个 ActionScript ByteArray 对象。

方法

方法	说明
public static FREByteArray newByteArray()	创建一个空 ActionScript ByteArray 对象。
public long getLength()	返回字节数组的长度 （字节）。
public ByteBuffer getBytes()	获取作为 Java ByteBuffer 的 ActionScript ByteArray 对象的内容。
public void acquire()	对 ActionScript 对象获取锁定。
public void release()	对 ActionScript 对象释放锁定。

通过调用 getBytes() 访问 ByteArray 对象中的数据。在访问 ActionScript 引用的数组中的字节数据前，必须调用 acquire() 以锁定该对象。在访问数据或修改数据完成后，调用 release() 以释放锁定。

当对数组进行锁定后，可以修改缓冲区中的现有数据，但不能更改数组的大小。若要修改数组大小，请释放锁定并使用 FREObject 超类定义的 setProperty() 方法更改 ActionScript 定义的 length 属性。您可以使用 getProperty()、setProperty() 和 callMethod() 函数访问 ActionScript ByteArray 类定义的所有属性和方法。

方法详细信息

newByteArray

```
public static FREByteArray newByteArray()
```

创建空 ActionScript ByteArray 对象及其关联的 Java FREByteArray 实例。

返回：

FREByteArray 表示 ActionScript ByteArray 的 FREByteArray 对象。

示例：

```
FREBytearray bytearray = FREByteArray.newByteArray();
```

acquire

```
public void acquire()
```

获取对此对象的锁定，以便当您访问数据时，运行时不能修改或丢弃这些数据。锁定时，不能读取或修改该对象的所有 ActionScript 定义的属性。

getLength

```
public long getLength()
```

返回字节数组中字节的数目。调用此方法前必须先调用此对象的 acquire() 函数。

返回：

long 字节数组中的字节的数目。

getBytes

```
public ByteBuffer getBytes()
```

返回数组中的字节数据。调用此方法前必须首先调用 acquire() 锁定该对象。缓冲区仅在锁定后有效。

java.nio.ByteBuffer 字节数组中的数据。

示例：

```
FREByteArray bytearray = FREByteArray.newByteArray();
bytearray.acquire();
ByteBuffer bytes = bytearray.getBytes();
bytes.putFloat( 16.3 );
bytearray.release();
```

release

```
public void release()
```

释放被 acquire() 获取的锁定。只有在释放锁定后才能访问所有 FREByteArray 属性，除了 getBytes() 返回的数据。

FREBitmapData

包： com.adobe.fre

继承 FREObject

运行时版本 AIR 3

FREBitmapData 类表示一个 ActionScript BitmapData 对象。

方法

方法	说明
public static FREBitmapData newBitmapData (int width, int height, boolean transparent, Byte[] fillColor)	创建一个 ActionScript BitmapData 对象。
public int getWidth()	获取位图宽度。
public int getHeight()	获取位图高度。
public boolean hasAlpha()	指示位图是否具有 alpha 通道。
public boolean isPremultiplied()	指示位图颜色是否已经乘以 alpha 值。
public int getLineStride32()	按位图扫描行获取 32 位的值的数量。
ByteBuffer getBits()	获取位图像素。
public void acquire()	对 ActionScript 对象获取锁定。
void invalidateRect(int x, int y, int width, int height)	当需要更新时，在位图内标记矩形。
public void release()	对 ActionScript 对象释放锁定。
AIR 3.1 添加了如下内容：	
public boolean isInvertedY	指明存储图像数据行的顺序。

FREBitmapData 对象的大部分属性仅是可读的。例如，您不能更改位图的宽度或高度。（若要更改这些属性，请创建具有所需的尺寸的新的 FREBitmapData 对象。）但也可以使用 getBits() 方法访问和修改现有位图的像素值。也可以使用 FREObject getProperty() 和 setProperty() 方法访问 BitmapData 对象的 ActionScript 定义的属性，并使用 FREObject callMethod() 函数调用该对象的 ActionScript 方法。

在调用 ActionScript 代码引用的 FREBitmapData 对象上的 getBits() 之前，使用 acquire() 方法锁定位图。这会阻止 AIR 运行时或应用程序在您的函数正在运行时 （可能在另一个线程内运行）修改或丢弃位图对象。修改位图后，调用 invalidateRect() 通知运行时位图已更改并使用 release() 释放锁定。不能在获取锁定后使用 ActionScript 定义的属性和方法。

AIR 运行时将像素数据定义为 32 位值，其中包含三个颜色组成，加采用顺序 ARGB 的 alpha。数据内的每个组成为一个字节。有效的位图数据存储为颜色组成预乘以 alpha 组成。（但也可能接收一个技术上无效的 BitmapData，其中，颜色未进行预乘。例如，具有 Context3D 类的呈现位图可以产生此类无效位图。在这种情况下， isPremultiplied() 方法仍会报告 true。）

方法详细信息

newBitmapData

```
public static FREBitmapData newBitmapData (int width, int height, boolean transparent, Byte[] fillColor)
```

创建一个 FREBitmapData 对象，该对象可以作为一个 BitmapData 实例返回到扩展的 ActionScript 端。

参数：

width 宽度 （以像素为单位）。在 AIR 3+ 中，对位图的宽度和高度没有任何限制 （除了 ActionScript 带符号整数的最大值）。但是，仍会有实际内存限制。位图数据会占用 32 位内存 （每像素）。

height 高度 （以像素为单位）。

transparent 指定此位图是否使用 alpha 通道。此参数对应于创建的 FREBitmapData 的 hasTransparency() 方法和 ActionScript BitmapData 对象的 transparent 属性。

fillColor 32 位，带有要填充新位图的 ARGB 颜色值。如果 transparent 参数为 false，则会忽略颜色的 alpha 组成。

返回：

FREBitmapData 与 ActionScript BitmapData 对象关联的 FREBitmapData 对象。

示例：

```
Byte[] fillColor = {0xf,0xf,0xf,0xf,0xf,0xf,0xf,0xf};
FREBitmapData bitmap = FREBitmapData.newBitmapData( 320, 200, true, fillColor );
```

getWidth

```
public int getWidth()
```

返回位图的宽度，以像素为单位。此值对应于 ActionScript BitmapData 类对象的 width 属性。调用此方法前必须先调用此对象的 acquire() 函数。

返回：

int 位图的跨水平轴的像素的数目。

getHeight

```
public int getHeight()
```

位图的高度 （以像素为单位）。此值对应于 ActionScript BitmapData 类对象的 height 属性。调用此方法前必须先调用此对象的 acquire() 函数。

返回：

int 位图的跨垂直轴的像素的数目。

hasAlpha

```
public boolean hasAlpha()
```

指示位图是否支持每个像素具有不同的透明度。此值对应于 ActionScript BitmapData 类对象的 transparent 属性。如果值为 true，则会使用像素颜色值的 alpha 组成。如果值为 false，则不使用颜色的 alpha 组成。调用此方法前必须先调用此对象的 acquire() 函数。

返回：

boolean 如果位图使用 alpha 通道，则值为 true。

isPremultiplied

```
public boolean isPremultiplied()
```

指示是否将位图像素作为预乘的颜色值进行存储。 True 值表示像素颜色的红、绿和蓝组成预乘 alpha 组成。 ActionScript BitmapData 对象目前始终进行预乘 （但以后会对运行时进行更改，从而可以创建不进行预乘的位图）。有关预乘颜色值的详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考中的 BitmapData.getPixel()。调用此方法前必须先调用此对象的 acquire() 函数。

返回：

boolean 如果像素颜色组成已经乘以 alpha 组成，则值为 true。

getLineStride32

```
public int getLineStride32()
```

指定位图的每个水平行的数据量。将此值与 getBits() 返回的字节数组结合使用以解析图像。例如，字节数组中紧跟位于位置 n 处的像素的位置为 n + getLineStride32()。调用此方法前必须先调用此对象的 acquire() 函数。

返回：

int 图像中每个水平行的数据量。

示例：

以下示例将包含像素颜色的字节数组的位置移至位图中第三行的开始处：

```
Byte[] fillColor = {0xf,0xf,0xf,0xf,0xf,0xf,0xf,0xf};
FREBitmapData bitmap = FREBitmapData.newBitmapData( 320, 200, true, fillColor );
int lineStride = bitmap.getLineStride32();
bitmap.acquire();
ByteBuffer pixels = bitmap.getBits();
pixels.position( lineStride * 3 );
//do something with the image data
bitmap.release();
```

getBits

```
ByteBuffer getBits()
```

返回像素颜色值的数组。在使用此方法前必须首先调用 acquire()，并在完成数据修改后调用 release()。若要对像素数据进行更改，请调用 invalidateRect() 以通知 AIR 运行时位图已更改。否则，不会更新显示的图形。

返回：

java.nio.ByteBuffer 图像数据。

示例：

```
FREBitmapData bitmap = FREBitmapData.newBitmapData( 320, 200, true, fillColor );
bitmap.acquire();
ByteBuffer pixels = bitmap.getBits();
//do something with the image data
bitmap.release();
```

acquire

```
public void acquire()
```

获取对图像的锁定，以便访问该图像时，运行时不能修改或丢弃这些数据。仅需对从扩展的 ActionScript 端传递到函数的位图获取锁定。

锁定后，仅可访问 getBits() 函数返回的数据。访问或试图修改 FREBitmapData 结果的其他属性将导致异常。

invalidateRect

```
void invalidateRect( int x, int y, int width, int height )
```

通知运行时您已更改图像的一个区域。位图数据对象的屏幕显示不会更新，除非您调用此方法。调用此方法前必须先调用此对象的 acquire() 函数。

参数：

x 无效的矩形的左上角。

y 无效的矩形的左上角。

width 无效的矩形的宽度。

height 无效的矩形的高度。

release

```
public void release()
```

释放被 acquire() 获取的锁定。只有在释放锁定后才能访问所有位图属性，除了 getBits() 返回的像素数据。

isInvertedY

```
public boolean isInvertedY()
```

指明存储图像数据行的顺序。如果为 true，则在 getBits() 方法返回的缓存中首先显示图像数据的最后一行。Android 上的图像通常从数据的第一行开始存储，因此在 Android 平台上，此属性通常为 false。

AIR 3.1 中添加了以下内容。

FREContext

包：com.adobe.fre

继承 java.lang.Object

运行时版本 AIR 3

FREContext 类表示 AIR 本机扩展定义的 Java 执行上下文。

方法

方法	说明
public abstract Map<String,FREFunction> getFunctions()	AIR 运行时调用以发现此上下文提供的函数。
public FREObject getActionScriptData()	获取与此 FREContext 关联的 ActionScript ExtensionContext 对象的 actionScriptData 属性中的所有数据。
public void setActionScriptData(FREObject)	设置与此 FREContext 关联的 ActionScript ExtensionContext 对象的 actionScriptData 属性。
public abstract void dispose()	当与此 FREContext 实例关联的 ActionScript ExtensionContext 对象被丢弃时，由 AIR 运行时调用。
public android.app.Activity getActivity()	返回应用程序活动。
public native int getResourceId(String resourceString)	返回 Android 资源的 ID。
public void dispatchStatusEventAsync(String code, String level)	通过与此 FREContext 实例关联的 ExtensionContext 对象，将 StatusEvent 调度到应用程序。

您的扩展必须定义 FREContext 类的一个或多个实际子类。当扩展的 ActionScript 端调用 ExtensionContext.createExtensionContext() 方法时， FREExtension 类的实现中的 createContext() 方法创建这些类的实例。

您可以采用多种方法组织扩展提供的上下文。例如，您可以创建在应用程序的生存期存在的上下文类的单个实例。或者，可以创建紧密绑定到具有有限生命期的一组本机对象的上下文类。然后为每组对象创建单独的上下文实例，并随关联的本机对象丢弃上下文实例。

注：如果已经调用了任何 FREByteArray 或 FREBitmapData 对象的 acquire() 方法，则不能调用由任何对象的 FREObject 类定义的方法。

方法详细信息

getFunctions

```
public abstract Map<String,FREFunction> getFunctions()
```

实现此函数以返回包含上下文公开的所有函数的 Java Map 实例。调用函数时，映射密钥值用于查找 FREFunction 对象。

返回：

Map<String, FREFunction> 包含字符串密钥值的映射对象和相应的 FREFunction 对象。当扩展的 ActionScript 部分调用 ExtensionContext 类的 call() 方法时， AIR 运行时使用此 Map 对象查找要调用的函数。

示例：

以下 getFunctions() 示例创建包含三个假设函数的 Java HashMap。通过将字符串 “negate”、“invert” 或 “reverse” 传递到 ExtensionContext call() 方法，可以调用扩展的 ActionScript 代码中的这些函数。

```
@Override
    public Map<String, FREFunction> getFunctions() {
        Map<String, FREFunction> functionMap = new HashMap<String, FREFunction>();
        functionMap.put( "negate", new NegateFunction() );
        functionMap.put( "invert", new InvertFunction() );
        functionMap.put( "reverse", new ReverseFunction() );
        return functionMap;
}
```

getActionScriptData

```
public FREObject getActionScriptData()
```

获取与此上下文关联的 ActionScript 数据对象。 Java 和 ActionScript 代码均可读取或修改此数据对象。

返回：

FREObject FREObject 表示数据对象。

示例：

```
FREObject data = context.getActionScriptData();
```

setActionScriptData

public void setActionScriptData(FREObject object)

设置与此上下文关联的 ActionScript 数据对象。 Java 和 ActionScript 代码均可读取或修改此数据对象。

参数：

value 包含原始值的 FREObject 或 ActionScript 对象。

示例：

```
FREObject data = FREObject.newObject( "A string" );
context.setActionScriptData( data );
```

getActivity

```
public android.app.Activity getActivity()
```

获取对应用程序活动对象的引用。 Android SDK 中的许多 API 都需要此类引用。

返回：

android.app.Activity AIR 应用程序的活动对象。

示例：

```
Activity activity = context.getActivity();
```

getResourceID

```
public native int getResourceId( String resourceString )
```

查找本机 Android 资源的资源 ID。

在 resourceString 参数中指定标识资源的字符串。遵循 Android 上资源访问的命名约定。例如，在 Android 本机代码中要访问的资源的形式为：

```
R.string.my_string
```

具有以下 resourceString：

```
"string.my_string"
```

如果无法找到由 resourceString 指定的资源，则此方法将引发 Resources.NotFoundException。

参数：

resourceString 标识 Android 资源的字符串。

返回：

int Android 资源的 ID。

示例：

```
int myResourceID = context.getResourceID( "string.my_string" );
```

dispatchStatusEventAsync

```
public void dispatchStatusEventAsync( String code, String level )
```

调用此函数可调度 ActionScript StatusEvent 事件。事件的目标是运行时关联此 FREContext 对象的 ActionScript ExtensionContext 实例。

通常，此函数调度的事件为异步事件。例如，扩展方法可启动另一线程来执行某项任务。当其他线程中的任务完成后，该线程会调用 dispatchStatusEventAsync() 以通知 ActionScript ExtensionContext 实例。

注：dispatchStatusEventAsync() 函数是您可从本机实现的任意线程中调用的唯一 Java API。

在下列情况下，运行时不会调度事件：

- 运行时已释放 ExtensionContext 实例。
- 运行时正在释放 ExtensionContext 实例。
- ExtensionContext 实例没有任何引用。应由运行时垃圾回收器来释放该实例。

将 code 和 level 参数设置为任意字符串值。这些值可以为所需的任何值，但必须将其与扩展的 ActionScript 端保持一致。

参数：

code 要报告的状态的描述。

level 消息的扩展定义的类别。

示例：

```
context.dispatchStatusEventAsync( "Processing finished", "progress" );
```

dispose

```
public abstract void dispose()
```

当关联的 ActionScript ExtensionContext 对象被丢弃且此 FREContext 对象没有任何其他引用时，由 AIR 运行时调用。您可以实现此方法来清除上下文资源。

类示例

下面的示例创建 FREContext 的一个子类以返回上下文信息：

```
package com.example;
import java.util.HashMap;
import java.util.Map;
import android.util.Log;
import com.adobe.fre.FREContext;
import com.adobe.fre.FREFunction;

public class DataExchangeContext extends FREContext {
    private static final String CTX_NAME = "DataExchangeContext";
    private String tag;

    public DataExchangeContext( String extensionName ) {
        tag = extensionName + "." + CTX_NAME;
        Log.i(tag, "Creating context");
    }
    @Override
    public void dispose() {
        Log.i(tag, "Dispose context");
    }
    @Override
    public Map<String, FREFunction> getFunctions() {
     Log.i(tag, "Creating function Map");
     Map<String, FREFunction> functionMap = new HashMap<String, FREFunction>();
     functionMap.put( NegateBooleanFunction.NAME, new NegateBooleanFunction() );
     functionMap.put( InvertBitmapDataFunction.NAME, new InvertBitmapDataFunction() );
     functionMap.put( InvertByteArrayFunction.NAME, new InvertByteArrayFunction() );
     functionMap.put( InvertNumberFunction.NAME, new InvertNumberFunction() );
     functionMap.put( ReverseArrayFunction.NAME, new ReverseArrayFunction() );
     functionMap.put( ReverseStringFunction.NAME, new ReverseStringFunction() );
     functionMap.put( ReverseVectorFunction.NAME, new ReverseVectorFunction() );

    return functionMap;
    }

    public String getIdentifier()
    {
        return tag;
    }
}
```

FREObject

包： com.adobe.fre

继承 java.lang.Object

子类 FREArray、 FREBitmapData、 FREByteArray

运行时版本 AIR 3

FREObject 类表示 Java 代码的 ActionScript 对象。

方法

方法	说明
public static FREObject newObject(int value)	创建包含 32 位带符号整数值的 FREObject。
public static FREObject newObject(double value)	创建包含 Java double 值的 FREObject，对应于 ActionScript Number 类型。
public static FREObject newObject(boolean value)	创建包含 boolean 值的 FREObject。
public static FREObject newObject(String value)	创建包含 String 值的 FREObject。
public int getAsInt()	访问 FREObject 中作为 Java int 值的数据。
public double getAsDouble()	访问 FREObject 中作为 Java double 值的数据。
public Boolean getAsBool()	访问 FREObject 中作为 Java Boolean 值的数据。
public String getAsString()	访问 FREObject 中作为 Java String 值的数据。
public static native FREObject newObject(String className, FREObject[] constructorArgs)	创建引用 ActionScript 类的新实例的 FREObject。
public FREObject getProperty(String propertyName)	获取 ActionScript 属性的值。
public void setProperty(String propertyName, FREObject propertyValue)	设置 ActionScript 属性的值。
public FREObject callMethod(String methodName, FREObject[] methodArgs)	调用 ActionScript 方法。

使用 FREObjects 在扩展中的 Java 和 ActionScript 代码间共享数据。运行时将 FREObject 变量与相应的 ActionScript 对象关联。使用 getProperty()、setProperty() 和 callMethod() 函数可以访问与 FREObject 关联的 ActionScript 对象的属性和方法。

FREObject 是一般目的对象。它可以表示原始值，也可以表示类类型。如果访问对象采用与对象中数据不兼容的方式，则会引发 FRETypeMismatchException。

FREObjects 可用于表示所有 ActionScript 对象。此外，子类 FREArray、FREBitmapData 和 FREByteArray 提供了其他方法来处理特定类型的数据。

通过从 FREFunction 实例的 call() 方法返回 FREObject，可以将数据从 Java 传递到 ActionScript 代码。ActionScript 代码将数据作为 ActionScript 对象接收，从而可以将其转换为相应的原始或类类型。还可以设置属性并调用对其具有 FREObject 引用的 ActionScript 对象的方法。当设置 ActionScript 方法的属性或参数时，请使用 FREObject。

您可以将数据从 ActionScript 作为对 FREFunction 实例的 call() 方法的参数传递到 Java 代码。当调用 call() 方法时，参数将封装到 FREObject 实例中。如果参数是对 ActionScript 对象的引用，则 Java 代码可以读取属性值并调用该对象的方法。这些属性和函数返回值将作为 FREObjects 提供给您的 Java 代码。您可以使用 FREObject 方法访问作为 Java 类型的数据，并在适当时将对象向下转换为 FREObject 子类，例如 FREBitmapData。

方法详细信息

newObject(int)

```
public static FREObject newObject( int value )
```

创建包含 32 位带符号整数值的 FREObject。

参数:

value 带符号整数。

返回:

FREObject FREObject。

示例:

```
FREObject value = FREObject.newObject( 4 );
```

newObject(double)

```
public static FREObject newObject( double value )
```

创建包含 Java double 值的 FREObject，对应于 ActionScript Number 类型。

参数:

value double 值。

返回:

FREObject FREObject。

示例:

```
FREObject value = FREObject.newObject( 3.14156d );
```

newObject(boolean)

```
public static FREObject newObject( boolean value )
```

创建包含 boolean 值的 FREObject。

参数:

value true 或 false。

返回:

FREObject FREObject。

示例:

```
FREObject value = FREObject.newObject( true );
```

newObject(String)

```
public static FREObject newObject( String value )
```

创建包含 String 值的 FREObject。

参数:

value String 值。

返回:

FREObject FREObject。

示例:

```
FREObject value = FREObject.newObject( "A string value" );
```

getAsInt

```
public int getAsInt()
```

访问 FREObject 中作为 Java int 值的数据。

返回：

int 整数值。

示例：

```
int value = FREObject.getAsInt();
```

getAsDouble

```
public double getAsDouble()
```

访问 FREObject 中作为 Java double 值的数据。

返回：

double double 值。

示例：

```
double value = FREObject.getAsInt();
```

getAsBool

```
public Boolean getAsBool()
```

访问 FREObject 中作为 Java Boolean 值的数据。

返回：

boolean true 或 false

示例：

```
boolean value = FREObject.getAsInt();
```

getAsString

```
public String getAsString()
```

访问 FREObject 中作为 Java String 值的数据。

返回：

String String 值。

示例：

```
String value = FREObject.getAsInt();
```

newObject(String, FREObject[])

```
public static native FREObject newObject( String className, FREObject[] constructorArgs )
```

创建引用 ActionScript 类的新实例的 FREObject。

参数：

className 完全限定 ActionScript 类名。

constructorArgs 作为 FREObjects 的数组传递到 ActionScript 类构造函数的参数。如果类构造函数没有任何参数，则设置为 null。

返回：

FREObject 表示 ActionScript 类的新实例的 FREObject。

示例：

```
FREObject matrix = FREObject.newObject( "flash.geom.Matrix", null );
```

getProperty

```
public FREObject getProperty( String propertyName )
```

获取 ActionScript 属性的值。

参数:

propertyName 要访问的属性的名称。

返回:

FREObject 作为 FREObject 的指定属性的值。

示例:

```
FREObject isDir = fileobject.getProperty( "isDirectory" );
```

setProperty

```
public void setProperty( String propertyName, FREObject propertyValue )
```

设置 ActionScript 属性的值。

参数:

propertyName 要设置的属性的名称。

propertyValue 包含新属性值的 FREObject。

示例:

```
fileobject.setProperty( "url", FREObject.newObject( "app://file.txt" ) );
```

callMethod

```
public FREObject callMethod( String methodName, FREObject[] methodArgs )
```

调用 ActionScript 方法。

参数:

methodName 要调用的方法的名称。

methodArgs 包含该方法的参数的 FREObjects 数组，采用方法参数的声明顺序。

返回:

FREObject 方法结果。如果 ActionScript 方法返回一个 Array、 Vector 或 BitmapData 对象，则可以将结果转换为 FREObject 的对应的子类。

示例:

```
FREObject[] args = new FREObject[1]
args[0] = FREObject.newObject( "assets/image.jpg" );
FREObject imageFile = directoryobject.callMethod( "resolvePath", args );
```