ADOBE® AIR® 用ネイティブ拡張の開発

法律上の注意

法律上の注意については、http://help.adobe.com/ja_JP/legalnotices/index.html を参照してください。

目次

第 1 章：Adobe AIR 用ネイティブ拡張の概要

ネイティブ拡張について 1
ネイティブ拡張のアーキテクチャ 3
ネイティブ拡張を作成するためのタスクの概要 6

第 2 章：ActionScript 側のコーディング

パブリックインターフェイスの宣言 7
ネイティブ拡張のサポートの確認 8
ExtensionContext インスタンスの作成 8
ネイティブ関数の呼び出し 9
イベントの監視 10
ExtensionContext インスタンスの破棄 12
ネイティブ拡張のディレクトリへのアクセス 12
ネイティブ拡張からの呼び出し元アプリケーションの特定 13
ネイティブ拡張の下位互換性 13

第 3 章：C を使用したネイティブ側のコーディング

拡張の初期化 15
拡張コンテキストの初期化 17
コンテキスト固有データ 18
拡張コンテキストのファイナライズ処理 18
拡張のファイナライズ処理 19
拡張関数 19
非同期イベントの送出 21
FREObject 型 21
ActionScript プリミティブ型およびオブジェクトの操作 23
スレッドとネイティブ拡張 28

第 4 章：Java を使用したネイティブ側のコーディング

FREExtension インターフェイスの実装 29
FREContext クラスの拡張 31
FREFunction インターフェイスの実装 32
非同期イベントの送出 33
ActionScript オブジェクトに対するアクセス 34
ActionScript プリミティブ型およびオブジェクトの操作 35
スレッドとネイティブ拡張 37

第 5 章：ネイティブ拡張のパッケージ化

ネイティブ拡張の ActionScript ライブラリの構築 39
ネイティブ拡張の署名済み証明書の作成 40
拡張記述ファイルの作成 41
ネイティブライブラリの構築 42
ネイティブ拡張パッケージの作成 46
ネイティブ拡張パッケージにリソースを含める 51

第 6 章：AIR for TV 向けネイティブ拡張の構築およびインストール

AIR for TV 拡張の開発におけるタスクの概要 55
AIR for TV 拡張の例 56
デバイスバンドル拡張とスタブ拡張 61
拡張のサポートの確認 62
AIR for TV ネイティブ拡張の構築 63
AIR for TV ネイティブ拡張へのリソースの追加 69
AIR for TV ネイティブ拡張の配布 69
AIR for TV デバイス上での AIR アプリケーションの実行 70

第 7 章：ネイティブ拡張記述ファイル

拡張記述ファイルの構造 71
ネイティブ拡張記述エレメント 72

第 8 章：ネイティブ C API リファレンス

typedef 80
構造体 typedef 81
列挙型 84
実装する関数 85
ユーザー使用関数 90

第 9 章：Android Java API リファレンス

インターフェイス 115
クラス 119

第 1 章：Adobe AIR 用ネイティブ拡張の概要

Adobe AIR 用ネイティブ拡張は、ActionScript API にラップされたネイティブコードを含むコードライブラリです。AIR アプリケーションでネイティブ拡張を使用すると、AIR でサポートされていないプラットフォーム機能にアクセスしたり、重要なアルゴリズムに対してネイティブコードレベルのパフォーマンスを実現したり、既存のネイティブコードライブラリを再利用したりできます。

ネイティブ拡張について

Adobe AIR とは

Adobe® AIR® は、コンテンツ開発者がリッチインターネットアプリケーション（RIA）を構築できる、クロスオペレーティングシステムのランタイムです。開発者はデスクトップ、モバイルデバイスおよびデジタルホームデバイスに RIA をデプロイできます。AIR アプリケーションは Adobe® Flex® および Adobe® Flash®（SWF ベース）を使用して構築でき、HTML、JavaScript および Ajax（HTML ベース）でも構築できます。AIR アプリケーションの構築に使用できる Adobe Flash Platform ツールについて詳しくは、『**Adobe AIR** アプリケーションの構築』の「AIR 開発用の Adobe Flash Platform ツール」を参照してください。

Adobe ActionScript とは

SWF ベースの AIR アプリケーションは Adobe ActionScript® 3.0 を使用できます。ActionScript 3.0 は、RIA にインタラクティブ性とデータハンドリングを追加できるオブジェクト指向言語です。この言語について詳しくは、『**ActionScript 3.0** の学習』および『**ActionScript 3.0** 開発ガイド』を参照してください。

ActionScript では多くのビルトインクラスが用意されています。例えば、MovieClip、Array、NetConnection はビルトインの ActionScript クラスです。さらに、コンテンツ開発者はアプリケーション固有のクラスを作成できます。アプリケーション固有のクラスはビルトインクラスの派生クラスとなる場合もあります。

ランタイムは ActionScript クラスのコードを実行します。また、ランタイムは HTML ベースのアプリケーションで使用される JavaScript コードも実行します。

ネイティブ拡張とは

ネイティブ拡張とは、次の 2 つを組み合わせたものです。

- ActionScript クラス
- ネイティブコードネイティブコードは、ここではランタイムの外部で実行するコードとして定義されます。例えば、C で記述されたコードはネイティブコードです。一部のプラットフォームでは、Java コードによる拡張もサポートされています。このドキュメントでは、Java コードも「ネイティブ」コードと見なされます。

ネイティブ拡張を記述する理由としては、以下があります。

- ネイティブコード実装を使用すると、デバイス固有の機能にアクセスできます。これらのデバイス固有の機能はビルトイン ActionScript クラスでは利用できません。また、アプリケーション固有の ActionScript クラスで実装することもできません。ネイティブコード実装では、デバイス固有のハードウェアおよびソフトウェアにアクセスできるので、このような機能を提供できます。
- ネイティブコード実装は ActionScript のみを使用する実装よりも速い場合があります。
- ネイティブコード実装によって既存コードを再利用できます。

例えば、アプリケーションで次のことを実行するネイティブ拡張を作成できます。

- モバイルデバイスの振動
- デバイス固有のライブラリおよび機能の操作

ActionScript およびネイティブの実装を完了したら、拡張をパッケージ化します。次に、AIR アプリケーション開発者はこのパッケージを使用して、拡張の ActionScript API を呼び出し、デバイス固有の機能を実行することができます。拡張は、AIR アプリケーションと同じプロセスで実行されます。

ネイティブ拡張と NativeProcess ActionScript クラスの比較

ActionScript 3.0 では、NativeProcess クラスが提供されます。このクラスを使用すると、AIR アプリケーションで、ホストオペレーティングシステム上でネイティブプロセスを実行できます。この機能はネイティブ拡張と類似しており、デバイス固有の機能およびライブラリにアクセスできます。NativeProcess クラスを使用するかネイティブ拡張を作成するかを判断する際には、次の点を考慮してください。

- NativeProcess クラスをサポートするのは extendedDesktop AIR プロファイルのみです。したがって、AIR プロファイル mobileDevice および extendedMobileDevice を使用しているアプリケーションの場合は、ネイティブ拡張が唯一の選択肢です。
- ネイティブ拡張の開発者は、多くの場合、様々なプラットフォーム用のネイティブ実装を提供しますが、ネイティブ拡張の開発に伴って提供される ActionScript API は、通常、どのプラットフォームでも同じになります。NativeProcess クラスを使用する場合、ネイティブプロセスを開始するための ActionScript コードは、プラットフォームによって異なることがあります。
- NativeProcess クラスは別プロセスを開始します。一方、ネイティブ拡張は AIR アプリケーションと同じプロセスで実行されます。そのため、コードの失敗が懸念される場合は、NativeProcess クラスを使用する方が安全です。ただし、別プロセスになると、通常はプロセス間通信処理を実装することになります。

ネイティブ拡張と ActionScript クラスライブラリ（SWC ファイル）の比較

ネイティブ拡張と SWC ファイルの最大の違いは、SWC ファイルにはネイティブコードが含まれていないことです。そのため、ネイティブコードを使用しなくても目的を達成できると判断した場合は、ネイティブ拡張ではなく SWC ファイルを使用してください。

関連項目

About SWC files

サポートされているデバイス

次のデバイス向けのネイティブ拡張を作成できます。

- Android デバイス（AIR 3 および Android 2.2 以降）
- iOS デバイス（AIR 3 および iOS 4.0 以降）
- iOS シミュレーター（AIR 3.3 以降）
- Blackberry PlayBook（AIR 2.7 以降）
- AIR 3.0 をサポートする Windows デスクトップデバイス
- AIR 3.0 をサポートする Mac OS X デスクトップデバイス

拡張は複数のプラットフォームをターゲットにできます。詳しくは、5 ページの「複数プラットフォームのターゲット」を参照してください。

サポートされているデバイスプロファイル

ネイティブ拡張は以下の AIR プロファイルでサポートされています。

- extendedDesktop（AIR 3.0 以降）
- mobileDevice（AIR 3.0 以降）

関連項目

デバイスプロファイル

ネイティブ拡張のアーキテクチャ

アーキテクチャの概要

AIR で拡張を使用すると次のことができます。

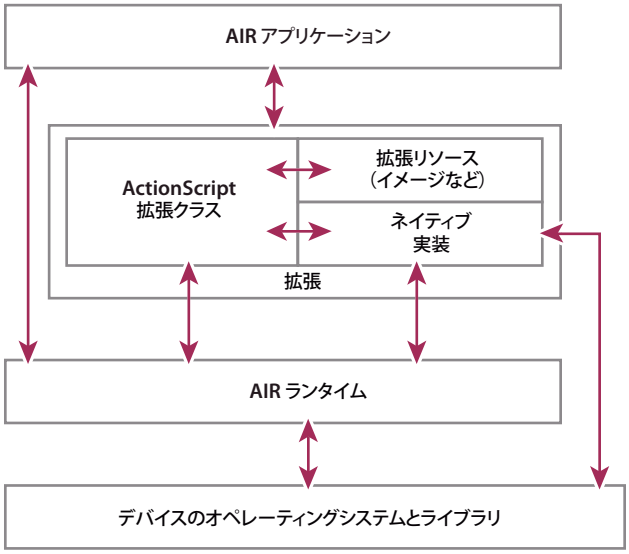
- ActionScript から、ネイティブコードで実装された関数を呼び出します。
- ActionScript とネイティブコードの間でデータを共有します。
- ネイティブコードから ActionScript にイベントを送出します。

ネイティブ拡張を作成する際には、次の項目を設定します。

- ユーザー定義の ActionScript 拡張クラス。これらの ActionScript クラスでは、ネイティブコードにアクセスし、ネイティブコードとデータを交換できる、ビルトインの ActionScript API を使用します。
- ネイティブコード実装。ネイティブコードでは、ユーザーの ActionScript 拡張クラスにアクセスし、それらとデータを交換できる、ネイティブコード API を使用します。
- ActionScript 拡張クラスまたはネイティブコードが使用する、画像などのリソース。

ネイティブ拡張では、複数のプラットフォームをターゲットにすることができます。複数のプラットフォームをターゲットにする場合は、それぞれのターゲットプラットフォーム向けに、異なる ActionScript 拡張クラスおよびネイティブコード実装のセットを作成できます。詳しくは、5 ページの「複数プラットフォームのターゲット」を参照してください。

次の図に、ネイティブ拡張、AIR ランタイム、デバイスの間で連携する様子を示します。



ネイティブ拡張のアーキテクチャ

ネイティブコードのプログラミング言語

Adobe AIR では、ネイティブコード実装が ActionScript 拡張クラスとの連携に使用する、ネイティブコード API が提供されます。これらの API は次の言語で使用できます。

- C プログラミング言語
- Java

ネイティブコード実装では、C API または Java API のいずれかを使用して（ただし両方同時には使用しない）、ActionScript 拡張クラスを操作します。ただし、残りのネイティブコード実装で、API と異なる言語を使用しても問題はありません。例えば、C API を使用する開発者は次の言語も使用できます。

- C++
- Objective-C
- アセンブラコード（高度に最適化されたルーチンの活用）

次の表に、ターゲットデバイスに応じて使用すべき拡張 API を示します。

デバイス	**使用するネイティブコード API**
Android デバイス	Java API と Android SDK C API と Android NDK
iOS デバイス	C API
Blackberry PlayBook	C API
Windows デスクトップデバイス	C API
Mac OS X デスクトップデバイス	C API

複数プラットフォームのターゲット

ネイティブ拡張では、複数のプラットフォームをターゲットにすることが多くあります。例えば、1 つの拡張が、iOS を実行しているデバイスと Android を実行しているデバイスをターゲットとする場合があります。この場合、ネイティブコード言語を含む、ActionScript クラス実装およびネイティブコード実装はターゲットプラットフォームに基づいて変更できます。

ベストプラクティスとしては、ActionScript 拡張クラスで、実装にかかわらず同じ ActionScript パブリックインターフェイスを提供します。常に同じパブリックインターフェイスを使用することで、真のクロスプラットフォームネイティブ拡張となります。ActionScript パブリックインターフェイスが同じで、ActionScript 実装が異なる場合は、プラットフォームごとに別々の ActionScript ライブラリを作成します。

一部のターゲットプラットフォームに対して、ネイティブコード実装を含まない拡張を作成することもできます。このような拡張は、次のような場合に便利です。

- 一部のターゲットプラットフォームのみが、目的の機能のネイティブ実装をサポートしている場合。

  拡張はこれらのプラットフォームのみでネイティブ実装を使用できますが、他のプラットフォーム上では ActionScript 専用の実装を使用します。例えば、コンピュータープロセス間通信のための特殊なメカニズムを提供するプラットフォームがあるとします。このプラットフォーム用の拡張にはネイティブ実装を用意します。他のプラットフォーム用の同じ拡張は、ActionScript Socket クラスを使用した ActionScript 専用の拡張とします。

  アプリケーション開発者が拡張を使用するときには、異なるターゲットプラットフォーム上で拡張がどのように実装されているのかを知らなくても、1 つのアプリケーションを記述できます。

- 拡張をテストする場合。

  モバイルデバイスの特定の機能を使用するネイティブ拡張があるとします。この場合、デスクトップ向けの ActionScript 専用の拡張を作成できます。次に、アプリケーション開発者は、実際のターゲットデバイスでテストする前に、開発中にシミュレーションテストを行うためにデスクトップ拡張を使用できます。同様に、拡張の開発者は、ネイティブコード実装と連携する前に、ActionScript 側の拡張をテストできます。

拡張をパブリッシュするときに、拡張記述ファイル内の <platform> エレメントでターゲットプラットフォームを指定できます。個々の <platform> エレメントには、ターゲットの名前を指定します（例：iPhone-ARM、Windows-x86）。また、default という名前の <platform> エレメントを指定することもできます。default プラットフォームには、<platform> エレメントで指定されていないすべてのプラットフォームで使用する ActionScript 専用の実装があります。詳しくは、71 ページの「ネイティブ拡張記述ファイル」を参照してください。

注意：少なくとも 1 つのターゲットプラットフォームの実装に、ネイティブコードが含まれている必要があります。ネイティブコードを必要とするターゲットプラットフォームがない場合は、ネイティブ拡張の使用は、適切な方法ではありません。その場合は SWC ライブラリを作成します。

ランタイムでの拡張の利用

アプリケーションはランタイムで、次のいずれかの方法でネイティブ拡張を利用できます。

アプリケーションバンドル 拡張は AIR アプリケーションと共にパッケージ化され、アプリケーションと共にターゲットデバイスにインストールされます。通常、拡張パッケージには、複数のプラットフォーム用のネイティブ実装と ActionScript 実装が含まれますが、1 つのプラットフォーム専用のネイティブ実装と ActionScript 実装が含まれる場合もあります。また、サポート対象外のプラットフォーム用またはテストプラットフォーム用に、ActionScript 専用の実装も含まれる場合があります。

デバイスバンドル 拡張は、ターゲットデバイス上のディレクトリ内に、AIR アプリケーションとは独立してインストールされます。デバイスバンドルを使用するには、通常、デバイス製造元の協力を得て、デバイス上に拡張をインストールします。

次の表に、アプリケーションバンドルとデバイスバンドルに関するデバイスのサポート状況を示します。

	アプリケーションバンドル	デバイスバンドル
Android	可	不可
iOS	可	不可
Blackberry PlayBook	可	可
Windows	可	不可
Mac OS X	可	不可

拡張コンテキスト

ネイティブ拡張は、アプリケーションが実行されるたびに 1 回ロードされます。ただし、ネイティブ実装を使用するには、拡張の ActionScript 側で特殊な ActionScript API を呼び出して、「拡張コンテキスト」を作成します。

ネイティブ拡張では次のいずれかの操作を実行できます。

- 拡張コンテキストを 1 つだけ作成します。

  拡張コンテキストを 1 つだけ作成する方法は、ネイティブ実装で 1 つの関数セットのみを提供するシンプルな拡張で一般的です。

- 共存する複数の拡張コンテキストを作成します。

  複数の拡張コンテキストは、ActionScript オブジェクトをネイティブオブジェクトと関連付けるのに便利です。ActionScript オブジェクトとネイティブオブジェクト間の関連のそれぞれが、1 つの拡張コンテキストインスタンスとなります。これらの拡張コンテキストインスタンスには異なるコンテキストタイプを設定できます。ネイティブ実装は各コンテキストタイプ用に、異なる関数セットを提供できます。

それぞれの拡張コンテキストには、ネイティブ実装内で定義して使用するコンテキスト固有のデータを含めることができます。

拡張コンテキストは、拡張内の ActionScript コードでのみ作成できます。ネイティブコードやアプリケーションコードでは作成できません。

ネイティブ拡張を作成するためのタスクの概要

ネイティブ拡張を作成するには、次のタスクを実行します。

1. ActionScript 拡張クラスのメソッドおよびプロパティを定義します。
2. ActionScript 拡張クラスをコーディングします。

   7 ページの「ActionScript 側のコーディング」を参照してください。

3. ネイティブ実装をコーディングします。

   15 ページの「C を使用したネイティブ側のコーディング」および 29 ページの「Java を使用したネイティブ側のコーディング」を参照してください。

4. ActionScript 側とネイティブ側を構築し、拡張記述ファイルを作成して、拡張およびそのリソースをパッケージ化します。

   すべてのデバイスについて、39 ページの「ネイティブ拡張のパッケージ化」を参照してください。

5. ActionScript 拡張クラスのパブリックインターフェイスをドキュメント化します。

通常、他のソフトウェア開発と同様に、これらの手順の実施は反復的なプロセスです。

第 2 章：ActionScript 側のコーディング

ネイティブ拡張は次の 2 つの部分で構成されています。

- 定義した ActionScript 拡張クラス
- ネイティブ実装

ActionScript 拡張クラスは、ネイティブ実装を使用してデータにアクセスし、データを交換します。このアクセスは、ActionScript クラスの ExtensionContext を使用して行われます。ExtensionContext クラスのメソッドにアクセスできるのは、拡張に含まれる ActionScript コードだけです。

拡張の ActionScript 側のコーディングには、次のタスクがあります。

- ActionScript 拡張クラスのパブリックインターフェイスを宣言します。
- 静的メソッド ExtensionContext.createExtensionContext() を使用して、ExtensionContext インスタンスを作成します。
- ExtensionContext インスタンスの call() メソッドを使用して、ネイティブ実装内のメソッドを呼び出します。
- ExtensionContext インスタンスにイベントリスナーを追加し、ネイティブ実装から送出されたイベントを監視します。
- dispose() メソッドを使用して、ExtensionContext インスタンスを削除します。
- ActionScript 側とネイティブ側の間でデータを共有します。どの ActionScript オブジェクトもデータとして共有できます。
- getExtensionDirectory() メソッドを使用して、拡張のインストール先ディレクトリにアクセスします。拡張に関連するすべての情報およびリソースはこのディレクトリ内にあります（iOS デバイスに関しては、この規則に例外があります）。

ネイティブ拡張の例については、「Adobe AIR 用のネイティブ拡張」を参照してください。

ExtensionContext クラスについて詳しくは、『**Adobe Flash Platform** 用 **ActionScript 3.0** リファレンスガイド』を参照してください。

パブリックインターフェイスの宣言

ネイティブ拡張を作成する最初の手順は、拡張のパブリックインターフェイスを決定することです。アプリケーションコードではこれらのパブリックインターフェイスを使用して拡張を操作します。ActionScript コードは、.as 拡張子のファイルに保存されます。クラス定義を含む .as ファイルを作成します。例えば、次のコードでは、まだ実装されていない単純な TVChannelController 拡張クラスの宣言を示しています。この単純なクラスによって、アプリケーションは仮想テレビのチャンネル設定を操作できます。

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

        public function TVChannelController() {
        }

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

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

注意：パブリックインターフェイスを設計する際には、拡張の後続バージョンをリリースするかどうかを検討します。このような場合には、初期の設計で、上位互換性をサポートすることを考慮してください。デバイスバンドル拡張の下位互換性問題について詳しくは、13 ページの「ネイティブ拡張の下位互換性」を参照してください。

ネイティブ拡張のサポートの確認

ベストプラクティスは、ネイティブ拡張と AIR アプリケーションとがハンドシェークするパブリックインターフェイスを必ず定義することです。作成した拡張を使用する AIR アプリケーション開発者に対して、他の拡張メソッドを呼び出す前にこのメソッドを確認するよう指示します。

例えば、isSupported() という ActionScript 拡張クラスパブリックインターフェイスについて考えます。AIR アプリケーションはこの isSupported() メソッドを使用して、アプリケーションを実行するデバイスで拡張がサポートされているかどうかに応じて、論理的な判断を行うことができます。isSupported() が false を返す場合は、AIR アプリケーションは拡張を使用せずに、何を実行するかを決定する必要があります。例えば、AIR アプリケーションを終了するという判断を行うことができます。

ExtensionContext インスタンスの作成

ネイティブ実装の操作を開始するには、ActionScript 拡張クラスで、ExtensionContext の createExtensionContext() 静的メソッドを使用します。このメソッドは、ExtensionContext クラスの新しいインスタンスを返します。

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

        private var extContext:ExtensionContext;

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

        }
        .
        .
        .
    }
}
```

次の例では、コンストラクターが createExtensionContext() を呼び出しています。拡張クラスは、拡張クラスのどのメソッド内でも createExtensionContext() を呼び出すことができますが、通常はコンストラクターまたは他の初期化メソッドでこれを呼び出します。返された ExtensionContext インスタンスをクラスのデータメンバーに保存します。

注意：createExtensionContext() を静的データメンバーの定義の一部として呼び出すことは推奨されません。これを行うと、ランタイムによって、アプリケーションが必要とするよりも前に拡張コンテキストが作成されます。アプリケーションの実行パスが最終的に拡張を使用しない場合、コンテキストの作成によってデバイスリソースが浪費されます。

createExtensionContext() メソッドには、拡張 ID とコンテキストタイプという 2 つのパラメーターがあります。

拡張 ID

createExtensionContext() メソッドには、拡張の識別子、つまり名前を表す String パラメーターがあります。この名前は、拡張記述ファイルの id エレメントで使用する名前と同じです（拡張記述ファイルは、拡張をパッケージ化するときに作成します）。アプリケーション開発者は、アプリケーション記述ファイルの extensionID エレメントでもこの値を使用します。指定された名前の拡張が使用できない場合、createExtensionContext() は Null を返します。

名前の競合を避けるため、DNS を逆にした文字列を拡張 ID に使用することをお勧めします。例えば、TVControllerChannel 拡張の ID は com.example.TVControllerExtension となります。すべての拡張が単一のグローバル名前空間を共有するので、拡張 ID に DNS を逆にした文字列を使用すると、拡張間で名前が競合するのを回避できます。

コンテキストタイプ

createExtensionContext() メソッドには、新しい拡張コンテキストのコンテキストタイプを表す String パラメーターがあります。この文字列には、新しい拡張コンテキストで実行することに関する詳細情報を指定します。

例えば、com.example.TVControllerExtension 拡張では、チャンネルとボリューム設定の両方を操作できるとします。createExtensionContext() に "channel" または "volume" を渡すことで、新しい拡張コンテキストでどちらの機能が使用されるかを示します。TVVolumeController などの拡張内の他の ActionScript クラスでは、contextType 値として "volume" を指定して createExtensionContext() を呼び出すことができます。ネイティブ実装では、コンテキストの初期化時にこの contextType 値が使用されます。

通常、定義可能なコンテキストタイプの各値は、ネイティブ実装内の異なるメソッドセットに対応します。そのため、コンテキストタイプは実質的に、ネイティブ実装のクラスに対応します。同じコンテキストタイプを指定して createExtensionContext() を複数回呼び出すと、通常はネイティブ実装で特定のネイティブクラスの複数のインスタンスが作成されます。

createExtensionContext() を複数回呼び出すときのコンテキストタイプが異なる場合、通常はネイティブ側で異なる初期化処理が実行されます。コンテキストタイプに応じて、ネイティブ側で異なるネイティブクラスのインスタンスを作成し、異なるネイティブ関数セットを提供できます。

注意：単純な拡張では多くの場合、コンテキストタイプは 1 つのみです。つまり、ネイティブ実装で 1 つのメソッドセットのみが用意されます。このような単純なケースでは、コンテキストタイプの String パラメーターを Null に設定できます。

ネイティブ関数の呼び出し

ActionScript 拡張クラスは ExtensionContext.createExtensionContext() を呼び出した後に、ネイティブ実装のメソッドを呼び出すことができます。TVChannelController の例では、次のようにしてネイティブメソッドの "setDeviceChannel" および "getDeviceChannel" を呼び出します。

```
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 の call() メソッドには、次のパラメーターがあります。

- functionName。この文字列はネイティブ実装内の関数を表します。TVChannelController の例では、functionName の文字列は ActionScript メソッド名と異なります。これらの名前を同じに設定することもできます。また、functionName の文字列を、それが表すネイティブ関数の名前と同じにするかどうかを選択することもできます。ネイティブ実装で、この functionName の文字列とネイティブ関数とを関連付けます。この関連付けは、FREContextInitializer() メソッドの出力パラメーターにあります。17 ページの「拡張コンテキストの初期化」を参照してください。
- パラメーターに関するオプションのリスト。各パラメーターはネイティブ関数に渡されます。パラメーターには、int などのプリミティブ型か、任意の ActionScript オブジェクトを指定できます。

ExtensionContext の call() メソッドの戻り値は、プリミティブ型または ActionScript オブジェクトです。このメソッドが返すオブジェクトのサブクラスは、ネイティブ関数の戻り値によって異なります。例えば、ネイティブ実装の "getDeviceChannel" は int を返します。

イベントの監視

ネイティブ実装は、ActionScript 拡張コードが監視できるイベントを送出できます。このメカニズムによって、ネイティブ実装は、タスクを非同期に実行し、タスク完了時に ActionScript 側に通知することができます。

イベントターゲットは ExtensionContext インスタンスです。そのため、ExtensionContext インスタンスの addEventListener() メソッドを使用して、ネイティブ実装からのイベントをサブスクライブします。

次の例では、TVChannelController に、ネイティブ実装からのイベントを受け取るコードを追加します。拡張を使用するアプリケーションは、ActionScript 拡張クラスの scanChannels() メソッドを呼び出します。次に、このメソッドが、"scanDeviceChannels" ネイティブ関数を呼び出します。

このネイティブ関数は、使用できるすべてのチャンネルを非同期にスキャンします。スキャンが完了すると、イベントを送出します。onStatus() メソッドが、"getDeviceChannels" ネイティブメソッドにチャンネルのリストを照会することで、イベントを処理します。onStatus() メソッドはそのリストを scannedChannelList データメンバーに格納し、アプリケーションの監視中のオブジェクトにイベントを送出します。このアプリケーションオブジェクトがイベントを受け取ると、ActionScript 拡張クラスのプロパティアクセッサー availableChannels を呼び出すことができます。

```
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") );
            }
        }
    }
}
```

この例で示されるポイントは次のとおりです。

- ネイティブ実装が送出できるのは StatusEvent オブジェクトだけです。そのため、addEventListener() メソッドでは、StatusEvent.STATUS イベントタイプを監視します。
- ネイティブ実装は、StatusEvent オブジェクトの code プロパティおよび level プロパティを設定します。これらのプロパティに使用する文字列を定義できます。この例では、ネイティブ実装は level プロパティを "status" に設定し、code プロパティを "scanCompleted" に設定します。通常、StatusEvent の level プロパティの値は "status"、"info" または "error" です。
- TVChannelController は EventDispatcher のサブクラスなので、このクラスもイベントを送出できます。この例では、type プロパティを "scanCompleted" に設定して Event オブジェクトを送出します。このイベントに関連するすべての ActionScript オブジェクトが、このイベントを監視できます。例えば、次のコードでは、この拡張を使用する AIR アプリケーションのスニペットを示しています。このアプリケーションは TVChannelController オブジェクトを作成します。次に、TVChannelController オブジェクトに、チャンネルをスキャンするよう要求します。この後、スキャンが完了するのを待ちます。

```
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;}
}
```

ExtensionContext インスタンスの破棄

ActionScript 拡張クラスは、ExtensionContext の dispose() メソッドを呼び出すことで、ExtensionContext インスタンスを破棄できます。このメソッドはネイティブ実装に対して、インスタンスが使用しているリソースをクリーンアップするよう通知します。例えば、TVChannelController クラスに、次のようなクリーンアップ用メソッドを追加できます。

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

ActionScript 拡張クラスでは、ExtensionContext インスタンスの dispose() メソッドを明示的に呼び出す必要はありません。その場合は、ランタイムのガベージコレクターによって ExtensionContext インスタンスが破棄されるときにこのメソッドが呼び出されます。ただし、ベストプラクティスとしては、dispose() を明示的に呼び出します。dispose() の明示的な呼び出しによって、通常、ガベージコレクター処理を待つよりもはるかに早くリソースがクリーンアップされます。

明示的に呼び出された場合も、ガベージコレクターによって呼び出された場合も、ExtensionContext の dispose() メソッドの結果、ネイティブ実装のコンテキストファイナライザーが呼び出されます。詳しくは、18 ページの「拡張コンテキストのファイナライズ処理」を参照してください。

ネイティブ拡張のディレクトリへのアクセス

拡張には、画像などの付加的なファイルが含まれる場合があります。また、拡張は、拡張バージョン番号などの拡張記述ファイル内の情報にアクセスする必要がある場合もあります。

iOS 以外のすべてのデバイスで拡張のこれらのファイルにアクセスするには、ExtensionContext クラスの静的メソッド getExtensionDirectory() を使用します。次に、例を示します。

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

getExtensionDirectory() に拡張の名前を渡します。この String 値は、次の項目で使用する名前と同じです。

- id エレメント内の拡張記述ファイル
- ExtensionContext.createExtensionContext() に渡す拡張 ID パラメーター

返された File インスタンスは、拡張の基本ディレクトリを参照します。拡張ディレクトリの構造は次のとおりです。

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

デバイス上のどこに拡張ディレクトリがあるかにかかわらず、拡張のファイルは常に拡張の基本ディレクトリと相対的に同じ場所に配置されます。そのため、返された File インスタンスおよび File クラスのメソッドを使用して、拡張に含まれる特定のファイルの場所まで移動し、そのファイルを操作します。

拡張ディレクトリの場所は、次のように、拡張がアプリケーションバンドルを通して使用できるか、デバイスバンドルを通して使用できるかによって異なります。

- アプリケーションバンドルでは、拡張ディレクトリはアプリケーションディレクトリ内に配置されています。
- デバイスバンドルでは、拡張ディレクトリの場所はデバイスによって異なります。

例外として、iOS デバイス用の ActionScript 拡張の場合は、getExtensionDirectory() を使用します。これらの拡張のリソースは、拡張ディレクトリではなく、アプリケーションのトップレベルのディレクトリに配置されています。詳しくは、53 ページの「iOS デバイス上のリソース」を参照してください。

関連項目

5 ページの「ランタイムでの拡張の利用」

ネイティブ拡張からの呼び出し元アプリケーションの特定

ActionScript 側の拡張では、拡張を使用して AIR アプリケーションを特定し評価することができます。例えば、ID や署名データなどの AIR アプリケーションに関する情報を取得するには、ActionScript の NativeApplication クラスを使用します。次に、ActionScript 側では、この情報に基づいてランタイムの決定を行うことができます。

ネイティブ実装でも、同様のランタイムの決定を行う場合があります。この場合は、ActionScript 側では ExtensionContext インスタンスの call() メソッドを使用して、ネイティブ実装にアプリケーション情報を通知できます。

ネイティブ拡張の下位互換性

下位互換性と拡張のパブリックインターフェイス

ベストプラクティスとしては、拡張の ActionScript パブリックインターフェイスで下位互換性を維持します。拡張のすべての後続バージョンで、拡張のクラス、メソッド、プロパティおよびイベントを継続的にサポートします。

デバイスバンドル拡張には、下位互換性に関して、より複雑な問題があります。拡張の「動作」は拡張のバージョン間で異なる場合があります。例えば、あるメソッドが、新しいバージョンの拡張では新しい意味を持つ値を返すことがあります。デバイスバンドル拡張でこの動作が発生した場合、アプリケーションが正常に機能しなくなる場合があります。この問題は、アプリケーションの構築で使用された拡張のバージョンが、デバイス上にインストールされた拡張のバージョンと異なる動作をする場合に発生することがあります。この場合、アプリケーションが予期する動作に対して、インストールされた拡張が異なる動作を提供します。

このような場合は、デバイスにインストールされた拡張で、続行する方法を決定できます。拡張は次のことを実行できます。

- AIR アプリケーションの構築に使用した拡張バージョンと、デバイスにインストールされたバージョンを確認します。
- 拡張の動作がこの 2 つのバージョンで異なるかどうかを判断します。
- AIR アプリケーションがより古いバージョンの拡張を使用して構築された場合は、古いバージョンの動作に戻します。

  注意：通常、デバイスよりも新しいバージョンの拡張を使用して構築された AIR アプリケーションは、デバイスでは利用できません。詳しくは、14 ページの「下位互換性とデバイスのアプリケーションストア」を参照してください。

アプリケーションの構築に使用した拡張のバージョン番号を確認するには、以下を実行します。

1. File.applicationDirectory を使用してアプリケーションのインストールディレクトリを取得します。
2. File クラスの API を使用して、アプリケーションの構築で使用した拡張の extension.xml ファイルにアクセスします。このファイルは次の場所にあります。

   `<application directory>/META-INF/AIR/extensions/<extensionID>/META-INF/ANE/extension.xml`
3. extension.xml ファイルのコンテンツを読み取り、<versionNumber> エレメントの値を検索します。

インストールされた拡張のバージョン番号を確認するには、以下を実行します。

1. ExtensionContext.getExtensionDirectory() 静的メソッドを使用して、拡張の基本ディレクトリを取得します。
2. File クラスの API を使用して、デバイスにインストールされた拡張の extension.xml ファイルにアクセスします。このファイルは次の場所にあります。

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

3. extension.xml ファイルのコンテンツを読み取り、<versionNumber> エレメントの値を探します。

下位互換性とデバイスのアプリケーションストア

デバイスにインストールされた拡張よりも新しいバージョンの拡張を使用して構築された AIR アプリケーションは、通常はデバイスでは利用できません。アプリケーションが利用できない原因は、デバイスの製造元が、デバイスのアプリケーションストアから、そのようなアプリケーションをダウンロードするサーバーへの要求を処理する方法にあります。アドビはデバイス製造元に対して、次の処理を行うことをお勧めします。

- サーバーが、新しいバージョンの拡張を使用するアプリケーションをダウンロードし、新しいバージョンの拡張もダウンロードする場合を考慮してください。この場合、デバイスのアプリケーションストアでは、アプリケーションと新しいバージョンの拡張の両方をインストールします。
- サーバーが新しいバージョンの拡張をダウンロードできず、そのバージョンの拡張を使用するアプリケーションもダウンロードしない場合を考慮してください。この場合、デバイスのアプリケーションストアでは、このシナリオを正常に処理し、必要に応じてエンドユーザーに通知します。
- サーバーは、新しいバージョンの拡張を使用するアプリケーションをダウンロードしますが、新しいバージョンの拡張はダウンロードしない場合を考慮してください。この場合、デバイスのアプリケーションストアでは、エンドユーザーがアプリケーションを実行できないようにします。アプリケーションストアでは、このシナリオを正常に処理し、必要に応じてエンドユーザーに通知します。

第 3 章：C を使用したネイティブ側のコーディング

一部のデバイスでは、ネイティブ実装で C プログラミング言語を使用します。ネイティブ拡張でこのようなデバイスをターゲットにする場合は、ネイティブ拡張 C API を使用して拡張のネイティブ側をコーディングします。

C API は、FlashRuntimeExtensions.h ファイル内にあります。このファイルは AIR SDK の include ディレクトリにあります。AIR SDK は、http://www.adobe.com/jp/special/products/air/sdk/ で取得できます。

AIR ランタイムは、拡張のネイティブ側と ActionScript 側を接続します。

C API を使用して、次のタスクを実行できます。

- 拡張を初期化します。
- 拡張の作成時に、それぞれの拡張コンテキストを初期化します。
- ActionScript 側が呼び出すことのできる関数を定義します。
- ActionScript 側にイベントを送出します。
- ActionScript 側から渡されたデータにアクセスし、ActionScript 側にデータを返します。
- コンテキスト固有のネイティブデータおよびコンテキスト固有の ActionScript データを作成し、これらのデータにアクセスします。
- 拡張の操作の完了時に、拡張リソースをクリーンアップします。

パラメーターや戻り値など、C API 関数について詳しくは、80 ページの「ネイティブ C API リファレンス」を参照してください。

C API を使用するネイティブ拡張の例については、「Adobe AIR 用のネイティブ拡張」を参照してください。

拡張の初期化

ランタイムは、ネイティブ側の拡張初期化関数を呼び出します。拡張を使用するアプリケーションが実行されるたびに 1 回だけ初期化関数を呼び出します。具体的には、任意のコンテキストで拡張が ExtensionContext.createExtensionContext() を最初に呼び出したときに、ランタイムは初期化関数を呼び出します。

初期化関数は、すべての拡張コンテキストが使用できるデータを初期化します。拡張の initializer 関数は、FREInitializer() のシグネチャを使用して定義します。

次に、例を示します。

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

FREInitializer() メソッドを定義すると、次のデータがランタイムに返されます。

- ランタイムが後で新しい拡張コンテキストのそれぞれに渡すデータへのポインター。例えば、すべての拡張コンテキストが同じユーティリティライブラリを使用する場合、このデータにはライブラリへのポインターを含めることができます。このデータは「拡張データ」と呼ばれます。

  ユーザーが選択したどのデータも拡張データとすることができます。単純なプリミティブデータ型や、ユーザー定義の構造体へのポインターを、拡張データとして使用できます。この例では、ポインターは NULL です。これは、拡張がこのデータを使用しないからです。

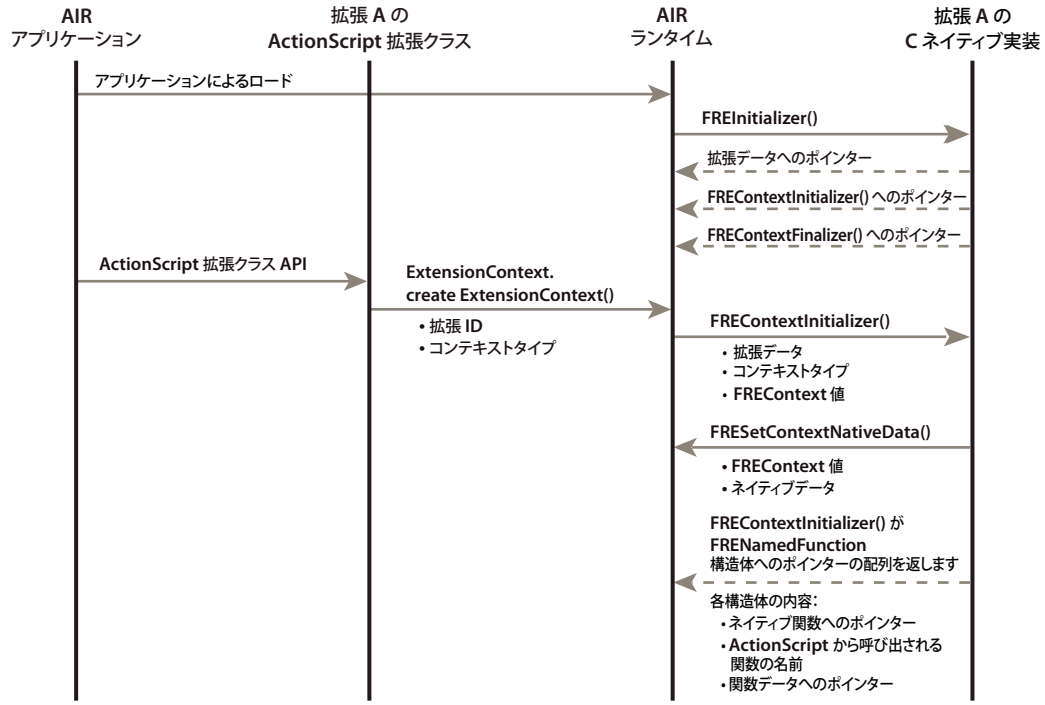
- コンテキスト初期化関数へのポインター。ActionScript 側が ExtensionContext.createExtensionContext() を呼び出すたびに、ランタイムはユーザーが指定する拡張コンテキスト初期化関数を呼び出します。86 ページの「FREContextInitializer()」を参照してください。

- コンテキストファイナライザー関数へのポインター。ランタイムは、拡張コンテキストを破棄するときにこの関数を呼び出します。この呼び出しは、ActionScript 側が ExtensionContext インスタンスの dispose() メソッドを呼び出すと発生します。dispose() が呼び出されない場合は、ランタイムは ExtensionContext インスタンスのガベージコレクションを行います。86 ページの「FREContextFinalizer()」を参照してください。

アプリケーションバンドル拡張では、FREInitializer() の実装は任意の名前に設定できます。拡張記述ファイル内に初期化関数の名前を指定します。71 ページの「ネイティブ拡張記述ファイル」を参照してください。

デバイスバンドルアプリケーションでは、拡張の initializer 関数の指定方法はデバイスによって異なります。

次のシーケンス図に、AIR ランタイムが FREInitializer() 関数を呼び出す様子を示します。また、コンテキストの初期化についても示します。詳しくは、17 ページの「拡張コンテキストの初期化」を参照してください。



拡張の初期化シーケンス

拡張コンテキストの初期化

ネイティブ C メソッドを使用するには、拡張の ActionScript 側で、最初に ExtensionContext.createExtensionContext() 静的メソッドを呼び出します。createExtensionContext() を呼び出すと、ランタイムは次の処理を実行します。

- ExtensionContext インスタンスを作成します。
- 拡張コンテキストの追跡に使用する内部データを作成します。
- 拡張コンテキスト初期化関数を呼び出します。

拡張コンテキスト初期化関数は、新しい拡張コンテキスト向けのネイティブ実装を初期化します。拡張コンテキスト初期化関数は、FREContextInitializer() のシグネチャを使用して定義します。

例えば、Vibration の例では、次の関数を使用します。

```
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;
}
```

コンテキスト初期化関数は、次の入力パラメーターを受け取ります。

- 拡張初期化関数が作成した拡張データ。15 ページの「拡張の初期化」を参照してください。
- コンテキストタイプ。ActionScript メソッドの ExtensionContext.createExtensionContext() に、コンテキストタイプを指定するパラメーターが渡されます。ランタイムは、コンテキスト初期化関数に、この文字列値を渡します。次に、この関数はこのコンテキストタイプを使用して、ActionScript 側で呼び出すことのできる、ネイティブ実装内のメソッドセットを選択します。各コンテキストタイプは様々なメソッドセットに対応します。9 ページの「コンテキストタイプ」を参照してください。

  コンテキストタイプの値は、ActionScript 側とネイティブ側の間で合意した任意の文字列です。

  拡張がネイティブ実装でのメソッドセットを 1 つだけ持つ場合は、ExtensionContext.createExtensionContext() に null または空の文字列を渡します。さらに、拡張コンテキスト initializer では、このコンテキストタイプパラメーターを無視します。

- FREContext 値。ランタイムは、拡張コンテキストを作成するときに内部データを作成します。内部データと ActionScript 側の ExtensionContext クラスインスタンスを関連付けます。

  ネイティブ実装が ActionScript 側にイベントを送出するときに、この FREContext 値を指定します。ランタイムは、FREContext 値を使用して、ExtensionContext インスタンスに対応するイベントを送出し ます。94 ページの「FREDispatchStatusEventAsync()」を参照してください。

  また、ネイティブ関数はこの値を使用して、コンテキスト固有のネイティブデータおよびコンテキスト固有の ActionScript データにアクセスし、これらのデータを設定できます。

拡張コンテキスト初期化関数は、次の出力パラメーターを設定します。

- ネイティブ関数の配列。ActionScript 側では、ExtensionContext インスタンスの call() メソッドを使用してこれらの関数のそれぞれを呼び出すことができます。

  各配列エレメントの型は FRENamedFunction です。この構造体には、ActionScript 側で関数を呼び出すために使用する名前となる文字列が含まれます。また、この構造体には、ユーザー記述の C 関数へのポインターも含まれます。ランタイムが名前と C 関数を関連付けます。名前の文字列が実際の関数名と一致する必要はありませんが、通常は同じ名前を使用します。

- ネイティブ関数の配列内にある関数の数。

FREContextInitializer() 関数を呼び出す AIR ランタイムを示すシーケンス図については、15 ページの「拡張の初期化」を参照してください。

関連項目

Vibration ネイティブ拡張の例

コンテキスト固有データ

コンテキスト固有データは、拡張コンテキストに固有のデータです（前述のとおり、拡張データは、拡張のすべての拡張コンテキストに対するデータです）。コンテキスト初期化メソッド、コンテキストファイナライズ処理メソッドおよびネイティブ拡張メソッドはコンテキスト固有データを作成し、データにアクセスし、データを変更することができます。

コンテキスト固有データには次のデータを含めることができます。

- ネイティブデータ。このデータはユーザーが選択する任意のデータです。単純なプリミティブデータ型、またはユーザー定義の構造体を使用できます。97 ページの「FREGetContextNativeData()」および 112 ページの「FRESetContextNativeData()」を参照してください。
- ActionScript データ。このデータは FREObject 変数です。FREObject 変数は ActionScript クラスオブジェクトに対応するので、このデータによって ActionScript オブジェクトを保存し、後でそれにアクセスすることができます。97 ページの「FREGetContextActionScriptData()」および 112 ページの「FRESetContextActionScriptData()」を参照してください。また、21 ページの「FREObject 型」も参照してください。

コンテキスト固有ネイティブデータを設定するネイティブ実装を示すシーケンス図については、15 ページの「拡張の初期化」を参照してください。コンテキスト固有データを取得するネイティブ実装を示すシーケンス図については、19 ページの「拡張関数」を参照してください。

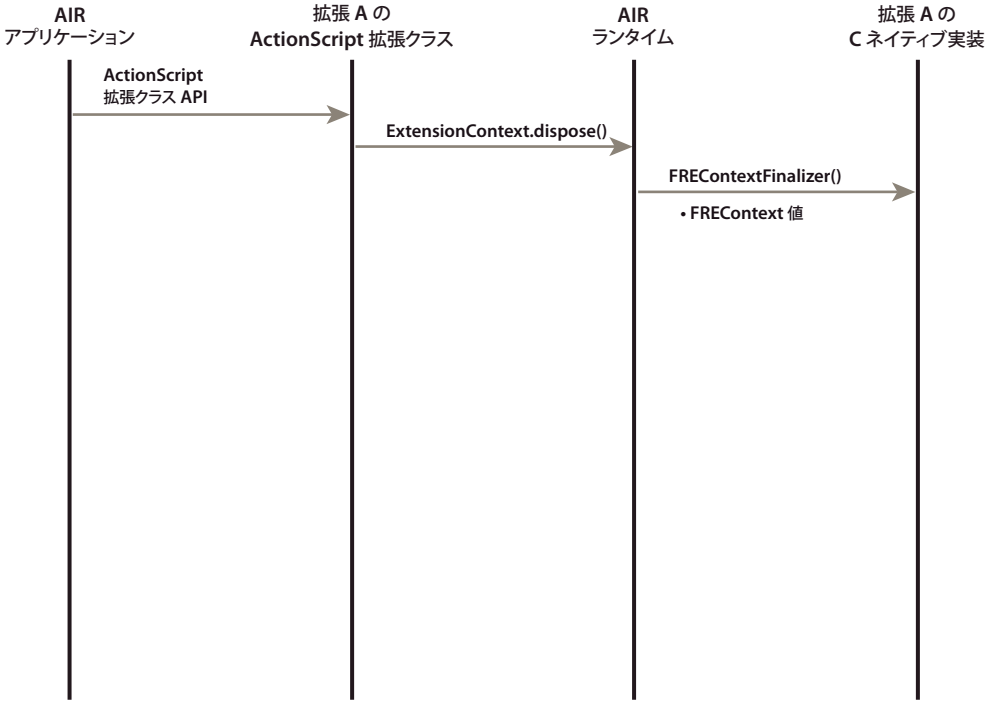
拡張コンテキストのファイナライズ処理

拡張の ActionScript 側では、ExtensionContext インスタンスの dispose() メソッドを呼び出すことができます。dispose() を呼び出すと、ランタイムは拡張のコンテキストファイナライズ処理関数を呼び出します。FREContextFinalizer() のシグネチャを使用して、拡張のコンテキストファイナライズ処理関数を定義します。

このメソッドには 1 つの入力パラメーター（FREContext 値）があります。この FREContext 値を FREGetContextNativeData() および FREGetContextActionScriptData() に渡して、コンテキスト固有データにアクセスできます。このコンテキストに関連付けられたすべてのデータとリソースをクリーンアップします。

ActionScript 側で dispose() を呼び出さない場合は、ランタイムのガベージコレクターが、ExtensionContext インスタンスに対する参照がなくなったときに、そのインスタンスを破棄します。この時点で、ランタイムはユーザーが作成したコンテキストファイナライズ処理関数を呼び出します。

次のシーケンス図に、AIR ランタイムが FREContextFinalizer() 関数を呼び出す様子を示します。



拡張コンテキストのファイナライズ処理シーケンス

拡張のファイナライズ処理

C API では、拡張をアンロードするときにランタイムが呼び出すための、拡張ファイナライズ処理関数が提供されます。ただし、ランタイムが拡張をアンロードするとは限りません。そのため、ランタイムは必ずしも拡張ファイナライズ処理関数を呼び出すわけではありません。

FREFinalizer() のシグネチャを使用して、拡張のコンテキストファイナライズ処理関数を定義します。このメソッドには 1 つの入力パラメーターがあります。それは、拡張初期化関数で作成された拡張データです。この拡張に関連付けられたすべてのデータとリソースをクリーンアップします。

アプリケーションバンドル拡張では、FREFinalizer() の実装は任意の名前に設定できます。拡張記述ファイル内にファイナライズ処理関数の名前を指定します。71 ページの「ネイティブ拡張記述ファイル」を参照してください。

デバイスバンドルアプリケーションでは、拡張ファイナライズ関数の指定方法はデバイスによって異なります。

拡張関数

ActionScript 側で、ExtensionContext インスタンスの call() メソッドを呼び出すことで、ユーザー実装の C 関数を呼び出します。call() メソッドには、次のパラメーターがあります。

- 関数名。この名前は、コンテキスト初期化関数の出力パラメーターに設定したものです。この名前は、ActionScript 側とネイティブ側の間で合意した、任意の文字列です。通常、実際のネイティブ C 関数名と同じ名前です。ただし、ランタイムが任意の名前と実際の関数を関連付けるため、これらの名前は異なる場合があります。

- ネイティブ関数の引数のリスト。これらの引数には、ActionScript オブジェクト（プリミティブ型または ActionScript クラスオブジェクト）を指定できます。

同じ関数シグネチャの FREFunction() を使用して、各ネイティブ関数を定義します。ランタイムは各ネイティブ関数に次のパラメーターを渡します。

- FREContext 値。ネイティブ関数はこの値を使用して、コンテキスト固有データにアクセスし、そのデータを設定できます。また、ネイティブ実装は FREContext 値を使用して、ActionScript 側に非同期イベントを送出します。
- 関数に関連付けられたデータへのポインター。このデータは任意のネイティブデータです。ランタイムがネイティブ関数を呼び出すときに、関数にこのデータポインターを渡します。
- 関数パラメーター数。
- 関数パラメーター。各関数パラメーターは、FREObject 型です。これらのパラメーターは ActionScript クラスオブジェクトまたはプリミティブデータ型に対応します。

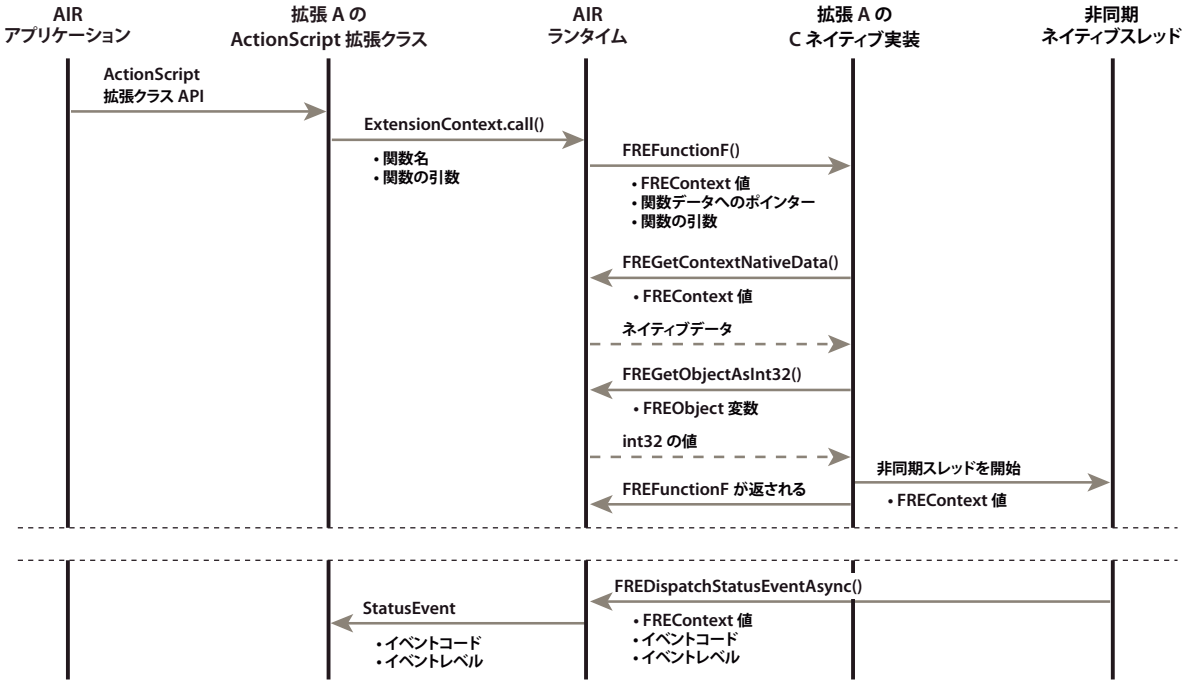
ネイティブ関数は FREObject 型の戻り値も持っています。ランタイムは、対応する ActionScript オブジェクトを、ExtensionContext call() メソッドの戻り値として返します。

注意：ネイティブ関数は非表示にせず、デフォルトの表示を使用してください。

次のシーケンス図に、AIR アプリケーションが関数呼び出しを行った結果、FREFunctionF() という名前のネイティブ C 関数が呼び出される様子を示します。この例では、C 関数は以下を実行します。

- コンテキスト固有ネイティブデータを取得します。
- ActionScript オブジェクトの int32 値を取得します。
- 後でイベントを送出する非同期スレッドを開始します。

注意：C 関数 FREFunctionF() の動作は、呼び出しシーケンスを示すサンプルの動作にすぎません。



ネイティブ関数の呼び出しシーケンスのサンプル

非同期イベントの送出

ネイティブ C コードは、拡張の ActionScript 側に非同期イベントを送出できます。例えば、拡張メソッドから、タスクを実行するための別のスレッドを開始させることができます。他のスレッドのタスクが完了すると、そのスレッドは FREDispatchStatusEventAsync() を呼び出して、拡張の ActionScript 側に通知します。イベントのターゲットは、ActionScript ExtensionContext インスタンスです。

19 ページの「拡張関数」内のシーケンス図では、ネイティブ C 関数が非同期スレッドを開始し、後でイベントを送出する様子を示しています。

関連項目

94 ページの「FREDispatchStatusEventAsync()」

FREObject 型

FREObject 型の変数は、ActionScript クラスオブジェクトまたはプリミティブ型に対応するオブジェクトを表します。ネイティブ実装で FREObject 変数を使用して ActionScript データを操作します。FREObject 型は主に、ネイティブ関数のパラメーターおよび戻り値で使用します。

ネイティブ関数を記述する際には、ユーザーがパラメーターの順序を決定します。また、ユーザーが ActionScript 側も記述するので、ExtensionContext インスタンスの call() メソッドで、そのパラメーター順序を使用します。そのため、すべてのネイティブ関数のパラメーターは FREObject 変数ですが、対応する ActionScript 型を把握できます。

同様に、ネイティブ関数の戻り値がある場合は、ユーザーがその戻り値の ActionScript 型を決定します。call() メソッドはこの型のオブジェクトを返します。ネイティブ関数の戻り値は常に FREObject 変数ですが、対応する ActionScript 型を把握できます。

拡張 C API では、FREObject 変数によって参照されるオブジェクトを使用するための関数が提供されます。これらのオブジェクトは ActionScript データに対応するので、これらの C API 関数を使用して ActionScript クラスオブジェクトまたはプリミティブデータ変数にアクセスできます。使用する C API は ActionScript オブジェクトの型によって異なります。この型は次のとおりです。

- ActionScript プリミティブデータ型
- ActionScript クラスオブジェクト
- ActionScript String オブジェクト
- ActionScript Array または Vector クラスオブジェクト
- ActionScript ByteArray クラスオブジェクト
- ActionScript BitmapData クラスオブジェクト

注意：FREFunction 関数が実行されているスレッドと同じスレッドからのみ、拡張 C API を呼び出すことができます。この唯一の例外となるのは、C API が ActionScript 側にイベントを送出する場合です。その関数 FREDispatchStatusEventAsync() は、どのスレッドからも呼び出すことができます。

FREObject 変数の型の判断

FREObject 変数が対応する ActionScript Object の型がわからない場合もあります。型を判断するには、C API 関数 FREGetObjectType() を使用します。

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

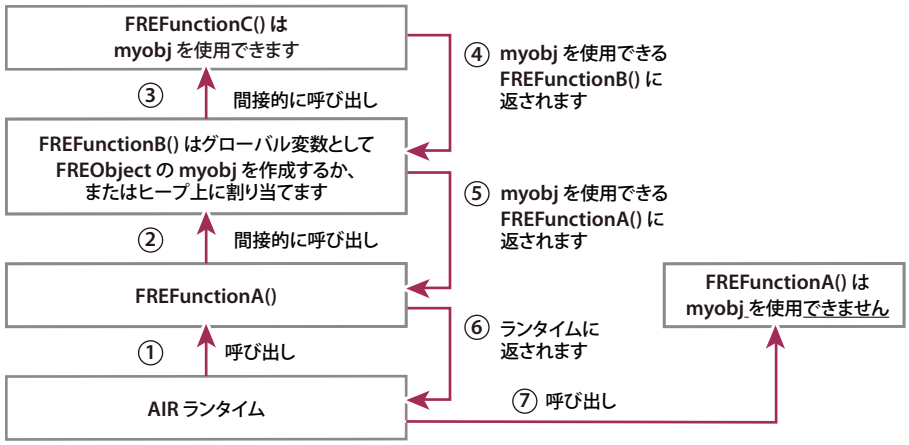
型が判明したら、適切な C API を使用して、その値を操作します。例えば、型が FRE_TYPE_VECTOR の場合、26 ページの「ActionScript Array および Vector オブジェクトの操作」の説明にある C API を使用して Vector オブジェクトを操作します。

FREObject の有効性

C API 呼び出しで無効な FREObject 変数の使用を試みると、C API は FRE_INVALID_OBJECT 戻り値を返します。

FREObject 変数は、コールスタック上の最初の FREFunction 関数が戻るまでの間のみ有効です。コールスタック関数上の最初の FREFunction 関数は、ActionScript 側で ExtensionContext インスタンスの call() メソッドを呼び出した結果、ランタイムが呼び出す関数です。

次の図に、この動作を示します。



コールスタック上での FREObject の有効性

注意：FREFunction 関数は間接的に他の FREFunction 関数を呼び出すことができます。例えば、FREFunctionA() は ActionScript オブジェクトのメソッドを呼び出すことができます。そのメソッドは次に FREFunctionB() を呼び出すことができます。

そのため、FREObject 変数を使用する場合は、次のことを考慮します。

- FREFunction 関数に渡された FREObject 変数は、コールスタック上の最初の FREFunction 関数が戻るまでの間のみ有効です。
- ネイティブ関数が拡張 C API を使用して作成する FREObject 変数は、コールスタック上の最初の FREFunction 関数が戻るまでの間のみ有効です。
- 他のスレッドでは FREObject 変数を使用できません。FREObject 変数は、変数を受け取った、または作成したネイティブ関数のスレッドと同じスレッド内でのみ使用してください。
- 複数の FREFunction 関数呼び出しをまたがって（例えばグローバルデータ内で）FREObject 変数を保存することはできません。コールスタック上の最初の FREFunction 関数が戻ると変数は無効となるので、保存された変数は使用できません。ただし、FRESetContextActionScriptData() メソッドを使用して、対応する ActionScript オブジェクトを保存することができます。
- FREObject 変数が無効になった後でも、対応する ActionScript オブジェクトは存続できます。例えば、FREObject 変数が FREFunction 関数の戻り値の場合、対応する ActionScript オブジェクトは参照されたままです。ただし、ActionScript 側でその参照が削除された場合は、ランタイムによって ActionScript オブジェクトが破棄されます。
- 複数の拡張で FREObject 変数を共有することはできません。

注意：FREObject 変数は、同じ拡張の複数の拡張コンテキスト間で共有できます。ただし、どのような場合でも、コールスタック上の最初の FREFunction 関数がランタイムに戻されると、FREObject 変数は無効になります。

ActionScript プリミティブ型およびオブジェクトの操作

ActionScript プリミティブ型の操作

ネイティブ関数の入力パラメーターは、ActionScript プリミティブ型に対応させることができます。すべてのネイティブ関数パラメーターは FREObject 型です。そのため、ActionScript プリミティブ型入力パラメーターを操作するには、FREObject パラメーターの ActionScript 値を取得します。その値を、対応する C プリミティブデータ型変数に保存します。次の C API 関数を使用します。

- 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);
```

出力パラメーターまたは戻り値が ActionScript プリミティブ型に対応する場合は、C API 関数を使用して ActionScript プリミティブを作成します。FREObject 変数および C データ変数のプリミティブの値へのポインターを指定します。ランタイムは ActionScript プリミティブを作成し、FREObject 変数を、そのプリミティブに対応するように設定します。次の C API 関数を使用します。

- 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);
```

ActionScript String オブジェクトの操作

ネイティブ関数の入力パラメーターは、ActionScript String クラスオブジェクトに対応させることができます。すべてのネイティブ関数パラメーターは FREObject 型です。そのため、ActionScript String パラメーターを操作するには、FREObject パラメーターの ActionScript String 値を取得します。その値を、対応する C 文字列変数に保存します。C API 関数の FREGetObjectAsUTF8() を使用します。

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

FREGetObjectAsUTF8() を呼び出した後は、ActionScript String 値は value パラメーターに設定され、length パラメーターは value 文字列の長さをバイト単位で示します。

出力パラメーターまたは戻り値が ActionScript String クラスオブジェクトに対応する場合は、C API を使用して ActionScript String オブジェクトを作成します。FREObject 変数、文字列値および C 文字列変数内の長さ（バイト単位）へのポインターを指定します。ランタイムは ActionScript String オブジェクトを作成し、FREObject 変数を、そのオブジェクトに対応するように設定します。C API 関数の FRENewObjectFromUTF8() を使用します。

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

value パラメーター文字列では、UTF-8 エンコーディングを使用し、null 終端文字を含める必要があります。

注意：C API 関数の文字列パラメーターでは、UTF-8 エンコーディングを使用し、null 終端文字を含めます。

ActionScript クラスオブジェクトの操作

ネイティブ関数の入力パラメーターは、ActionScript クラスオブジェクトに対応させることができます。すべてのネイティブ関数パラメーターは FREObject 型であるので、C API では、FREObject 変数を使用してクラスオブジェクトを操作するための関数が提供されます。

ActionScript クラスオブジェクトのプロパティを取得および設定するには、次の C API 関数を使用します。

- 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
);
```

ActionScript クラスオブジェクトのメソッドを呼び出すには、次の C API を使用します。

FRECallObjectMethod()

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

出力パラメーターまたは戻り値が ActionScript クラスオブジェクトに対応する場合は、C API を使用して ActionScript オブジェクトを作成します。FREObject 変数へのポインターに加えて、ActionScript クラスコンストラクターのパラメーターに対応する FREObject 変数を指定します。ランタイムは ActionScript クラスオブジェクトを作成し、FREObject 変数を、そのクラスオブジェクトに対応するように設定します。次の C API 関数を使用します。

FRENewObject()

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

注意：これらの一般的な ActionScript オブジェクト操作関数は、すべての ActionScript クラスオブジェクトに適用されます。ただし、ActionScript Array、Vector、ByteArray および BitmapData クラスは、大容量のデータが関係するので特殊なケースとなります。そのため、C API ではこれらの特殊なケースのオブジェクトを操作するための、他の個別の関数が提供されます。

ActionScript ByteArray オブジェクトの操作

拡張の ActionScript 側とネイティブ側の間で大量のバイトデータの受け渡しを効率的に行うには、ActionScript ByteArray クラスを使用します。ネイティブ関数の入力パラメーター、出力パラメーターまたは戻り値は、ActionScript ByteArray クラスオブジェクトに対応させることができます。

他の ActionScript クラスオブジェクトと同様に、FREObject 変数は ActionScript ByteArray オブジェクトをネイティブ側で表現したものです。C API には、FREObject 変数を使用して ByteArray クラスオブジェクトを操作するための関数が用意されています。FRESetObjectProperty()、FREGetObjectProperty() および FRECallObjectMethod() を使用すると、ActionScript ByteArray オブジェクトに対して、プロパティの取得および設定とメソッドの呼び出しができます。

ただし、ByteArray オブジェクト内のバイトデータをネイティブコードから操作するには、C API の FREAcquireByteArray() 関数を使用します。これは、ActionScript 側で作成された ByteArray オブジェクトのバイトデータにアクセスするメソッドです。

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

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

バイトを操作した後で、C API FREReleaseByteArray() を使用します。

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

注意：FREAcquireByteArray() の呼び出しと FREReleaseByteArray() の呼び出しの間では、C API 関数を呼び出さないでください。これが禁止されるのは、他の呼び出しの副作用として、バイト配列コンテンツへのポインターを無効にするコードが実行されるからです。

例

この例では、拡張の ActionScript 側で ByteArray オブジェクトを作成し、その中のバイトデータを初期化します。次に、バイトデータを操作するネイティブ関数を呼び出します。

```
// 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;
}
```

ActionScript Array および Vector オブジェクトの操作

拡張の ActionScript 側とネイティブ側の間で配列を効率的に渡すには、ActionScript Array および Vector クラスを使用します。ネイティブ関数の入力パラメーター、出力パラメーターまたは戻り値は、ActionScript Array または Vector クラスオブジェクトに対応させることができます。

他の ActionScript クラスオブジェクトと同様に、FREObject 変数は ActionScript Array または Vector オブジェクトをネイティブ側で表現したものです。C API では、FREObject 変数を使用して Array または Vector クラスを操作するための関数が提供されます。

次の C API 関数を使用して、Array または Vector オブジェクトの長さを取得および設定します。

- FREGetArrayLength()

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

- FRESetArrayLength()

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

次の C API 関数を使用して、Array または Vector オブジェクトのエレメントを取得および設定します。

- FREGetArrayElementAt()

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

- FRESetArrayElementAt()

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

ActionScript BitmapData オブジェクトの操作

拡張の ActionScript 側とネイティブ側の間でビットマップを渡すには、ActionScript BitmapData クラスを使用します。ネイティブ関数の入力パラメーター、出力パラメーターまたは戻り値は、ActionScript BitmapData クラスオブジェクトに対応させることができます。

他の ActionScript クラスオブジェクトと同様に、FREObject 変数は ActionScript BitmapData オブジェクトをネイティブ側で表現したものです。C API には、FREObject 変数を使用して BitmapData クラスオブジェクトを操作するための関数が用意されています。FRESetObjectProperty()、FREGetObjectProperty() および FRECallObjectMethod() を使用すると、ActionScript ByteArray オブジェクトに対して、プロパティの取得および設定とメソッドの呼び出しができます。

ただし、BitmapData オブジェクト内のビットデータをネイティブコードから操作するには、C API の FREAcquireBitmapData() 関数または FREAcquireBitmapData2() 関数を使用します。これらは、ActionScript 側で作成された BitmapData オブジェクトのビットデータにアクセスするメソッドです。

```
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 構造体または FREBitmapData2 構造体のすべてのフィールドは読み取り専用です。ただし、bits32 フィールドは実際のビットマップ値をポイントします。これはネイティブ実装内で変更できます。FREBitmapData2 構造体および FREAquireBitmapData2 関数は、AIR 3.1 の API に追加されました。FREBitmapData2 には isInvertedY という 1 つのフィールドが追加されています。このフィールドは、画像データの行が保存される順序を示すものです。

ネイティブ実装でビットマップのすべてまたは一部を変更したことを示すには、ビットマップの長方形領域を無効にします。C API 関数の FREInvalidateBitmapDataRect() を使用します。

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

x フィールドおよび y フィールドは、ビットマップの左上隅である 0,0 座標との相対位置として、無効にする長方形領域の座標を示します。width フィールドおよび height フィールドは、無効にする長方形領域の大きさをピクセル単位で示します。

ビットマップを操作した後で、C API 関数 FREReleaseBitmapData() を使用します。

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

注意：FREAcquireBitmapData() の呼び出しと FREReleaseBitmapData() の呼び出しの間で、FREInvalidateBitmapDataRect() を除く C API 関数を呼び出さないでください。これが禁止されるのは、他の呼び出しの副作用として、ビットマップコンテンツへのポインターを無効にするコードが実行されるからです。

スレッドとネイティブ拡張

ネイティブ実装をコーディングするときに、次のことを考慮します。

- ランタイムは、異なる複数のスレッドで、単独の拡張コンテキストの FREFunction 関数を並行して呼び出すことができます。
- ランタイムは、異なる複数のスレッドで、異なる複数の拡張コンテキストの FREFunction 関数を並行して呼び出すことができます。

そのため、ネイティブ実装を適切にコーディングします。例えば、グローバルデータを使用する場合は何らかのロック機能を使用して保護します。

注意：ネイティブ実装では異なるスレッドを作成することもできます。その場合、22 ページの「FREObject の有効性」で説明している制限を考慮してください。

第 4 章: Java を使用したネイティブ側のコーディング

Android デバイスでは、アプリケーション開発言語として主に Java が使用されています。ネイティブ拡張の開発ターゲットが Android の場合は、拡張の「ネイティブ」側のコーディングにネイティブ拡張 Java API を使用します。また、目的によっては、AIR 拡張 C API と Android Native Development Kit を使用することもできます。C API の使用について詳しくは、15 ページの「C を使用したネイティブ側のコーディング」を参照してください。

Java API は、FlashRuntimeExtensions.jar ファイルに含まれています。このファイルは AIR SDK の lib/android ディレクトリにあります。AIR SDK は、http://www.adobe.com/jp/special/products/air/sdk/ で取得できます。

Adobe AIR 用ネイティブ拡張の Java 側のコーディングは、次のようにして行います。

- FREExtension インターフェイスを実装します。
- FREContext 抽象クラスを拡張して、1 つまたは複数の具象サブクラスを作成します。
- 拡張の ActionScript 側から呼び出せるすべての Java 関数に、FREFunction インターフェイスを実装します。

パラメーターや戻り値など、Java API 関数について詳しくは、115 ページの「Android Java API リファレンス」を参照してください。

FREExtension インターフェイスの実装

Java API を使用する拡張では、FREExtension インターフェイスを実装する必要があります。この FREExtension インスタンスは、拡張内の Java コードに対する初期エントリポイントとなります。拡張記述ファイルの <initializer> エレメントに、クラスの完全修飾名を指定します。Java による実装は、Android-ARM プラットフォームでのみ使用できます。71 ページの「ネイティブ拡張記述ファイル」を参照してください。

createContext() メソッドは、FREExtension の実装内で最も重要な部分です。拡張の ActionScript コードから ExtensionContext.createExtensionContext() が呼び出されると、AIR ランタイムによって createContext() メソッドが呼び出されます。このメソッドは FREContext クラスのインスタンスを返す必要があります。ActionScript createExtensionContext() メソッドには 1 個の文字列パラメーターがあります。このパラメーターは Java createContext() 関数に渡されます。この値を使用すると、目的に応じて動作のコンテキストを変化させることができます。コンテキストクラスを 1 つしか使用しない拡張の場合、このパラメーターは無視してかまいません。

FREExtension インターフェイスに備わっている他のメソッドとしては initialize() および dispose() があります。これらはランタイムによって自動的に呼び出されるメソッドであり、その拡張で必要となる永続的なリソースの作成やクリーンアップを実行するために使用できます。拡張によっては、これらの関数で処理を実行する必要がまったくない場合もあります。

FREExtension を実装するクラスのコンストラクターは、パラメーターを 1 つも受け取ることができません。

FREExtension インスタンスは、ActionScript コードで createExtensionContext() が最初に呼び出されたときに AIR ランタイムによってインスタンス化されます。Java 拡張のクラスに対する呼び出しは次の流れに従って実行されます。

- FREExtension を実装するクラスのコンストラクター
- initialize()
- createContext()

FREExtension の例

次に示す例は、FREExtension の単純な実装です。この例では拡張コンテキストを 1 つだけ使用しています。コンテキストに対する参照は、AIR ランタイムによって createContext() メソッドが最初に呼び出されたときに作成されます。この参照は以後使用するために保持されます。

```
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 の例

次に示す例は、ネイティブ拡張の機能をアプリケーションコードに対して提供するような拡張の ActionScript コードです。

```
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;
    }
}
}
```

アプリケーションコードの例

ネイティブ拡張を使用するアプリケーションは、拡張によって提供される ActionScript のクラスおよびメソッドにアクセスします（アクセスの方法は、一般の ActionScript ライブラリで提供されるクラスやメソッドにアクセスする場合と同じです）。

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

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

FREContext クラスの拡張

FREContext オブジェクトは、Java 関数のセットと、コンテキストに特化した状態を提供します。拡張は、FREContext クラスの具象サブクラスを少なくとも 1 つ実装する必要があります。

FREContext クラスには、実装しなくてはならない 2 つの抽象メソッドが定義されています。

- getFunctions() — Map オブジェクトを返す必要があります。この Map オブジェクトは、AIR ランタイムで、コンテキストにどのような関数が提供されているかを調べるときに使用されます。
- dispose() — コンテキストリソースのクリーンアップが可能になったとき、ランタイムによって呼び出されます。

コンテキストの破棄

拡張の ActionScript 側では、ExtensionContext インスタンスの dispose() メソッドを呼び出すことができます。ActionScript の dispose() メソッドを呼び出すと、ランタイムによって Java の FREContext クラスの dispose() メソッドが呼び出されます。

ActionScript 側で dispose() を呼び出さない場合は、ランタイムのガベージコレクターが、ExtensionContext インスタンスに対する参照がなくなったときに、そのインスタンスを破棄します。その時点で、ランタイムによって Java の FREContext クラスの dispose() メソッドが呼び出されます。

FREContext の例

次に示す例は、FREContext の単純な実装です。クラス内で、関数 1 つを含んだ関数マップを作成しています。

```
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;
    }
}
```

FREFunction インターフェイスの実装

FREFunction インターフェイスは、ActionScript コードから Java 関数を呼び出すときに AIR ランタイムによって使用されます。このインターフェイスを実装し、機能の具象部分を提供する必要があります。

FREFunction インターフェイスに定義されているメソッドは call() の 1 つだけです。このメソッドは、拡張内の ActionScript コードから ExtensionContext クラスの call() メソッドが呼び出されたときに、AIR ランタイムによって呼び出されます。ランタイムは、FREContext 関数マップ内を検索して適切な FREFunction インスタンスを見つけます。ActionScript 側のコードでは、ActionScript の通常の型やクラスで表現した引数の配列をメソッドに渡します。それらの値は、Java 側のコードには FREObject インスタンスとして渡されます。同じように、Java call() メソッドは FREObject インスタンスを返し、その戻り値は ActionScript コードには何らかの ActionScript 型として渡されます。

FREFunction クラスを実装するときは、call() メソッドの引数配列に含めるパラメーターの順序を決定します。また、ActionScript 側のコードを作成するときは、それと同じ順序に並べたパラメーターを ExtensionContext インスタンスの call() メソッドに渡すようにします。つまり、Java 関数のパラメーターはすべて FREObject 型ですが、それぞれがどの ActionScript 型に対応するかを決めておく必要があります。

同じように、Java 関数が戻り値を返す場合も、どの ActionScript 型に対応するかを決めておく必要があります。call() メソッドはこの型のオブジェクトを返します。Java 関数の戻り値は常に FREObject インスタンスですが、どの ActionScript 型に対応するかを決めておく必要があります。

作成される FREObject インスタンスや、FREFunction オブジェクトに渡される FREObject インスタンスの存続期間には限りがあります。FREObject への参照を保存しておいて次回以降の関数呼び出し時に使用することはできません。

FREFunction の例

次に示す例は、FREFunction の単純な実装です。この関数はブール型のパラメーターを 1 つ受け取り、その値を反転して返します。

```
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;
    }
}
```

非同期イベントの送出

Java コードでは、拡張の ActionScript 側コードに対して非同期イベントを送出できます。例えば、拡張メソッドから、タスクを実行するための別のスレッドを開始させることができます。他のスレッドのタスクが完了したときには、そのスレッドから、該当する FREContext の dispatchStatusEventAsync() メソッドが呼び出されます。ActionScript 側では、Java FREContext に関連付けられている ActionScript ExtensionContext インスタンスから Status イベントが送出されます。

ActionScript オブジェクトに対するアクセス

FREObject クラスのインスタンスは、何らかの ActionScript クラスオブジェクトまたはプリミティブ型を表します。Java 実装内で FREObject インスタンスを使用すると、ActionScript データを操作できます。FREObject の主な使用目的は、Java 関数のパラメーターおよび戻り値の受け渡しです。

FREObject クラスには、関連付けられた ActionScript クラスオブジェクトまたはプリミティブ値にアクセスするための関数があります。使用する Java API は ActionScript オブジェクトの型によって異なります。使用できる型の種類は次のとおりです。

- ActionScript プリミティブデータ型
- ActionScript クラスオブジェクト
- ActionScript String オブジェクト
- ActionScript Array または Vector クラスオブジェクト
- ActionScript ByteArray クラスオブジェクト
- ActionScript BitmapData クラスオブジェクト

重要：FREObject にアクセスできるのは、それを「所有」する FREFunction 関数が動作しているのと同じスレッドのみです。なお、何らかの FREByteArray または FREBitmapData オブジェクトに対して acquire() メソッドを呼び出した後は、その FREByteArray または FREBitmapData オブジェクトに対して release() メソッドを呼び出すまでの間、どの ActionScript オブジェクトのメソッドにもアクセスできません。オブジェクトを acquire() でロックしている間、そのオブジェクトや他の FREObject に対して getProperty()、setProperty() または callMethod() を呼び出すと IllegalStateException がスローされます。

FREObject の有効性

無効な FREObject を Java API 呼び出しで使用すると、Java API から例外がスローされます。

FREObject インスタンスは、コールスタック上の最初の FREFunction 関数が戻るまでの間のみ有効です。コールスタック上の最初の FREFunction 関数は、ActionScript 側で ExtensionContext インスタンスの call() メソッドが呼び出された結果として、ランタイムから呼び出された関数です。また、FREObject は、ランタイムによる最初の FREFunction 呼び出しに使用されたスレッド内でのみ有効です。

注意：FREFunction 関数は間接的に他の FREFunction 関数を呼び出すことができます。例えば、FREFunctionA() は ActionScript オブジェクトのメソッドを呼び出すことができます。そのメソッドは次に FREFunctionB() を呼び出すことができます。

したがって、FREObject を使用するときは次の点を考慮する必要があります。

- FREFunction インスタンスに渡された FREObject は、コールスタック上の最初の FREFunction 関数が戻るまでの間のみ有効です。
- Java 関数で作成された FREObject は、コールスタック上の最初の FREFunction 関数が戻るまでの間のみ有効です。
- FREObject を他のスレッドで使用することはできません。FREObject は、そのオブジェクトを受け取った、または作成した Java 関数のスレッドと同じスレッド内でのみ使用できます。
- FREObject への参照をグローバルデータなどに保存しておいて、次回以降の FREFunction 関数呼び出し時に使用することはできません。コールスタック上の最初の FREFunction 関数が戻るとオブジェクトが無効になるので、保存した参照は使用不可になります。ただし、FREContext クラスの setActionScriptData() メソッドを使用すると、次回以降の関数呼び出し時までデータを保存しておくことができます。

- FREObject が無効になった後も、それに対応する ActionScript オブジェクトは存続します。例えば、FREObject が FREFunction 関数の戻り値の場合、対応する ActionScript オブジェクトは参照されたままです。ただし、ActionScript 側でその参照が削除された場合は、ランタイムによって ActionScript オブジェクトが破棄されます。
- 複数の拡張の間で FREObject を共有することはできません。

  注意：1 つの拡張内では、複数の拡張コンテキスト間で FREObject を共有できます。ただし、この場合も他の場合と同じく、コールスタック上の最初の FREFunction 関数がランタイムに処理を戻すと FREObject は無効になります。

ActionScript プリミティブ型およびオブジェクトの操作

ActionScript プリミティブ型の操作

Java の FREFunction 実装内では、入力パラメーターを何らかの ActionScript プリミティブ型と対応させることができます。ActionScript コードから拡張に渡されたすべてのパラメーターは、FREObject 型になります。そのため、ActionScript プリミティブ型入力パラメーターを操作するには、FREObject パラメーターの ActionScript 値を取得します。次の FREObject 関数を使用します。

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

FREFunction の出力パラメーターまたは戻り値が ActionScript プリミティブ型に対応している場合は、FREObject ファクトリメソッドのいずれかを使用して ActionScript プリミティブを作成します。次の静的 FREObject 関数を使用します。

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

ActionScript クラスオブジェクトの操作

FREFunction の実装内では、入力パラメーターを何らかの ActionScript クラスオブジェクトと対応させることができます。FREObject クラスには、オブジェクトに含まれる ActionScript で定義されたプロパティやメソッドにアクセスするための関数があります。

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

出力パラメーターまたは戻り値が ActionScript クラスオブジェクトに対応する場合は、静的な FREObject ファクトリメソッドを使用して ActionScript オブジェクトを作成します。

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

注意：これらの一般的な ActionScript オブジェクト操作関数は、すべての ActionScript クラスオブジェクトに適用されます。ただし、ActionScript Array、Vector、ByteArray および BitmapData クラスは、大容量のデータが関係するので特殊なケースとなります。そのため、Java API には、これらに該当する特殊なオブジェクトを操作するためのクラスが別途用意されています。

ActionScript ByteArray オブジェクトの操作

拡張の ActionScript 側と Java 側の間で大量のバイトデータの受け渡しを効率的に行うには、ActionScript ByteArray クラスを使用します。FREByteArray クラスは、FREObject を拡張したクラスであり、バイト配列オブジェクトを扱うためのメソッドを備えています。

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

ByteArray オブジェクト内のバイトデータを Java コードから操作するには、FREByteArray acquire() メソッドを呼び出した後に getBytes() メソッドを使用します。バイトデータの操作が完了した後には、FREByteArray release() メソッドを呼び出します。

acquire() を呼び出してから release() を呼び出すまでの間、FREObject のメソッドを呼び出すことは一切できません（acquire() でロックしたオブジェクトだけでなく、他の FREObject オブジェクトのメソッドも呼び出せません）。これが禁止されているのは、他の呼び出しを実行すると副作用でバイト配列オブジェクトやその内容が無効になってしまう可能性があるからです。

ロックした FREByteArray オブジェクトの bytes フィールドには、length フィールドに示されている長さを超えるサイズのバイトデータを書き込むことはできません。余分なバイトデータは ActionScript 側で無視されます。

静的ファクトリメソッドの FREByteArray.newByteArray() を使用すると、FREByteArray オブジェクトを新規作成できます。新しいバイト配列の内容は空です。この配列に Java からデータを追加する場合は、前もって ActionScript length プロパティを設定する必要があります。

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

ActionScript Array および Vector オブジェクトの操作

拡張の ActionScript 側と Java 側の間で配列の受け渡しを効率的に行うには、ActionScript の Vector クラスと Array クラスを使用します。

FREArray クラスは、FREObject を拡張したクラスであり、配列および Vector オブジェクトの扱いに適した次のメソッドを備えています。

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

FREArray には、配列および Vector を作成する次のファクトリメソッドも定義されています。

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

ActionScript BitmapData オブジェクトの操作

拡張の ActionScript 側と Java 側の間でビットマップの受け渡しを行うには、ActionScript BitmapData クラスを使用します。FREBitmapData クラスは、FREObject を拡張したクラスであり、ビットマップデータオブジェクトの扱いに適した次のメソッドを備えています。

- acquire()

  次に示すメソッドを呼び出す場合は、前もって acquire() を呼び出しておく必要があります。

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

ビットマップデータに変更を加えるには、acquire() を呼び出した後、getBits() メソッドでピクセルカラーデータを取得します。getLineStride32() メソッドは、水平ライン 1 本あたりに含まれる 32 bit データの数を示します。通常、この値はビットマップの幅と一致しますが、場合によっては、表示されない余分なバイトデータがビットマップデータに余白として付加されることがあります。ビットマップ内の各ピクセルは、ARGB 形式の 32 bit 値で表現されます。ビットマップを操作した後は、release() を呼び出してビットマップデータを解放します。

acquire() を呼び出してから release() を呼び出すまでの間、FREObject のメソッドを呼び出すことは一切できません（acquire() でロックしたオブジェクトだけでなく、他の FREObject オブジェクトのメソッドも呼び出せません）。これが禁止されているのは、他の呼び出しを実行すると副作用でバイト配列オブジェクトやその内容が無効になってしまう可能性があるからです。

Java 実装からビットマップの全体または一部に変更を加えたことを示すには、invalidateRect() 関数を呼び出して、現在の表示を無効にする長方形領域を指定します。x および y パラメーターは、現在の表示を無効にする長方形領域の座標を、ビットマップの左上隅を表す 0,0 座標との相対位置で示します。width フィールドおよび height フィールドは、現在の表示を無効にする長方形領域の大きさをピクセル単位で示します。

静的ファクトリメソッドの FREBitmapData.newBitmapData() を使用すると、FREBitmapData オブジェクトを新規作成できます。

スレッドとネイティブ拡張

Java 実装をコーディングするときは、次の点を考慮する必要があります。

- ランタイムは、異なる複数のスレッドで、単独の拡張コンテキストの FREFunction 関数を並行して呼び出すことができます。
- ランタイムは、異なる複数のスレッドで、異なる複数の拡張コンテキストの FREFunction 関数を並行して呼び出すことができます。

したがって、Java 実装は適切にコーディングする必要があります。例えば、グローバルデータを使用する場合は何らかのロック機能を使用して保護します。

注意：Java 実装では異なるスレッドを作成することもできます。その場合は、34 ページの「FREObject の有効性」で説明している制限を考慮してください。

第 5 章：ネイティブ拡張のパッケージ化

ネイティブ拡張をアプリケーション開発者に提供するには、関連するすべてのファイルを 1 つの ANE ファイルにパッケージ化します。AIR アプリケーション開発者は次の方法で ANE ファイルを使用します。

- SWC ファイルをライブラリパスに含める場合と同じ方法で、ANE ファイルをアプリケーションのライブラリパスに含めます。この操作を行うことで、アプリケーションは拡張の ActionScript クラスを参照できるようになります。
- ANE ファイルを AIR アプリケーションと共にパッケージ化します。

ANE ファイルを構築して AIR アプリケーションと共にパッケージ化する方法について詳しくは、「Adobe AIR 用ネイティブ拡張の使用」を参照してください。

ネイティブ拡張のパッケージを作成するには、次のタスクを実行します。

1. 拡張の ActionScript ライブラリを SWC ファイル内に構築します。
2. 拡張のネイティブライブラリを、サポートされているターゲットプラットフォームごとに 1 つずつ構築します。
3. 拡張の署名済み証明書を作成します。拡張への署名はオプションです。
4. 拡張記述ファイルを作成します。
5. ADT を使用して拡張パッケージを作成します。

ネイティブ拡張の ActionScript ライブラリの構築

拡張の ActionScript 側を SWC ファイル内に構築します。この SWC ファイルは ActionScript ライブラリです。つまり、ActionScript クラスおよびその他のリソース（画像や文字列など）を含むアーカイブファイルです。

ネイティブ拡張をパッケージ化する際には、SWC ファイルと個別の library.swf ファイルの両方が必要になります。library.swf ファイルは SWC ファイルから抽出します。SWC ファイルには、オーサリングとコンパイルのための ActionScript 定義が含まれています。library.swf には、特定のプラットフォームで使用される ActionScript 実装が含まれています。拡張のターゲットプラットフォームごとに別々の ActionScript 実装が必要な場合は、複数の SWC ライブラリを作成し、プラットフォームごとに個別に library.swf を抽出します。ただし、ベストプラクティスは、すべての ActionScript 実装で同じパブリックインターフェイスを使用することです（ANE パッケージに含めることのできる SWC ファイルは 1 つだけです）。

SWC ファイルには、library.swf というファイルが含まれています。詳しくは、49 ページの「ANE パッケージ内の SWC ファイルと SWF ファイル」を参照してください。

次のいずれかの方法を使用して SWC ファイルを構築します。

- Adobe Flash Builder を使用して Flex ライブラリプロジェクトを作成する。

  Flex ライブラリプロジェクトを作成すると、Flash Builder によって SWC ファイルが作成されます。「Flex ライブラリプロジェクトの作成」を参照してください。

  Flex ライブラリプロジェクトを作成するときは、Adobe AIR ライブラリを含めるオプションを必ず選択してください。

  SWC は適切なバージョンの SWF 形式にコンパイルしてください。AIR 2.7 は SWF 11、AIR 3 は SWF 13、AIR 3.1 は SWF 14 などのように使用してください。SWF ファイル形式のバージョンは、プロジェクトのプロパティで設定できます。ActionScript コンパイラーを選択して、次のような追加のコンパイラー引数を入力します。

  ```
  -swf-version 17
  ```

注意： Flex SDK bin ディレクトリにある swfdump を次のように使用して、SWF ファイルの SWF ファイル形式のバージョンを確認できます。swfdump myFlexLibraryProjectSWF.swf

- AIR の Flex ライブラリプロジェクトを作成するには、acompc コマンドラインツールを使用します。このツールは、Flex SDK で提供されるコンポーネントコンパイラーです。Flash Builder を使用していない場合は、acompc を直接使用します。「Using compc, the component compiler」を参照してください。

  次に、例を示します。

```
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
```

注意：ActionScript ライブラリで外部リソースを使用している場合は、ADT を使用して、外部リソースを ANE ファイルにパッケージ化します。46 ページの「ネイティブ拡張パッケージの作成」を参照してください。

SWF バージョンの互換性

ActionScript ライブラリのコンパイル時に指定される SWF のバージョンは、拡張記述子の名前空間と同様、拡張に AIR アプリケーションとの互換性があるかどうかを決定する要素の 1 つになります。メインアプリケーションの SWF ファイルよりも上のバージョンの SWF を拡張で使用することはできません。

互換性のある AIR アプリケーションバージョン	ANE SWF バージョン	拡張の名前空間
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

注意：プラットフォームオプション（platform.xml）ファイルには ns.adobe.com/air/extension/3.1 以降の名前空間が必要です。-platformoptions フラグを使用して ANE をパッケージ化する場合は、ns.adobe.com/air/extension/3.1 以降と、14 以降のバージョンの SWC を指定する必要があります。プラットフォームオプションファイル機能によっては、以降のバージョンの AIR の名前空間と SWF が必要な場合があります。

ネイティブ拡張の署名済み証明書の作成

ネイティブ拡張に電子署名することができます。拡張への署名はオプションです。

承認された証明機関（CA）が発行した証明書を使用してネイティブ拡張に電子署名すると、AIR アプリケーション開発者に対して、次のことが確実に保証されます。

- アプリケーションと共にパッケージ化しようとしているネイティブ拡張が、誤って、または悪意によって改ざんされていないこと。
- 作成者自身がネイティブ拡張の署名者（発行者）であること。

注意：AIR アプリケーションがパッケージ化されると、ネイティブ拡張の証明書（提供されている場合）ではなく、AIR アプリケーションの証明書がアプリケーションのユーザーに表示されます。

証明書は認証期間から入手してください。ネイティブ拡張の電子署名済み証明書を作成することは、AIR アプリケーションの証明書を作成することと同じです。『Adobe® AIR® アプリケーションの構築』の「AIR アプリケーションへの署名」を参照してください。

拡張記述ファイルの作成

各ネイティブ拡張には、拡張記述ファイルが含まれています。この XML ファイルでは、拡張識別子、名前、バージョン番号、実行可能なプラットフォームなど、拡張に関する情報が指定されます。

拡張を作成するときは、71 ページの「ネイティブ拡張記述ファイル」の詳細なスキーマに従って拡張記述ファイルを記述します。

次に、例を示します。

```
<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>
```

拡張記述ファイルを作成するときは、次の情報を考慮します。

拡張 ID

<id> エレメントの値は、以下で使用されている値と同じです。

- CreateExtensionContext() に対する ActionScript の呼び出し。
- 拡張を使用するアプリケーションのアプリケーション記述ファイル内にある extensionID エレメント。

拡張 ID の名前を設定する場合のベストプラクティスについては、9 ページの「拡張 ID」を参照してください。

バージョン番号

<versionNumber> エレメントの値では、拡張のバージョンを指定します。バージョン番号の重要な用途の 1 つは、デバイスバンドル拡張の下位互換性を維持することです。13 ページの「ネイティブ拡張の下位互換性」を参照してください。

プラットフォーム

5 ページの「複数プラットフォームのターゲット」で説明しているように、複数のプラットフォームをターゲットとするネイティブ拡張を作成できます。

5 ページの「ランタイムでの拡張の利用」で説明しているように、プラットフォームに応じて、拡張はアプリケーションバンドルまたはデバイスバンドルになります。

ターゲットとなるプラットフォームごとに、拡張記述ファイルで <platform> エレメントを指定します。<platform> エレメントの name 属性では、iPhone-ARM や Windows-x86 など、ターゲットのプラットフォームを指定します。アプリケーションバンドル拡張では、name 属性の値として default も指定できます。この値は、拡張が ActionScript 専用であり、ネイティブコードライブラリが含まれていないことを示します。

アプリケーションバンドル拡張を使用する AIR アプリケーションを実行すると、AIR では次の処理が実行されます。

- 拡張記述ファイルによってデバイスのプラットフォームに対応するプラットフォーム名に関連付けられている拡張ライブラリがロードされます。
- デバイスに対応するプラットフォーム名がない場合は、拡張記述ファイルによってデフォルトのプラットフォームに関連付けられている拡張ライブラリがロードされます。

記述子の名前空間

記述ファイルのルート <extension> エレメントで指定される名前空間によって、拡張で必要な AIR SDK バージョンが決定されます。SWF のバージョン同様、名前空間は、拡張が AIR アプリケーションで使用できるかどうかを決定する要素の 1 つです。AIR アプリケーション記述子の名前空間は、拡張記述子の名前空間と同じかそれ以降である必要があります。

拡張の名前空間値	互換性のある AIR バージョン	ANE SWF バージョン
ns.adobe.com/air/extension/2.5	AIR 3 以降	13
ns.adobe.com/air/extension/3.1	AIR 3.1 以降	14
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

注意：プラットフォームオプション（platform.xml）ファイルには ns.adobe.com/air/extension/3.1 以降の名前空間が必要です。-platformoptions フラグを使用して ANE をパッケージ化する場合は、ns.adobe.com/air/extension/3.1 以降と、14 以降のバージョンの SWC を指定する必要があります。プラットフォームオプションファイル機能によっては、以降のバージョンの AIR の名前空間と SWF が必要な場合があります。

ネイティブライブラリの構築

拡張のネイティブライブラリを構築するには、ターゲットデバイスに適した開発環境を使用します。次に、例を示します。

- Android SDK を使用して開発する場合は、Eclipse 統合開発環境（IDE）用の Android 開発ツールプラグインを使用します。
- iOS デバイスおよび Mac OS X デバイス用に開発する場合は、Apple の Xcode IDE を使用します。
- Windows デバイス用に開発する場合は、Microsoft Visual Studio を使用できます。

これらの開発環境を使用したネイティブ拡張の例については、「Adobe AIR 用のネイティブ拡張」を参照してください。

拡張のネイティブ側を、アプリケーションではなくライブラリ内に構築します。ネイティブ拡張を ANE ファイルにパッケージ化するときは、ネイティブライブラリを指定します。

Android ネイティブライブラリ

Android SDK を使用する場合に、ライブラリを JAR ファイルとして指定します。

Android NDK を使用する場合に、次のようなファイル名で共有ライブラリを指定します。

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

アプリケーション APK パッケージを正確にインストールするには、この共有ライブラリ命名規則に従う必要があります。

共有ライブラリを含む ANE パッケージを作成する場合は、共有ライブラリを次のディレクトリ構造で保存する必要があります。

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

iOS ネイティブライブラリ

.a ファイル名拡張子を付けて静的ライブラリを指定します。Xcode IDE の Cocoa Touch 静的ライブラリテンプレートを使用すると、.a ファイルを作成できます。デバイス上で実行されるネイティブライブラリを作成する場合、ターゲットタイプは「デバイス」を使用してください。iOS シミュレーター（AIR 3.3 以降でサポートされている iOS シミュレーター）上で実行されるネイティブライブラリを作成する場合、ターゲットタイプは「シミュレーター」を使用してください。

AIR の各バージョンは、iOS SDK のバージョンと共にバンドルします。対応する AIR バージョンをターゲットとするネイティブ拡張で、そのバージョンの iOS SDK で使用可能な公開されている任意のフレームワークにリンクすることができます。以下の表は、AIR SDK のバージョンとバンドルされる iOS のバージョン、および追加機能のサポートのリストです。

AIR SDK	含まれる iOS SDK	追加ライブラリへのリンク	サードパーティライブラリのバンドル
3.0 - 3.2	4.2	不可	不可
3.3 - 3.4	5.1	可	不可
3.5	6.0	可	可

拡張で特定の AIR SDK バージョンをターゲットにする場合、対応する iOS バージョンよりも後に導入された iOS フレームワークは使用できません。ターゲットにする AIR SDK 以外の、他の共有ライブラリやサードパーティのフレームワークを使用できません。

AIR SDK にバンドルされている iOS SDK を使用する代わりに、AIR 3.3 以降では外部の iOS SDK にリンクすることができます。ADT -platformsdk スイッチを使用して、外部 iOS SDK へのパスを指定します。

AIR はデフォルトで次の iOS フレームワークライブラリにリンクします。

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

他のフレームワークおよびライブラリにリンクするには、プラットフォームオプションの XML ファイルにリンケージオプションを指定します。

注意： AIR 3.4 以降では、ADT -hideAneSymbols yes オプションを使用してシンボルの競合が起こらないようにすることができます。詳しくは、「ネイティブ拡張オプション」を参照してください。

iOS プラットフォームオプション（platform.xml）ファイル

追加のフレームワークやライブラリにリンクするため、またはサードパーティのフレームワークやライブラリをバンドルするために、プラットフォームオプション（platform.xml）ファイルを使用して、ネイティブ拡張で iOS 固有のオプションを指定することができます。iOS の -platform フラグの後に -platformoptions フラグを指定して拡張をパッケージ化すると、プ

ラットフォームオプションファイルは ANE に追加されます。後で開発者が拡張を使用するアプリケーションの IPA ファイルを作成すると、ADT は platform.xml ファイル内でこのオプションを使用して追加ライブラリにリンクしたり、バンドルされた依存関係を含めたりします。

注意：プラットフォームオプションファイルには任意の名前を付けることができます。「platform.xml」という名前を付ける必要はありません。

iOS プラットフォームオプションファイルは iPhone-ARM（デバイス）プラットフォームおよび iPhone-x86（iOS シミュレーター）プラットフォームの両方と共に使用できます。

プラットフォームオプションファイルには AIR 3.1 以降が必要です。

iOS 用の ANE のパッケージ化に **packagedDependencies** 機能を使用する場合は、次の項目を追加します。

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

platformoptions.xml ファイルの linkerOptions タグ内に追加してください。

iOS リンカーオプション

iOS リンカーオプションは、任意のオプションをリンカーに渡す方法を提供します。指定されたオプションは、変更されずにリンカーに渡されます。これを使用して、追加の iOS フレームワークなど、追加のフレームワークやライブラリにリンクできます。リンカーオプションを指定するには、プラットフォームの options xml ファイルで <linkerOptions> タグを使用します。以下の例で示すように、<linkerOptions> タグ内で、各リンケージオプションを <options> タグのペアでラップして指定します。

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

この方法でリンクされた依存関係は、ネイティブ拡張をネイティブ拡張自体とは別に使用する開発者に配布される必要があります。これは、iOS に含まれる追加のライブラリを使用する際に最も便利な方法ですが、AIR ではデフォルトではリンクされません。このオプションを使用して、別に提供する静的ライブラリに開発者がリンクできるようにすることもできます。

<linkerOptions> タグには、AIR 3.3 以降が必要です。

パッケージ化されたサードパティの依存関係

ライブラリのソースコードにアクセスせずに、または拡張とは別にライブラリにアクセスすることを開発者に強要せずに、ネイティブ拡張内で静的ライブラリ（ネイティブサードパーティライブラリなど）を使用する場合があります。静的ライブラリをパッケージ化された依存関係として指定することで、ネイティブ拡張と共にバンドルすることができます。プラットフォームの options xml ファイルで、<packagedDependencies> タグを使用します。以下の例に示すように、拡張パッケージに含める各依存関係に対して、その名前または相対パスを <packagedDependency> タグのペアで囲んで指定します。

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

パッケージ化された依存関係は、.a、.framework または .o のいずれかの拡張子を持つ静的ライブラリである必要があります。ライブラリをデバイス上で使用するには、ライブラリが ARMv7 アーキテクチャをサポートする必要があり、iOS シミュレーターと共に使用するには、i386 アーキテクチャをサポートする必要があります。

ネイティブ拡張をパッケージ化する場合は、依存関係の名前を -platformoptions フラグとして指定する必要があります。次の例に示すように、platform.xml ファイルのファイル名の後、続く -package フラグの前に、依存関係をリストします。

```
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> タグには、AIR 3.5 以降が必要です。

プライベート埋め込みフレームワークの使用

Xcode6 以降でプライベートフレームワークを使用して iOS ANE を作成するには、次の手順を実行します。

1. platform.xml を編集します。

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

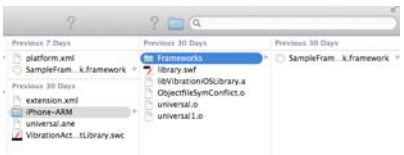
   上記の項目を linkerOptions タグ内に追加します。

   次に例を示します。

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

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

2. Frameworks という名前のフォルダーを既存の iPhone-ARM フォルダー内に作成します。
3. プライベートフレームワークを Frameworks フォルダーにコピーして、フレームワークを ANE とともにパッケージ化します。



プライベートフレームワークのコピー

プラットフォームオプション（platform.xml）ファイルの例

以下は、プラットフォームオプション（platform.xml）ファイルの構成の例です。

```
<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>
```

このプラットフォームオプションファイルをネイティブ拡張パッケージに含めるには、以下のような ADT コマンドを使用できます。

```
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 ネイティブライブラリ

Mac OS X デバイスの場合は、.framework ライブラリを指定します。ライブラリを構築するときは、ベース SDK の Xcode プロジェクトの設定が Mac OS X 10.5 SDK であることを確認します。

拡張として使用するために Mac OS X フレームワークをコンパイルする際には、すべてのシナリオで AIR フレームワークへの依存性を適切に解決できるように、次のオプションを設定します。

- LD_RUNPATH_SEARCH_PATHS に、@executable_path/../runtimes/air/mac、@executable_path/../Frameworks および /Library/Frameworks を追加します。
- フレームワークへの弱いリンクとフラットな名前空間オプションを使用します。

これらのオプション設定を同時に使用することで、まずアプリケーションで AIR フレームワークの適切なコピーが読み込まれ、次に拡張でその読み込み済みのコピーが使用されるようになります。

Windows ネイティブライブラリ

Windows デバイスの場合は、ライブラリを DLL ファイルとして指定します。lib/windows ディレクトリ内の AIR SDK ディレクトリにあるライブラリ FlashExtensions.lib を、DLL に動的にリンクします。また、ネイティブコードライブラリで Microsoft の C ランタイムライブラリのいずれかを使用している場合は、マルチスレッドの静的バージョンの C ランタイムライブラリにリンクします。このタイプのリンクを指定するには、/MT コンパイラーオプションを使用します。

ネイティブ拡張パッケージの作成

ネイティブ拡張をアプリケーション開発者に提供するには、関連するすべてのファイルを 1 つの ANE ファイルにパッケージ化します。ANE ファイルは、以下を含むアーカイブファイルです。

- 拡張の ActionScript ライブラリ
- 拡張のネイティブコードライブラリ
- 拡張記述ファイル
- 拡張の証明書
- 拡張のリソース（画像など）

ANE ファイルを作成するには AIR 開発ツール（ADT）を使用します。ADT の詳細なドキュメントは、「AIR 開発ツール」にあります。

ADT による拡張のパッケージ化の例

以下の例では、ADT を使用して ANE ファイルをパッケージ化する方法を説明します。この例では、次のデバイス上で動作するアプリケーションで使用するための ANE ファイルをパッケージ化します。

- Android デバイス

- Android x86 デバイス
- iOS デバイス
- iOS シミュレーター
- デフォルトの ActionScript 専用の実装を使用しているその他のデバイス

```
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
```

この例では、次のコマンドラインオプションを使用して ANE パッケージを作成します。

- <signing options>

  オプションで、ANE ファイルに署名することもできます。詳しくは、40 ページの「ネイティブ拡張の署名済み証明書の作成」を参照してください。

- -target ane

  -target フラグでは、作成するパッケージのタイプを指定します。ane ターゲットを使用すると、ネイティブ拡張がパッケージ化されます。

- MyExtension.ane

  作成するパッケージファイルの名前を指定します。ファイル名拡張子は .ane を使用します。

- MyExt.xml

  拡張記述ファイルを指定します。このファイルでは、拡張 ID とサポートされているプラットフォームを指定します。AIR では、この情報を使用して、アプリケーションに対応する拡張を特定し、ロードします。この例では、拡張記述ファイルは次のようになります。

```
<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>
```

詳しくは、71 ページの「ネイティブ拡張記述ファイル」を参照してください。

- MyExtension.swc

  拡張の ActionScript 側を含む SWC ファイルを指定します。

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

  -platform フラグでは、この ANE ファイルがサポートするプラットフォームの名前を指定します。名前の後のオプションでは、プラットフォーム固有のライブラリとリソースの場所を指定します。この場合、Android-ARM プラットフォームの -C オプションは、相対ディレクトリ platform/Android を作業ディレクトリにすることを示します。プラットフォーム固有のディレクトリとファイルは、新しい作業ディレクトリに対して相対的な位置に存在することになります。

  つまり、この例では、相対ディレクトリ platform/Android には、Android のネイティブコードライブラリとリソースがすべて含まれます。また、Android プラットフォーム固有の library.swf ファイルおよびその他の Android プラットフォーム固有の SWF ファイルも含まれます。

  iPhone-ARM プラットフォーム用の -platformoptions フラグは、プラットフォーム固有のオプションを指定する場合に使用するオプションの項目です。これらのオプションは、iOS フレームワーク（デフォルトのフレームワーク以外）へのリンクや、サードパーティの静的ライブラリとネイティブ拡張のバンドルに関するオプションを含みます。43 ページの「iOS ネイティブライブラリ」を参照してください。

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

  -platform オプションで default プラットフォームを指定するときは、ネイティブコードファイルは指定しないでください。library.swf ファイルと、その他の SWF ファイル（存在する場合）のみを指定してください。

ANE パッケージ内の SWC ファイルと SWF ファイル

SWC ファイルは、ADT パッケージコマンドの -swc オプションで指定します。この SWC ファイルは ActionScript ライブラリです。これには library.swf というファイルが含まれています。ADT では、library.swf を SWC ファイルから ANE ファイルに配置します。ネイティブ拡張を使用している AIR アプリケーションでは、コンパイルが可能なように、拡張の ANE ファイルがライブラリパスに含まれています。実際には、アプリケーションでは、library.swf 内のパブリックインターフェイスに対するコンパイルが行われます。

ターゲットプラットフォームごとに別々の ActionScript 実装を作成する場合もあります。その場合は、各プラットフォームの SWC ファイルをコンパイルし、各 SWC ファイルに含まれる library.swf ファイルを適切なプラットフォームディレクトリに配置してから、ADT パッケージ化コマンドを使用します。library.swf は、WinZip などの解凍ツールを使用して SWC ファイルから抽出できます。

例えば、作成する拡張のパブリックインターフェイスがどのプラットフォーム用もすべて同じであるとします。その場合、ADT コマンドの -swc オプションで指定する、プラットフォームに特有の SWC ファイルは、どのプラットフォーム用でもかまいません。SWC ファイルに含まれている SWF ファイルはアプリケーションのコンパイルにのみ使用されるものだからです。ネイティブ拡張の実行時に使用されるのは、プラットフォームディレクトリに配置した SWF ファイルです。

library.swf ファイルの注意事項は次のとおりです。

- ADT では、プラットフォームごとに library.swf という名前のメインの SWF ファイルを指定する必要があります。SWC ファイルを作成すると、library.swf がその SWF ファイルの名前となります。
- library.swf ファイルは、プラットフォームごとに異なる場合があります。
- ActionScript 側にプラットフォームの依存関係がない場合は、各プラットフォームの library.swf を同じにすることもできます。
- 各プラットフォームの library.swf では、プラットフォーム固有のディレクトリに含めた他の SWF ファイルをロードすることができます。この場合の他の SWF ファイルは、どのような名前でもかまいません。

アプリケーションのパッケージ化に関する ANE ファイルのルール

ネイティブ拡張を使用するアプリケーションをパッケージ化するには、ADT を使用します。拡張を使用するアプリケーションをパッケージ化する際に、アプリケーションのパッケージ化ターゲットと一致するプラットフォームが ANE ファイル内で指定されているかどうかが、ADT によって検証されます。例えば、プラットフォーム Android-ARM は、Android apk パッケージと照合されます。

また、ADT では、default プラットフォームが任意のターゲットパッケージと照合されます。default プラットフォームは、ActionScript 専用の拡張を指定します。アプリケーションバンドル拡張を使用する AIR アプリケーションについて考えてみます。AIR では、拡張で指定されているプラットフォームの中にデバイスと対応するものがない場合にのみ、default プラットフォーム拡張の ActionScript ライブラリをロードします。

例えば、iPhone-ARM、Android-ARM および default の各プラットフォームを指定しているアプリケーションバンドル拡張について考えてみます。この拡張を使用しているアプリケーションを Windows プラットフォーム上で実行した場合、拡張の default プラットフォームライブラリが使用されます。

したがって、アプリケーションバンドル用に ANE ファイルを作成する場合は、ANE ファイルを使用するアプリケーションをパッケージ化するときに ADT で使用される次のルールを考慮してください。

- Android アプリケーションパッケージを作成するには、ANE ファイルに Android-ARM プラットフォームを含める必要があります。または、ANE ファイルに default プラットフォームとその他のプラットフォームを 1 つ以上含める必要があります。
- iOS アプリケーションパッケージを作成するには、ANE ファイルに iPhone-ARM プラットフォームを含める必要があります。または、ANE ファイルに default プラットフォームとその他のプラットフォームを 1 つ以上含める必要があります。

- iOS シミュレーターアプリケーションパッケージを作成するには、ANE ファイルに iPhone-x86 プラットフォームを含める必要があります。または、ANE ファイルに default プラットフォームとその他のプラットフォームを 1 つ以上含める必要があります。
- Mac OS X アプリケーションパッケージを作成するには、ANE ファイルに MacOS-x86-64 プラットフォームを含める必要があります。または、ANE ファイルに default プラットフォームとその他のプラットフォームを 1 つ以上含める必要があります。
- Windows アプリケーションパッケージを作成するには、ANE ファイルに Windows-x86 プラットフォームを含める必要があります。または、ANE ファイルに default プラットフォームとその他のプラットフォームを 1 つ以上含める必要があります。

アプリケーションには 1 つのプラットフォーム実装しかバンドルされませんが、ADL ユーティリティを使用して拡張を含むアプリケーションをテストする際には、実装が実行時に選択されます。このように実行時に選択されることから、テストプラットフォームおよび ANE パッケージによって動作が異なることがあります。例えば、ANE に Android-ARM、Windows-x86 および default のプラットフォームの実装が含まれている場合、テストコンピューターが Windows を実行しているか Mac OS X を実行しているかによってテスト時に使用される実装が異なります。Windows では、Windows-x86 プラットフォーム実装が使用されます（これは、モバイルプロファイルでテストする場合も当てはまります）。Mac OS X では、default 実装が使用されます。

ANE パッケージに追加の Android .so 共有ライブラリを含める

プラットフォーム Android-ARM をターゲットとするネイティブ拡張について考えます。拡張のネイティブ側の主要ライブラリは次のいずれかになります。

- .so ライブラリ（Android NDK を使用する場合）
- JAR ファイル（Android SDK を使用する場合）

ただし、場合によっては、拡張用の主要な .so ライブラリまたは JAR ファイル以外にも、追加のネイティブライブラリ（.so ライブラリ）がネイティブ側に必要となることがあります。

次に、例を示します。

- Java API を使用したネイティブ拡張の開発で、その Java コードから JNI（Java Native Interface）を使用して .so ネイティブライブラリにアクセスすることが必要な場合。
- C API を使用したネイティブ拡張の開発で、コードを複数の共有ライブラリに分割することになり、拡張内のロジックに基づいて実行時に適切な .so 共有ライブラリを読み込むことが必要な場合。

この ANE パッケージを作成するときは、次のディレクトリ構造を使用します。

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

ADT を使用してこの ANE パッケージを作成するときは、Android プラットフォームディレクトリのコンテンツを指定するように -platform オプションを設定します。

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

アプリケーション開発者が ADT を使用して ANE パッケージを APK パッケージに取り込むと、APK パッケージには次のライブラリが含まれます。

- libs/armeabi-v7a 内のライブラリ（ADT のターゲットが apk または apk-captive-runtime の場合）。
- libs/armeabi 内のライブラリ（ADT のターゲットが apk-emulator、apk-debug または apk-profile の場合）。

注意：iPhone-ARM プラットフォームの場合は、共有ライブラリを ANE ファイルに含めることはできません。詳しくは、42 ページの「ネイティブライブラリの構築」を参照してください。

ネイティブ拡張パッケージにリソースを含める

拡張の ActionScript 側とネイティブコード側では、画像などの外部リソースを使用する場合があります。

ActionScript 側では、プラットフォームに依存しない SWC ファイルに、必要なリソースを含めることができます。プラットフォームに依存する SWF ファイルにリソースが必要な場合は、プラットフォームに依存するディレクトリ構造（ADT コマンドで指定）内にリソースを含めます。

ネイティブコード側でも、プラットフォームに依存するディレクトリ構造にリソースを含めることができます。ネイティブコードディレクトリと相対的な位置にあるサブディレクトリ内の、ネイティブコードで想定されている場所に、リソースを配置します。ADT では、ANE をパッケージ化するときに、このディレクトリ構造を保持します。

Android デバイスおよび iOS デバイスでリソースを使用する場合も、いくつかの追加要件があります。

Android デバイス上のリソース

Android-ARM プラットフォームの場合は、ネイティブコードライブラリを含むディレクトリに対して相対的な位置にある res というサブディレクトリにリソースを配置します。developer.android.com にある「Providing Resources」の説明に従って、ディレクトリにリソースを配置します。ADT では、ANE ファイルをパッケージ化するときに、生成される ANE パッケージの Android-ARM/res ディレクトリにリソースが配置されます。

拡張の Java ネイティブコードライブラリからリソースにアクセスするときは、FREContext クラスの getResourceID() メソッドを使用します。標準の Android リソース ID メカニズムを使用してリソースにアクセスしないでください。getResourceID() メソッドについて詳しくは、128 ページの「メソッドの詳細」を参照してください。

getResourceId() メソッドは、リソース名の文字列パラメーターを使用します。アプリケーション内の拡張間におけるリソース名の競合を回避するために、拡張内の各リソースファイルには一意の名前を指定してください。例えば、リソース名の先頭に拡張 ID を付け、com.sample.ext.MyExtension.myImage1.png のようなリソース名を作成します。

R.* メカニズムを使用したネイティブリソースへのアクセス

以前のリリースでは、Android ネイティブ拡張のネイティブの Android リソースにアクセスするには、getResourceID() API を使用する必要がありました。R.* メカニズムを ANE とともに使用することはできませんでした。AIR 4.0 以降では、R.* メカニズムを使用してリソースにアクセスできます。R.* メカニズムを使用する場合は、プラットフォーム記述ファイル（platform.xml）を使用してください。このファイルには、すべての依存関係が定義されています。

```
<?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>
```

platform.xml 内に定義された依存関係の例を次に示します。

```
<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>
```

次に、各指定項目について説明します。

- packagedDependencies は、ANE が依存するすべての jar の名前を指定するために使用します。
- packagedResources は、ANE またはその他の jar ファイルが使用するリソースを定義します。
- folderName は、リソースフォルダーの名前を定義します。
- packageName は、リソースを使用する jar のパッケージ名を定義します。
- packagedDependencies と packagedResources は拡張の名前空間 4.0 以降から使用できます。

Android-ARM フォルダーには、ANE jar ファイルとリソースおよびサードパーティの jar ファイルがすべて格納されます。サンプルの ANE パッケージ化コマンドを次に示します。

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

R.* リソースアクセスメカニズムを使用する場合、サードパーティの jar ファイルとリソースを ANE jar とリソースに結合する必要はありません。ADT は jar とリソースを内部で結合します。すべての依存関係とリソースは引き続き ANE にパッケージ化する必要があります。

次の点に注意してください。

- ANE プロジェクトは、使用する R.* リソースアクセスメカニズム用のライブラリプロジェクトである必要があります。
- リソースフォルダーの名前に制限はありません。「res」またはその他の任意の有効な文字列で始まる名前を使用できます。
- ANE のパッケージ化で -platformoptions スイッチを使用しない場合は、getResourceId() メカニズムを使用してリソースにアクセスする必要があります。

iOS デバイス上のリソース

リソースの場所

ADT を使用して iOS デバイス用の ANE ファイルを作成する前に、ネイティブコードライブラリを含むディレクトリに、ローカライズされていないリソースを配置します。ローカライズされているリソースは、次の節で説明するように、サブディレクトリに配置されます。

ただし、iOS アプリケーションバンドルでは、アプリケーションバンドルディレクトリの最上位にリソースが含まれています。このようなリソースには、各拡張のプラットフォーム固有の部分で使用されるすべてのリソースが含まれます。AIR アプリケーション開発者が ANE ファイルをアプリケーションと共にパッケージ化すると、ADT では次の処理が行われます。

1. ANE パッケージの iPhone-ARM ディレクトリの内容を確認します。
2. そのディレクトリ内の library.swf を除くすべてのファイルおよび拡張記述ファイルを、リソースファイルと見なします。

3 リソースファイルをアプリケーションの最上位ディレクトリに移動します。

複数の拡張からのリソースファイルが同じ場所に移動されるので、リソースファイル名が競合する可能性があります。競合が発生した場合、ADT ではアプリケーションがパッケージ化されず、エラーが報告されます。したがって、拡張内の各リソースファイルには一意の名前を指定してください。例えば、名前の先頭に拡張 ID を付け、com.sample.ext.MyExtension.myImage1.png のようなリソース名を作成します。

注意：リソースは、拡張のディレクトリではなく、アプリケーションの最上位ディレクトリにあります。したがって、リソースにアクセスするには、ActionScript プロパティ File.applicationDirectory を使用します。ActionScript API ExtensionContext.getExtensionDirectory() を使用して拡張ディレクトリに移動し、リソースを検索することはしないでください。リソースは、そこにはありません。

ローカライズされたリソース

拡張のネイティブコードライブラリ（ActionScript 側ではない）では、ローカライズされたリソースを使用できます。ローカライズされたリソースを使用するには、次の手順を実行します。

- ADT を使用して ANE ファイルを作成するときに指定したプラットフォーム固有のディレクトリで、ローカライズされたリソースを言語固有のサブディレクトリに配置します。これらのサブディレクトリに、それぞれ次のように名前を付けます。

  language[**_region**].lproj

  language および region の値を、iOS の規則に従って設定します。iOS Developer Library の「Language and Locale Designations」を参照してください。

- developer.apple.com/library/ios/navigation で説明されている形式で、ローカライズされた文字列を .strings ファイル内に指定します。ただし、どのファイルにも、Localizable.strings という名前は付けないでください。これはアプリケーションバンドルで使用されるデフォルト名であるからです。

以下のディレクトリ構造は、ANE ファイルにパッケージ化するすべての iOS プラットフォーム固有ファイルを含むディレクトリの例です。

```
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：AIR for TV 向けネイティブ拡張の構築およびインストール

AIR for TV デバイス向けネイティブ拡張を開発するには、AIR for TV 拡張開発キット（EDK）にアクセスできる必要があります。この EDK は、AIR for TV を自社製品に付属させているデバイス製造元や system-on-a-chip（SoC）製造元に配布されています。

AIR for TV を使用している製造元について詳しくは、『**Getting Started with Adobe AIR for TV（PDF）**』（PDF）を参照してください。

AIR for TV 拡張の開発におけるタスクの概要

AIR for TV デバイス向けのネイティブ拡張を開発する際には、次の反復的なタスクを実行します。

1 ネイティブ実装を記述します。

詳しくは、15 ページの「C を使用したネイティブ側のコーディング」を参照してください。

2 実際の ActionScript 実装を記述します。

詳しくは、7 ページの「ActionScript 側のコーディング」を参照してください。必ず 13 ページの「ネイティブ拡張の下位互換性」を考慮してください。

3 ActionScript スタブ実装を記述し、オプションで ActionScript シミュレーター実装を記述します。

詳しくは、61 ページの「デバイスバンドル拡張とスタブ拡張」および 62 ページの「拡張のサポートの確認」を参照してください。

4 ネイティブ拡張に署名するための証明書を作成します。この手順は必須ではありません。

詳しくは、40 ページの「ネイティブ拡張の署名済み証明書の作成」を参照してください。

5 AIR for TV の make ユーティリティを使用して、デバイスバンドル拡張およびスタブ拡張をビルドします。このプロセスによって、デバイスにデバイスバンドル拡張をインストールするための ZIP ファイルが作成されます。AIR アプリケーションを構築しテストするためのスタブ拡張の ANE ファイルも作成されます。

詳しくは、63 ページの「AIR for TV ネイティブ拡張の構築」を参照してください。

6 ActionScript シミュレーター実装が利用できる場合、AIR for TV の make ユーティリティを使用してシミュレーター拡張をビルドします。このプロセスによって、AIR アプリケーションを構築しテストするための ANE ファイルも作成されます。

詳しくは、63 ページの「AIR for TV ネイティブ拡張の構築」を参照してください。

7 ZIP ファイルおよび ANE ファイルに、画像などの必要なリソースを追加します。

詳しくは、69 ページの「AIR for TV ネイティブ拡張へのリソースの追加」を参照してください。

8 デスクトップコンピューターでシミュレーター拡張をテストします。

詳しくは、「AIR for TV アプリケーションのデバッグ」を参照してください。

9 デバイスに、デバイスバンドル拡張をインストールします。

詳しくは、69 ページの「AIR for TV デバイスへのデバイスバンドル拡張のインストール」を参照してください。

10 デバイス上でデバイスバンドル拡張をテストします。

詳しくは、70 ページの「AIR for TV デバイス上での AIR アプリケーションの実行」および「AIR for TV アプリケーションのデバッグ」を参照してください。

11 アプリケーション開発者に、スタブ ANE ファイル、シミュレーター ANE ファイルのどちらか、または両方を配布します。

AIR for TV 拡張の例

テレビ用 Adobe® AIR® では、ネイティブ拡張の例がいくつか提供されます。ネイティブ実装は C++ で記述されており、AIR for TV 拡張開発キット（EDK）を使用します。この EDK は、AIR for TV を自社製品に付属させているデバイス製造元や system-on-a-chip（SoC）製造元に配布されています。AIR for TV EDK および AIR for TV 拡張の作成について詳しくは、63 ページの「AIR for TV ネイティブ拡張の構築」を参照してください。

AIR for TV 拡張の開発者は、次のことを実行できます。

- これらの例をコピーし、拡張の出発点として使用します。
- 様々な C API 拡張関数や ActionScript ExtensionContext クラスの使用方法を示すコードサンプルを確認するために、これらの例を参照します。
- これらの例のいずれかの makefile をコピーし、自分の拡張用の makefile を作成するための出発点として使用します。

HelloWorld の例

HelloWorld の例は、AIR for TV 配布ファイルの次のディレクトリにあります。

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

HelloWorld の例は、拡張の基本動作を示すシンプルな拡張です。以下を実行します。

- ExtensionContext.call() メソッドを使用して、ActionScript 側からネイティブ実装に文字列を渡します。
- ネイティブ実装から ActionScript 側に文字列を返します。
- ネイティブ実装で、ActionScript 側に非同期イベントを送信するスレッドを開始します。

次の表に、各ファイルと、その各ファイルの helloworld/ ディレクトリに対する場所を示します。

ファイル	説明
HelloWorld.as ディレクトリ： as/device/src/tv/adobe/extension/example/	HelloWorld クラスを定義する、実際の（スタブではない）ActionScript 側の拡張。以下を実行します。 • ExtensionContext インスタンスを作成します。 • 拡張の ActionScript API である、Hello() および StartCount() を定義します。 • ExtensionContext インスタンス上のイベントを監視し、HelloWorld インスタンスのリスナーにイベントを再送出します。
HelloWorld.as ディレクトリ： as/distributable/src/tv/adobe/extension/example/	HelloWorld クラスを定義するスタブ拡張の ActionScript。この ActionScript 専用スタブは、拡張の ActionScript API である Hello() および StartCount() を定義します。ただし、このスタブ実装はネイティブ関数を呼び出しません。
HelloWorldExtensionClient.as ディレクトリ： client/src	拡張を使用する AIR アプリケーション。この AIR アプリケーションは拡張のクライアントです。以下を実行します。 • HelloWorld クラスのインスタンスを作成します。 • HelloWorld インスタンスのイベントを監視します。 • HelloWorld インスタンスの Hello() および StartCount() API を呼び出します。
HelloWorldExtensionClient-app.xml ディレクトリ： client/src	AIR アプリケーション記述ファイル。<extensions> エレメントが含まれ、extensionID 値として tv.adobe.extension.example.HelloWorld が指定されています。
HelloWorld.h ディレクトリ： native/	HelloWorld クラスの C++ ヘッダーファイル。
HelloWorld.cpp ディレクトリ： native/	HelloWorld クラスの C++ 実装ファイル。この実装は、以下を実行します。 • コンソールに文字列パラメーターを書き出す FREFunction Hello() を定義します。また、「Hello from extensionland」という文字列を返します。 • ExtensionContext インスタンスに 500 ミリ秒おきにイベントを送信する非同期スレッドを開始する FREFunction StartCount() を定義します。
HelloWorldExtension.cpp ディレクトリ： native/	次の C API 拡張関数の実装が含まれます。 • FREInitializer() • FREContextInitializer() • FREContextFinalizer() • FREFinalizer()
PlatformEDKExtension_HelloWorld.mk	HelloWorld 拡張の makefile。

ファイル	説明
ExtensionUtil.h ディレクトリ： **<AIR for TV installation directory>**/products/stagecraft/source/ae/edk	C または C++ 実装のコーディングに便利なマクロが含まれます。
ExtensionBridge.cpp ディレクトリ： **<AIR for TV installation directory>**/products/stagecraft/source/ae/edk	AIR for TV 拡張モジュール実装。ネイティブ実装を構築する際に、ビルド内にこのソースファイルを含めます。
phonyEdkAneCert.p12 ディレクトリ： **<AIR for TV installation directory>**/products/stagecraft/source/ae/edk	偽の証明書。make ユーティリティで ANE ファイル内に HelloWorld 拡張をパッケージ化するために使用します。
extension.mk ディレクトリ： **<AIR for TV installation directory>**/products/stagecraft/source/ae/edk	拡張モジュールの makefile。このファイルは変更しないでください。

Process の例

Process の例は、AIR for TV の配布の次のディレクトリにあります。

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

Process 拡張では、AIR アプリケーションは Linux プロセスを操作できます。例えば次のような機能があります。

- Linux プロセスをコマンド実行します（spawn）。プロセスは、AIR アプリケーションが指定する Linux コマンドを実行します。
- プロセスのプロセス識別子を取得します。
- プロセスが完了したかどうかを確認します。
- プロセスが完了したイベントを受け取ります。
- 完了したプロセスのリターンコードを取得します。
- プロセスが stdout または stderr に書き出したことを示すイベントを受け取ります。
- stdout および stderr の出力文字列を受け取ります。
- stdin に文字列を書き込みます。
- プロセスに中断シグナルを送信します。
- プロセスを強制終了します（kill）。

次の表に、各ファイルと、その各ファイルの process/ ディレクトリに対する場所を示します。

ファイル	説明
Process.as ディレクトリ： as/device/src/tv/adobe/extension/process/example/	Process クラスを定義する、実際の（スタブではない）ActionScript 側の拡張。以下を実行します。 • ExtensionContext インスタンスを作成します。 • 拡張の ActionScript API を定義します。 • ExtensionContext インスタンス上のイベントを監視し、Process インスタンスのリスナーにイベントを再送出します。
ProcessEvent.as ディレクトリ： as/device/src/tv/adobe/extension/process/example/	ProcessEvent クラスを定義します。これは、Event クラスの派生クラスです。 AIR アプリケーション ActionScript は、これらの ProcessEvent 通知を監視します。
Process.as ディレクトリ： as/distributable/src/tv/adobe/extension/process/example/	Process クラスを定義するスタブ拡張の ActionScript。この ActionScript 専用スタブは、拡張の ActionScript API を定義します。ただし、このスタブ実装はネイティブ関数を呼び出しません。
ProcessExtensionClient.as ディレクトリ： client/simple/src	拡張を使用する AIR アプリケーション。この AIR アプリケーションは拡張のクライアントです。AIR アプリケーションが Process 拡張 API を使用する方法の例を示します。
ProcessExtensionClient-app.xml ディレクトリ： client/simple/src	AIR アプリケーション記述ファイル。<extensions> エレメントが含まれ、extensionID 値として tv.adobe.extension.process.Process が指定されています。
Process.h ディレクトリ： native/	Process 抽象クラスの C++ ヘッダーファイル。
ProcessLinux.h ディレクトリ： native/	ProcessLinux 具象クラスの C++ ヘッダーファイル。 ProcessLinux クラスは、Process クラスの派生クラスです。Process クラスの Linux 実装向けの private メソッドおよびデータを宣言します。
ProcessLinux.cpp ディレクトリ： native/	ProcessLinux クラスの C++ 実装。この実装には次の機能が含まれます。 • FREFunction 関数を定義します。これらの関数は Linux システムコールを使用して、例えば、Linux プロセスの分岐（fork）と実行（exec）、stdin、stdout および stderr の操作を行います。 • コマンド実行（spawn）されたプロセスのステータスを監視します。実装はこの目的でスレッドを作成します。スレッドは、C 拡張 API の FREDispatchStatusEventAsync() を使用して、イベントをレポートします。 • FREFunction 関数から ActionScript 側に情報を返すための FREObject 変数を作成する、ヘルパー関数を定義します。これらのヘルパー関数は、FRENewObjectFromBool()、FRENewObjectFromUTF8()、FRENewObjectFromUint32() などの、C API 拡張関数を使用します。

ファイル	説明
ProcessExtension.cpp ディレクトリ： native/	次の C API 拡張関数の実装が含まれます。 • FREInitializer() • FREContextInitializer() • FREContextFinalizer() • FREFinalizer()
PlatformEDKExtension_Process.mk	Process 拡張の makefile。
ExtensionUtil.h ディレクトリ： **<AIR for TV installation directory>**/products/stagecraft/source/ae/edk	C または C++ 実装のコーディングに便利なマクロが含まれます。
ExtensionBridge.cpp ディレクトリ： **<AIR for TV installation directory>**/products/stagecraft/source/ae/edk	AIR for TV 拡張モジュール実装。ネイティブ実装を構築する際に、ビルド内にこのソースファイルを含めます。
phonyEdkAneCert ディレクトリ： **<AIR for TV installation directory>**/products/stagecraft/source/ae/edk	偽の証明書。make ユーティリティで ANE ファイル内に Process 拡張をパッケージ化するために使用します。
extension.mk ディレクトリ： **<AIR for TV installation directory>**/products/stagecraft/source/ae/edk	拡張モジュールの makefile。このファイルは変更しないでください。

x86Desktop プラットフォーム上でのサンプル拡張のビルド

サンプル拡張の Hello World または Process を x86Desktop プラットフォーム上でビルドするには、まず、x86Desktop プラットフォーム用の AIR for TV ディストリビューションをビルドします。その手順については、『Getting Started with Adobe AIR for TV』（PDF）の「Quick Start on Linux」を参照してください。

注意：AIR 3 SDK、オープンソース Flex® SDK および Java™ ランタイムがインストールされていることと、PATH 環境変数に bin ディレクトリが含まれていることを確認してください。AIR for TV 向けネイティブ拡張をビルドするには、これらのライブラリが必要です。

次に、サンプル拡張の Hello World または Process のビルドを、x86Desktop ビルドに追加します。次の手順を実行してください。

1 作業ディレクトリを次の場所に変更します。

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

2 ビルドする拡張の .mk ファイルにリンクします。次に、例を示します。

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

3 作業ディレクトリを次の場所に変更します。

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

4 ディストリビューション（サンプル拡張を含む）をビルドします。

```
make
```

または、サンプル拡張だけをビルドすることもできます。次に、例を示します。

```
make PlatformEDKExtension_HelloWorld.mk
```

make ユーティリティにより、1 つのサンプル拡張に対して 2 つのファイルが作成されます。それらのファイルは、SC_BUILD_MODE にデバッグとリリースのどちらを指定したかに応じて、次のディレクトリのいずれかに配置されます。

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

サンプル拡張に対して make ユーティリティにより作成されるファイルは次のとおりです。

- デバイスにバンドルされた拡張を含む ZIP ファイル。
- スタブ拡張またはシミュレーター拡張を含む ANE ファイル。

デバイスバンドル拡張とスタブ拡張

テレビデバイス用 Adobe® AIR® 向けのネイティブ拡張を記述する際には、拡張の次の 2 つのバリエーションを作成する必要があります。

- デバイスバンドル拡張（実際の拡張とも呼びます）
- スタブ拡張

さらに、オプションで 3 つ目のバリエーションであるシミュレーター拡張も作成できます。

デバイスバンドル拡張

デバイスバンドル拡張は、デバイスにインストールされるバリエーションです。ActionScript 側ではネイティブ実装の関数を呼び出します。ネイティブ実装と共にこの実際の ActionScript 実装を構築し、ZIP ファイルを作成します。デバイス製造元は、デバイスの特定のディレクトリにこのファイルを解凍します。

スタブ拡張

ネイティブスタブ拡張は、実際の ActionScript 実装と同じ ActionScript インターフェイスを持っていますが、この ActionScript メソッドは何も実行しません。スタブ拡張は、ActionScript 専用の拡張です。つまり、ネイティブ実装はありません。ActionScript スタブ実装を構築する際には、ANE ファイルを作成します。

AIR アプリケーション開発者は、次の 3 つの目的でこの ANE ファイルを使用します。

- ネイティブ拡張を使用する AIR アプリケーションをコンパイルします。
- ターゲットデバイスではなく、デスクトップコンピューター上で AIR アプリケーションを実行します。
- AIR アプリケーションパッケージに含めます。

シミュレーター拡張

オプションである 3 つ目のバリエーションは、シミュレーター拡張です。この実装も、実際の ActionScript 実装と同じ ActionScript インターフェイスを持っています。ただし、その ActionScript メソッドは、ActionScript 内の拡張の動作をシミュレートします。スタブ拡張と同様に、シミュレーター拡張は ActionScript 専用の拡張です。つまり、ネイティブ実装はありません。ActionScript シミュレーター実装を構築する際には、ANE ファイルを作成します。

AIR アプリケーション開発者は、シミュレーター拡張 ANE ファイルを使用してアプリケーションをコンパイルできます。この ANE ファイルを使用して、スタブ拡張でテストするよりも徹底的に、デスクトップコンピューター上でアプリケーションをテストできます。また、AIR アプリケーションパッケージにシミュレーター拡張を含めることもできます。

注意：シミュレーター拡張は、スタブ拡張の代わりに、またはスタブ拡張に追加して作成できます。

デバイスバンドル拡張、スタブ拡張、シミュレーター拡張の使用

AIR アプリケーション開発者は、スタブ拡張およびシミュレーター拡張を使用して以下を実行します。

- スタブ拡張またはシミュレーター拡張を使用して AIR アプリケーションをコンパイルします。
- スタブ拡張またはシミュレーター拡張を使用してデスクトップコンピューター上でアプリケーションをテストします。
- 配布可能な AIR アプリケーションに、スタブ拡張またはシミュレーター拡張をパッケージ化します。

  注意： AIR アプリケーション開発者にスタブ拡張とシミュレーター拡張の両方を提供する場合は、どちらを配布可能なアプリケーションと共にパッケージ化するのか指示してください。

AIR アプリケーションがデバイス上で実行される場合、AIR for TV は以下を実行します。

1. 次の 2 つのデバイス上の対応するデバイスバンドル（実際の）拡張を検索します。
2. 見つかった場合は、AIR アプリケーションで使用するために、AIR for TV がその拡張をロードします。
3. 見つからなかった場合、AIR for TV は、アプリケーションと共にパッケージ化されたスタブ拡張またはシミュレーター拡張を代わりにロードします。

拡張のサポートの確認

ベストプラクティスとしては、ネイティブ拡張と AIR for TV アプリケーションの間のハンドシェークを定義します。ハンドシェークによって、デバイス上の拡張を使用できるかどうかをアプリケーションに伝えることができます。この情報を使用することで、AIR for TV アプリケーションでは、どのロジックパスを使用するかを判断できます。

ActionScript 拡張クラスでこのハンドシェークを行うパブリックインターフェイスを決定します。次に、AIR for TV アプリケーション開発者にこれらのインターフェイスの使用法について指示します。

例えば、次の要素を持つネイティブ拡張について考えます。

- AIR for TV アプリケーションと共にパッケージ化するためのスタブ拡張
- デスクトップコンピューター上でアプリケーションをテストするためのシミュレーター拡張
- ターゲットデバイスにインストールするためのデバイスバンドル拡張（実際の拡張）

各バリエーションの ActionScript 側で、次のような実装で isSupported() というメソッドを定義します。

- ネイティブ実装と連携する実際の ActionScript 実装ではこのメソッドが true を返すように実装します。

  呼び出し元の AIR アプリケーションは、拡張がデバイスバンドルとなるデバイス上で実行されます。これにより、アプリケーションは、拡張を使用して処理を続行できることを判別できます。
- ActionScript スタブ実装ではこのメソッドを、false を返すように実装します。

呼び出し元の AIR アプリケーションは、拡張がデバイスバンドルとならないデバイス上で実行されます。この状況では、AIR for TV は AIR アプリケーションと共にパッケージ化されたスタブ拡張を使用します。false の戻り値は、アプリケーションに対して、拡張の他のどのメソッドも呼び出さないように通知します。アプリケーションは、正常に終了するなどの適切なロジックの決定を行うことができます。

- ActionScript シミュレーター実装ではこのメソッドが true を返すように実装します。

  呼び出し元の AIR アプリケーションはテストの目的で、デスクトップコンピューターで実行されます。この場合、true の戻り値はアプリケーションに対して、拡張を使用して続行できることを通知します。

  ただし、拡張によっては、アプリケーション開発者に、シミュレーター拡張と共にアプリケーションをパッケージ化するよう指示できます。これにより、アプリケーションがデバイスバンドル拡張のないデバイス上で実行される場合に、アプリケーションはシミュレーターバージョンの拡張を使用して続行できます。

AIR for TV ネイティブ拡張の構築

AIR for TV ネイティブ拡張を構築する際には、次の 2 つのバージョンの拡張を構築します。

- デバイスバンドル拡張
- スタブ拡張またはシミュレーター拡張

デバイスバンドル拡張には次のものが含まれます。

- ネイティブ実装（通常 C または C++ で記述）
- ネイティブ実装の関数を呼び出す、実際の ActionScript 実装

スタブ拡張またはシミュレーター拡張は、ActionScript 専用の実装です。

実際の ActionScript スタブ実装およびシミュレーター実装について詳しくは、61 ページの「デバイスバンドル拡張とスタブ拡張」を参照してください。

拡張の署名済み証明書の作成

ネイティブ拡張に電子署名することができます。拡張への署名はオプションです。

AIR for TV の make ユーティリティは、デフォルトで偽の証明書を使用します。この偽の証明書は、テストのみで有効です。証明機関の証明書の作成について詳しくは、40 ページの「ネイティブ拡張の署名済み証明書の作成」を参照してください。

ネイティブ実装の記述

AIR for TV では、拡張のネイティブ実装は AIR for TV モジュールです。モジュール構築の詳細を含む、AIR for TV モジュールの一般的な情報については、『**Optimizing and Integrating Adobe AIR for TV** 』（PDF）を参照してください。

AIR for TV 配布ファイルでは、拡張のネイティブ実装の記述と構築のための拡張開発キット（EDK）が提供されます。

EDK には次のものが含まれます。

- C 拡張 API ヘッダーファイル

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

  このヘッダーファイルは、ネイティブ実装が使用する C の型および関数を宣言するものです。

- 拡張モジュール実装。次のソースファイルにあります。

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

この拡張モジュール実装は変更しないでください。ネイティブ実装を構築する際に、ビルド内にこのソースファイルを含める必要があります。

- デバイスバンドル拡張を構築するための makefile サポート。詳しくは、64 ページの「.mk ファイルの作成」および 67 ページの「make ユーティリティの実行」を参照してください。

注意：AIR for TV EDK では、FREInitializer() メソッドの名前が Initializer() であり、FREFinalizer() メソッドの名前が Finalizer() である必要があります。これらのメソッドについて詳しくは、15 ページの「拡張の初期化」および 19 ページの「拡張のファイナライズ処理」を参照してください。

ディレクトリ構造での ActionScript およびネイティブコードの配置

デバイスバンドル拡張は、ハードウェアプラットフォームごとに異なります。デバイスバンドル拡張を開発する際には、プラットフォームに合わせてサブディレクトリ内にファイルを配置します。このサブディレクトリは、次のディレクトリ内にあります。

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

例えば、CompanyA は PlatformB をターゲットにした開発で次のサブディレクトリを使用します。

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

<yourPlatform>/edk ディレクトリまたはそのサブディレクトリ内に、C 実装のヘッダーファイルとソースファイルを配置します。例えば、次のディレクトリ内に、拡張の .cpp ファイルと .h ファイルを配置します。

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

同様に、***<yourPlatform>***/edk ディレクトリまたはそのサブディレクトリ内に、実際の ActionScript 実装の .as ファイルを配置します。次に、例を示します。

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

また、***<yourPlatform>***/edk ディレクトリまたはそのサブディレクトリ内に、ActionScript スタブ実装またはシミュレーター実装の .as ファイルを配置します。次に、例を示します。

```
<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
```

注意：AIR for TV に付属するサンプル拡張は、<AIR for TV installation directory>/products/stagecraft/source/edk ディレクトリ内にあります。このディレクトリに、拡張ファイルを配置しないでください。

.mk ファイルの作成

他の AIR for TV モジュールと同様に、拡張モジュールを構築するには、最初に .mk ファイルを作成します。.mk ファイルの主な目的は、構築するソースファイルを指定することです。

.mk ファイルを作成するには：

1 次の場所から PlatformEDKExtension_HelloWorld.mk ファイルまたは PlatformEDKExtension_Process.mk ファイルをコピーします。

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

または

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

コピー先は以下です。

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

このディレクトリは、プラットフォームの Makefile.config ファイルを含むディレクトリと同じです。『**Optimizing and Integrating Adobe AIR for TV**』（PDF）の「**Coding, building, and testing**」の章の「**Creating your platform Makefile.config**」を参照してください。

2 .mk ファイルの名前を PlatformEDKExtension_**<your extension name>**.mk に変更します。AIR for TV の make ユーティリティは、この命名規則で自動的に .mk ファイルを検索します。

mk ファイルの名前は必ず PlatformEDKExtension_ から開始してください。

3 「REQUIRED」のマークがついた .mk ファイルのセクションを編集します。

次の必須の変更を行います。

- SC_EDK_EXTENSION_NAME を拡張名に設定します。この変数を、PlatformEDKExtension_**<your extension name>**.mk の ***<your extension name>*** の値に設定します。
- SC_EDK_EXTENSION_PACKAGE を拡張パッケージ名に設定します。この値を、拡張の ActionScript 側で使用するパッケージ名に設定します。

  make ユーティリティは、この値を、拡張の拡張記述ファイル内にある <id> エレメントの値として使用します。また、作成される ANE ファイルにも、この値の名前と .ane ファイル名拡張子が付けられます。

  拡張記述ファイルについて詳しくは、71 ページの「ネイティブ拡張記述ファイル」を参照してください。
- SC_EDK_EXTENSION_VERSION を拡張のバージョン番号に設定します。

  make ユーティリティはこの値を、拡張の拡張記述ファイル内の <versionNumber> エレメントの値として使用します。
- SC_MODULE_SOURCE_DIR、SC_MODULE_SOURCE_FILES および SC_ADDITIONAL_MODULE_OBJ_SUBDIRS を、AIR for TV が提供するネイティブ実装ファイルを指定するように設定します。

  注意：このリストから ExtensionBridge.cpp を削除しないでください。HelloWorld または Process 拡張実装ファイルを削除します。通常は、このリストには拡張のソースファイルを追加しません。

  次に、例を示します。

  ```
  SC_MODULE_SOURCE_DIR := $(SC_SOURCE_DIR_EDK)
  SC_MODULE_SOURCE_FILES := ExtensionBridge.cpp
  ```
- SC_PLATFORM_SOURCE_DIR および SC_PLATFORM_SOURCE_FILES を、拡張のネイティブ実装ファイルを指定するように設定します。次に、例を示します。

  ```
  SC_PLATFORM_SOURCE_DIR:= $(SC_PLATFORM_MAKEFILE_DIR)/edk/myExtension/native
  SC_PLATFORM_SOURCE_FILES := \
                  MyExtension.cpp \
                  helper\MyHelperClass1.cpp \
                  helper\MyHelperClass2.cpp
  ```
- SC_EDK_AS_SOURCE_DIR を、（スタブ実装ではなく）実際の ActionScript ファイルを含むディレクトリに設定します。次に、例を示します。

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

  注意：このディレクトリは、ActionScript パッケージの基本ディレクトリです。例えば、tv.adobe.extension.example という名前の ActionScript パッケージについて考えます。この場合、tv、adobe、extension および example の各ディレクトリは、SC_EDK_AS_SOURCE_DIR の次に続くサブディレクトリです。
- SC_EDK_AS_CLASSES を、実際の ActionScript 実装が定義するすべての ActionScript クラスのリストに設定します。次に、例を示します。

```
SC_EDK_AS_CLASSES := MyExtension \
                     MyHelperClass1 \
                     MyHelperClass2
```

- SC_EDK_AS_SOURCE_DIR_AUTHORING を、拡張のスタブ実装またはシミュレーター実装の ActionScript ファイルを含むディレクトリに設定します。次に、例を示します。

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

  注意：このディレクトリは、ActionScript パッケージの基本ディレクトリです。例えば、tv.adobe.extension.example という名前の ActionScript パッケージについて考えます。この場合、tv、adobe、extension および example の各ディレクトリは、SC_EDK_AS_SOURCE_DIR_AUTHORING の次に続くサブディレクトリです。

- ActionScript スタブ実装またはシミュレーター実装が定義するすべての ActionScript クラスのリストに、SC_EDK_AS_CLASSES_AUTHORING を設定します。次に、例を示します。

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

サードパーティ製ライブラリのインストール

AIR for TV の構築にはいくつかのサードパーティ製ライブラリが必要です。これらのライブラリについて詳しくは、『**Getting Started with Adobe AIR for TV**』（PDF）の「**Install third-party software**」を参照してください。

拡張モジュールのみを構築し、AIR for TV 全体を構築するわけではない場合、必要なライブラリは次のとおりです。

- AIR 3 SDK

  http://www.adobe.com/jp/special/products/air/sdk/ から、Mac OS X ダウンロードを選択します。

  この .tbz2 ファイルの内容を展開するディレクトリを作成します。次に、例を示します。

  ```
  /usr/AIRSDK
  ```

  作成したディレクトリ内に、.tbz2 ファイルの内容を展開します。

  ```
  tar jxf AdobeAIRSDK.tbz2
  ```

  PATH 環境変数を、AIR SDK bin ディレクトリを含めるように設定します。この例では、bin ディレクトリは /usr/AIRSDK/bin です。

- オープンソース Flex® SDK

  http://opensource.adobe.com/wiki/display/flexsdk/Downloads からオープンソース Flex SDK の最新リリースビルドの ZIP ファイルをダウンロードします。

  ZIP ファイルのコンテンツを格納するディレクトリを作成します。次に、例を示します。

  ```
  /usr/flexSDK
  ```

  作成したディレクトリ内に、ZIP ファイルの内容を展開します。

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

  PATH 環境変数を、Flex SDK bin ディレクトリを含めるように設定します。この例では、bin ディレクトリは /usr/flexSDK/bin です。

- Java™ ランタイム。Flex SDK では最新の Java ランタイムが必要です。開発システムにまだ Java ランタイムがない場合は、http://www.java.com/en/download/manual.jsp のダウンロードおよびインストール手順を参照してください。

  PATH 環境変数を、Java bin ディレクトリを含めるように設定します。

make ユーティリティの実行

AIR for TV の make ユーティリティの使用について詳しくは、次のドキュメントを参照してください。

- 『Getting Started with Adobe AIR for TV』（PDF）の「Installing and building the source distribution」の章。
- 『Optimizing and Integrating Adobe AIR for TV』（PDF）の「Coding, building, and testing」の章。「Creating your platform Makefile.config」の節の説明に従って、様々なビルド変数を設定します。

特に、make ユーティリティは、拡張の構築時に Makefile.config で次のビルド変数を使用します。

- SC_ZIP
- SC_UNZIP
- SC_PLATFORM_NAME
- SC_PLATFORM_ARCH

プラットフォームの Makefile.config ファイルおよび拡張の .mk ファイルを作成した後で、make ユーティリティを使用して次のことを実行できます。

- AIR for TV 全体の構築
- 拡張モジュールのみの構築

AIR for TV 全体を構築するには、以下を実行します。

1. 環境変数 SC_BUILD_MODE および SC_PLATFORM が設定されていることを確認します。
2. 作成した証明書を使用して拡張に署名する場合は、環境変数 SC_EDK_ANE_CERT_FILE および SC_EDK_ANE_CERT_PASSWD を設定します。

   SC_EDK_ANE_CERT_FILE を、証明書の相対パスまたは絶対パスに設定します。この相対パスは、ビルドディレクトリの <**AIR for TV installation directory**>/stagecraft/build/linux からの相対パスです。

   SC_EDK_ANE_CERT_PASSWD を、証明書のパスワードに設定します。

   これらの環境変数を設定しなければ、make ユーティリティはデフォルトの偽の証明書を使用し、警告メッセージを表示します。この偽の証明書は、テストのみに適しています。
3. 作業ディレクトリを次の場所に変更します。

   ```
   <AIR for TV installation directory>/products/stagecraft/build/linux
   ```
4. 次のコマンドを入力します。

   ```
   make
   ```

拡張モジュールのみを構築するには、以下を実行します。

1. 環境変数 SC_BUILD_MODE および SC_PLATFORM が設定されていることを確認します。
2. 作成した証明書を使用して拡張に署名する場合は、前述のとおり、環境変数 SC_EDK_ANE_CERT_FILE および SC_EDK_ANE_CERT_PASSWD を設定します。
3. ディレクトリを stagecraft/build/linux ディレクトリに変更します。
4. 次のコマンドを入力します。

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

次のコマンドを使用して、拡張に対して前に構築されたすべてのオブジェクトを削除できます。

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

次のコマンドを使用して、拡張に対して前に構築されたすべてのオブジェクトを削除し、再構築できます。

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

重要：ビルドマシンがファイアウォールの背後にある場合は、make ユーティリティが失敗することがあります。ファイアウォールでは、ADT が使用するタイムスタンプサーバー（ネイティブ拡張を ANE ファイルにパッケージ化する際のタイムスタンプサーバー）へのアクセスが禁止されていることがあります。この失敗が発生した場合、次のエラーが出力されます。

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

この失敗を回避するには、make ユーティリティが使用する ADT コマンドを変更します。次のディレクトリ内の extension.mk ファイルを編集します。

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

次の行を検索します。

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

次のように、コマンドに -tsa none パラメーターを追加します。

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

make ユーティリティの拡張の出力

make ユーティリティは拡張の 2 つのファイルを作成します。これらのファイルは、SC_BUILD_MODE にデバッグとリリースのどちらを指定したかに応じて、次のディレクトリのいずれかに配置されます。

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

make ユーティリティが拡張向けに作成するファイルは次のとおりです。

- デバイスにデプロイするための、デバイスバンドル拡張を含む ZIP ファイル。
- スタブ拡張またはシミュレーター拡張を含む ANE ファイル。AIR アプリケーション開発者はこの ANE ファイルを使用して、アプリケーションを構築します。また、ADL を使用してデスクトップコンピューター上でアプリケーションをテストするためにも、このファイルを使用します。さらに、アプリケーションと共に、この ANE ファイルを AIRN パッケージにパッケージ化します。

スタブ拡張とシミュレーター拡張の両方の構築

実際の拡張に加えて、スタブ拡張とシミュレーター拡張の両方を構築する場合があります。通常、AIR アプリケーション開発者に次のように指示します。

- シミュレーター拡張を使用してデスクトップ上でテストすること
- アプリケーションと共に、スタブ拡張を AIRN パッケージにパッケージ化すること

スタブ拡張とシミュレーター拡張の両方を構築するには、以下を実行します。

1. スタブ拡張とその .mk ファイルを作成します。スタブ拡張と実際の拡張を構築できることを確認します。
2. スタブ実装ディレクトリと同じ階層に、シミュレーター実装のディレクトリを作成します。次に、例を示します。

   ```
   <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. 拡張の .mk ファイルをコピーします。
4. このコピーで、SC_EDK_AS_SOURCE_DIR_AUTHORING と SC_EDK_AS_CLASSES_AUTHORING の値を編集します。これらの値を、シミュレーター実装のディレクトリおよびクラスを反映するように設定します。
5. 拡張の元の .mk ファイルの名前を変更し、安全に保護します。次に、コピーの名前を、拡張用の .mk ファイル名（PlatformEDKExtension_<your extension name>.mk）に変更します。

6 プラットフォームの bin ディレクトリにあるスタブ ANE ファイルを、安全な場所に移動します。この手順を省くと、次の手順でファイルが上書きされます。

7 make ユーティリティを実行し、実際の拡張およびシミュレーター拡張を構築します。

AIR for TV ネイティブ拡張へのリソースの追加

AIR for TV ネイティブ拡張の中には、画像ファイルなどのリソースが必要なものがあります。拡張でリソースを使用する場合は、以下を実行します。

1 make ユーティリティが作成した ZIP ファイル内のファイルを、ディレクトリ構造を維持して解凍します。

2 このディレクトリ構造に、拡張で必要となるリソースディレクトリおよびファイルを追加します。

3 更新されたディレクトリ構造を、make ユーティリティが作成した ZIP ファイルと同じ名前で、ZIP ファイルに再度圧縮します。

4 make ユーティリティが作成した ANE ファイル内のファイルを、ディレクトリ構造を維持して解凍します。ZIP ファイルの処理に使用するのと同じツールを使用できます。

5 このディレクトリ構造に、拡張で必要となるリソースディレクトリおよびファイルを追加します。

6 更新されたディレクトリ構造を ZIP ファイルに再度圧縮します。ファイルの名前を、make ユーティリティが作成した ANE ファイルと同じ名前に変更します。ファイル名の拡張子は .ane に変更してください。

拡張のディレクトリ構造内での、リソース用の正確なサブディレクトリ構造は、拡張がリソースを検索する場所によって異なります。拡張の ActionScript 側では、ExtensionContext.getExtensionDirectory() を使用して拡張のディレクトリを検索できます。詳しくは、12 ページの「ネイティブ拡張のディレクトリへのアクセス」を参照してください。

AIR for TV ネイティブ拡張の配布

AIR for TV デバイスへのデバイスバンドル拡張のインストール

デバイスバンドル拡張を構築した後に、デバイス上にそれをインストールします。

make ユーティリティによって、デバイス上にインストールするすべてのファイルを含む ZIP ファイルが作成されています。このファイルを、デバイス上の次の場所に解凍します。

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

注意：/opt/adobe/stagecraft/extensions ディレクトリは、デバイスファイルシステム上の他のディレクトリに対する Linux シンボリックソフトリンクにすることもできます。詳しくは、『**Optimizing and Integrating Adobe AIR for TV**』（PDF）の「Filesystem usage」の章を参照してください。

AIR アプリケーションとのスタブ拡張またはシミュレーター拡張のパッケージ化

AIR アプリケーションでデバイス上のデバイスバンドル拡張を使用するには、AIR アプリケーションと共にスタブ拡張またはシミュレーター拡張をパッケージ化します。ADT を使用して、AIR アプリケーションと共にスタブ拡張またはシミュレーター拡張の ANE ファイルをパッケージ化し、AIRN パッケージファイルを作成します。AIRN パッケージファイルは AIR パッケージファイルのようなものですが、AIRN パッケージファイルには拡張 ANE ファイルが含まれます。

詳しくは、「**AIR for TV アプリケーションのパッケージ化**」を参照してください。

AIR for TV デバイス上での AIR アプリケーションの実行

AIR for TV デバイス上で AIR アプリケーションをインストールし実行する方法については、『**Getting Started with Adobe AIR for TV**』（PDF）を参照してください。stagecraft バイナリ実行ファイルを実行する際には、次のコマンドラインオプションを使用してください。

```
--profile extendedTV
```

AIR アプリケーションがネイティブ拡張を使用できるようにするには、このオプションは必須です。

AIR for TV が、スタブ拡張またはシミュレーター拡張を含むアプリケーションを実行すると、AIR for TV はデバイス上の対応するデバイスバンドル拡張を検索します。見つかった場合は、AIR アプリケーションはその拡張を使用します。デバイスバンドル拡張がデバイスにインストールされていない場合は、AIR アプリケーションはスタブ拡張またはシミュレーター拡張を使用します。

第 7 章：ネイティブ拡張記述ファイル

拡張記述ファイルは、ネイティブ拡張パッケージのコンテンツについて記述するものです。

拡張記述ファイルの例

以下に示した拡張記述ドキュメントの例では、次のものに対応するネイティブ拡張が記述されています。

- Android デバイス
- その他のプラットフォーム向けの default ActionScript 実装

```
<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>
```

拡張記述ファイルの構造

拡張記述ファイルは、次の構造を持つ XML ドキュメントです。

```
<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>
```

ネイティブ拡張記述エレメント

以下のエレメントの辞書では、AIR アプリケーション記述ファイルの有効なエレメントについて説明します。

applicationDeployment

拡張パッケージ内に含まれ、その結果アプリケーションと共にデプロイされる、ネイティブコードライブラリを宣言します。

platform エレメントごとに、applicationDeployment エレメントまたは deviceDeployment エレメントのいずれかを含める必要があります。ただし、両方は含めないでください。

親エレメント：platform

子エレメント：

- finalizer
- initializer
- nativeLibrary

コンテンツ

ネイティブコードライブラリ、初期化関数およびファイナライズ処理関数を指定します。プラットフォーム名が default の場合、applicationDeployment エレメントには子エレメントはありません。これは、default プラットフォームにネイティブコードライブラリがないからです。

例

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

copyright

オプション

拡張の著作権に関する宣言。

親エレメント：extension

子エレメント：なし

コンテンツ

著作権情報を含む文字列。

例

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

description

オプション

拡張の記述。

親エレメント：extension

子エレメント：text

コンテンツ
単純なテキストノードか、複数の text エレメントを使用します。

複数の text エレメントを使用すると、description エレメントで複数の言語を指定できます。各テキストエレメントの xml:lang 属性は、RFC4646（http://www.ietf.org/rfc/rfc4646.txt）に定義された言語コードを指定します。

例
単純な text ノードを使用する記述：

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

英語、フランス語およびスペイン語でローカライズされた text エレメントを使用する記述：

```
<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

コードライブラリがデバイス上に個別にデプロイされ、この拡張パッケージには含まれないネイティブ拡張を宣言します。

デバイスのデプロイは一部のプラットフォームではサポートされていません。

platform エレメントごとに、applicationDeployment エレメントまたは deviceDeployment エレメントのいずれかを含める必要があります。ただし、両方は含めないでください。

親エレメント：platform

子エレメント：なし

コンテンツ
なし。deviceDeployment エレメントは空である必要があります。

例

```
<deviceDeployment/>
```

extension

必須

拡張記述ドキュメントのルートエレメント。

親エレメント：なし

子エレメント：

- copyright
- description
- id
- name

- platforms
- versionNumber

コンテンツ
サポートされるプラットフォームと、プラットフォームごとのコードライブラリを指定します。

extension エレメントには、xmlns という名前空間属性が含まれています。xmlns 値には、次の値のいずれか 1 つを指定できます。

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

SWF のバージョン同様、名前空間は、AIR SDK と ANE ファイル間の互換性を決定する要素の 1 つです。AIR アプリケーションは、拡張の名前空間と同じかそれ以降のバージョンの AIR SDK でパッケージ化される必要があります。そのため、AIR 3 アプリケーションは、2.5 の名前空間の拡張を使用できますが、3.1 の名前空間の拡張を使用することはできません。

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

オプション

ネイティブライブラリで定義されたファイナライズ処理関数。

親エレメント：applicationDeployment

子エレメント：なし

コンテンツ
拡張のネイティブライブラリで C API が使用されている場合のファイナライザー関数の名前。

拡張で Java API を使用している場合、このエレメントには、FREExtension インターフェイスを実装するクラスの名前が含まれます。

値に含めることができる文字は、A - Z、a - z、0 - 9、ピリオド（.）およびハイフン（-）です。

例

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

id

必須

拡張の ID。

親エレメント：extension

子エレメント：なし

コンテンツ
拡張の ID を指定します。

値に含めることができる文字は、A - Z、a - z、0 - 9、ピリオド（.）およびハイフン（-）です。

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

initializer

オプション

ネイティブライブラリで定義された初期化関数。nativeLibrary エレメントが使用されている場合は、initializer エレメントは必須です。

親エレメント：applicationDeployment

子エレメント：なし

コンテンツ
拡張のネイティブライブラリで C API が使用されている場合の初期化関数の名前。

拡張で Java API を使用している場合、このエレメントには、FREExtension インターフェイスを実装するクラスの名前が含まれます。

値に含めることができる文字は、A - Z、a - z、0 - 9、ピリオド（.）およびハイフン（-）です。

例
```
<initializer>...</initializer>
```

name

オプション

拡張の名前。

親エレメント：extension

子エレメント：text

コンテンツ
1 つのテキストノードを指定した場合（複数の <text> エレメントを指定しない場合）、システム言語に関係なく、この名前が AIR アプリケーションインストーラーで使用されます。

各テキストエレメントの xml:lang 属性は、RFC4646（http://www.ietf.org/rfc/rfc4646.txt）に定義された言語コードを指定します。

例

次に、簡単な text ノードを使用して名前を定義する例を示します。

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

次の例では <text> エレメントノードを使用して、3 つの言語（英語、フランス語およびスペイン語）で名前を指定します。

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

nativeLibrary

オプション

あるプラットフォーム向けの拡張パッケージに含まれるネイティブライブラリファイル。以下を考慮してください。

- 拡張に含まれるのが ActionScript コードのみの場合は、nativeLibrary エレメントは必須ではありません。
- nativeLibrary エレメントが使用されない場合は、initializer エレメントと finalizer エレメントも使用できません。
- nativeLibrary エレメントが使用されている場合は、initializer エレメントも必要です。

親エレメント：applicationDeployment

子エレメント：なし

コンテンツ

拡張パッケージに含まれるネイティブライブラリのファイル名。

値に含めることができる文字は、A - Z、a - z、0 - 9、ピリオド（.）およびハイフン（-）です。

例

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

platform

必須

特定のプラットフォームの拡張向けネイティブコードライブラリを指定します。

親エレメント：platforms

子エレメント：次のエレメントのいずれかのみ

- applicationDeployment
- deviceDeployment

コンテンツ

name 属性ではプラットフォームの名前を指定します。特別な default プラットフォーム名を使用することで、拡張の開発者は、サポート対象外のプラットフォームでのネイティブコードの動作をシミュレートする ActionScript ライブラリを含めることができます。シミュレートされた動作は、デバッグをサポートし、複数プラットフォーム対応アプリケーションのフォールバック動作を提供するために使用できます。

name 属性には次の値を使用します。

- Android-ARM（Android デバイスの場合）

- default
- iPhone-ARM（iOS デバイスの場合）
- iPhone-x86（iOS シミュレーターの場合）
- MacOS-x86-64（Mac OS X デバイスの場合）
- QNX-ARM（Blackberry Tablet OS デバイスの場合）
- Windows-x86（Windows デバイスの場合）

注意：デバイスバンドル拡張では、デバイス製造元が定義している name 属性値を使用します。

子エレメントでは、ネイティブコードライブラリのデプロイ方法を指定します。アプリケーションデプロイは、コードライブラリが、それを使用する各 AIR アプリケーションと共にデプロイされることを意味します。コードライブラリは拡張パッケージに含める必要があります。デバイスデプロイは、コードライブラリがプラットフォームに個別にデプロイされ、拡張パッケージに含まれないことを意味します。この 2 つのデプロイタイプは相互に排他的です。つまり、デプロイエレメントは 1 つしか含めることができません。

例

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

platforms

必須

この拡張でサポートされるプラットフォームを指定します。

親エレメント：extension

子エレメント：platform

コンテンツ

サポートされるプラットフォームごとに platform エレメントを指定します。オプションで、特定のコードライブラリでサポートされていないプラットフォーム上で使用する ActionScript 実装を含める、特別な「default」プラットフォームを指定できます。

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

オプション

ローカライズされたストリングを指定します。

テキストエレメントの xml:lang 属性は、RFC4646（http://www.ietf.org/rfc/rfc4646.txt）に定義された言語コードを指定します。

AIR では、オペレーティングシステムのユーザーインターフェイス言語に最も近い xml:lang 属性値を持つ text エレ ントが使用されます。

例えば、text エレメントに en（英語）ロケール の値が含まれているインストールを行うとします。オペレーティングシステムのユーザーインターフェイス言語が en（英語）になっている場合は、AIR で名前 en が使用されます。システムのユーザーインターフェイス言語が en-US（アメリカ英語）の場合にも名前 en が使用されます。ただし、ユーザーインターフェイス言語が en-US で、アプリケーション記述ファイルに en-US と en-GB の両方が定義されている場合は、AIR アプリケーションインストーラーで en-US が使用されます。

システムのユーザーインターフェイス言語に一致する text エレメントがアプリケーションに定義されていない場合、AIR では拡張記述ファイルに定義された最初の name の値が使用されます。

親エレメント：

- name
- description

子エレメント：なし

コンテンツ

ロケールとローカライズされたテキストのストリングを示す xml:lang 属性。

例

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

versionNumber

必須

拡張のバージョン番号。

親エレメント：extension

子エレメント：なし

コンテンツ

バージョン番号には、3 個以下の連結した整数をピリオドで区切って含めることができます。各整数には、0 以上 999 以下の数を指定する必要があります。

例

```
<versionNumber>1.0.657</versionNumber>

<versionNumber>10</versionNumber>

<versionNumber>0.01</versionNumber>
```

第 8 章：ネイティブ C API リファレンス

ネイティブ C API を使用したネイティブ拡張の例については、「Adobe AIR 用のネイティブ拡張」を参照してください。

typedef

FREContext

AIR 3.0 以降

```
typedef void* FREContext;
```

ActionScript 側は、ExtensionContext.createExtensionContext() を呼び出すことで、拡張コンテキストを作成します。AIR ランタイムは、拡張コンテキストごとに、対応する FREContext 変数を作成します。

AIR ランタイムは、ネイティブ実装の次の関数に FREContext 変数を渡します。

- FREContextInitializer()（コンテキスト初期化関数）。
- FREContextFinalizer()（コンテキストファイナライザー関数）。
- FREFunction()（各拡張の関数）。

この FREContext 変数は、次のときに使用します。

- ActionScript ExtensionContext インスタンスにイベントを送出するとき。FREDispatchStatusEventAsync() を参照してください。
- ネイティブコンテキストデータを取得または設定するとき。97 ページの「FREGetContextNativeData()」および 112 ページの「FRESetContextNativeData()」を参照してください。
- ActionScript コンテキストデータを取得または設定するとき。97 ページの「FREGetContextActionScriptData()」および 112 ページの「FRESetContextActionScriptData()」を参照してください。

FREObject

AIR 3.0 以降

```
typedef void* FREObject;
```

ネイティブ C 実装から ActionScript クラスオブジェクトまたはプリミティブデータ型変数にアクセスする場合は、FREObject 変数を使用します。FREObject 変数とそれに対応する ActionScript オブジェクトとの関連付けはランタイムによって行われます。

FREObject 変数は、実装するネイティブ関数で、FREFunction() シグネチャと共に使用されます。ネイティブ関数は次の処理を実行します。

- FREObject 変数をパラメーターとして取得します。
- FREObject 変数を返します。

FREObject 変数は、ネイティブ拡張 C API 関数を使用して次の処理を行う場合にも使用されます。

- ActionScript のクラスオブジェクトまたは ActionScript のプリミティブデータ型を作成します。
- ActionScript のクラスオブジェクトまたは ActionScript のプリミティブデータ型の値を取得します。

- ActionScript の String オブジェクトを作成します。
- ActionScript の String オブジェクトの値を取得します。
- ActionScript オブジェクトのプロパティを取得または設定します。
- ActionScript オブジェクトのメソッドを呼び出します。
- ActionScript の BitmapData オブジェクトのビットにアクセスします。
- ActionScript の ByteArray オブジェクトのバイトにアクセスします。
- ActionScript の Array または Vector オブジェクトの長さを取得または設定します。
- ActionScript の Array または Vector オブジェクトのエレメントを取得または設定します。
- ネイティブ拡張 C API 関数呼び出しでランタイムが例外をスローした場合に、ActionScript の Error オブジェクトを取得します。
- ActionScript のコンテキストデータの ActionScript オブジェクトを設定または取得します。

構造体 typedef

FREBitmapData

AIR 3.0

ActionScript BitmapData クラスオブジェクトのビットを取得し操作する場合は、FREBitmapData または FREBitmapData2 構造体を使用します。FREBitmapData 構造体は次のように定義されます。

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

次に、FREBitmapData のフィールドの意味を示します。

width uint32_t。ビットマップの幅をピクセル単位で示します。この値は、ActionScript BitmapData クラスオブジェクトの width プロパティに対応します。このフィールドは読み取り専用です。

height uint32_t。ビットマップの高さをピクセル単位で示します。この値は、ActionScript BitmapData クラスオブジェクトの height プロパティに対応します。このフィールドは読み取り専用です。

hasAlpha uint32_t。ビットマップがピクセル単位の透明度をサポートするかどうかを示します。この値は、ActionScript BitmapData クラスオブジェクトの transparent プロパティに対応します。値がゼロ以外であれば、ピクセルのフォーマットは ARGB32 です。値がゼロであれば、ピクセルのフォーマットは _RGB32 です。値がビッグエンディアンになるかリトルエンディアンになるかは、ホストデバイスによって異なります。このフィールドは読み取り専用です。

isPremultiplied uint32_t。ビットマップピクセルが乗算済みカラー値として保存されるかどうかを示します。値がゼロ以外の場合、値は乗算済みです。このフィールドは読み取り専用です。乗算済みカラー値について詳しくは、『**Adobe Flash Platform** 用 **ActionScript 3.0** リファレンスガイド』の「BitmapData.getPixel()」を参照してください。

lineStride32 uint32_t。スキャンラインあたりの uint32_t 値の数を示します。この値は通常、width パラメーターと同じです。このフィールドは読み取り専用です。

bits32 uint32_t へのポインター。この値は uint32_t 値の配列です。それぞれの値は、ビットマップの 1 ピクセルです。

注意：ネイティブ実装で変更できる FREBitmapData 構造体のフィールドは、bits32 フィールドだけです。bits32 フィールドには、実際のビットマップ値が含まれます。FREBitmapData 構造体の他のすべてのフィールドは、読み取り専用フィールドとして扱います。

関連項目

82 ページの「FREBitmapData2」

90 ページの「FREAcquireBitmapData()」

103 ページの「FREInvalidateBitmapDataRect()」

FREBitmapData2

AIR 3.1 以降

FREBitmapData2 構造体は、isInvertedY フィールドを FREBitmapData 構造体に追加します。それ以外の点では、2 つの構造体は同一です。FREBitmapData2 構造体は次のように定義されます。

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

次に、FREBitmapData2 のフィールドの意味を示します。

width uint32_t。ビットマップの幅をピクセル単位で示します。この値は、ActionScript BitmapData クラスオブジェクトの width プロパティに対応します。このフィールドは読み取り専用です。

height uint32_t。ビットマップの高さをピクセル単位で示します。この値は、ActionScript BitmapData クラスオブジェクトの height プロパティに対応します。このフィールドは読み取り専用です。

hasAlpha uint32_t。ビットマップがピクセル単位の透明度をサポートするかどうかを示します。この値は、ActionScript BitmapData クラスオブジェクトの transparent プロパティに対応します。値がゼロ以外であれば、ピクセルのフォーマットは ARGB32 です。値がゼロであれば、ピクセルのフォーマットは _RGB32 です。値がビッグエンディアンになるかリトルエンディアンになるかは、ホストデバイスによって異なります。このフィールドは読み取り専用です。

isPremultiplied uint32_t。ビットマップピクセルが乗算済みカラー値として保存されるかどうかを示します。値がゼロ以外の場合、値は乗算済みです。このフィールドは読み取り専用です。乗算済みカラー値について詳しくは、『**Adobe Flash Platform** 用 **ActionScript 3.0** リファレンスガイド』の「BitmapData.getPixel()」を参照してください。

lineStride32 uint32_t。スキャンラインあたりの uint32_t 値の数を示します。この値は通常、width パラメーターと同じです。このフィールドは読み取り専用です。

isInvertedY uint32_t。画像のビットマップデータ行が格納される順番を示します。値がゼロ以外の場合、画像の一番下の行が画像データでは最初に表示されます（つまり、bit32 配列の最初の値は、画像最後の行の最初のピクセルです）。値がゼロの場合、画像の一番上の行が画像データの最初に表示されます。このフィールドは読み取り専用です。

bits32 uint32_t へのポインター。この値は uint32_t 値の配列です。それぞれの値は、ビットマップの 1 ピクセルです。

注意：ネイティブ実装で変更できる FREBitmapData2 構造体のフィールドは、bits32 フィールドだけです。bits32 フィールドには、実際のビットマップ値が含まれます。FREBitmapData2 構造体の他のすべてのフィールドは、読み取り専用フィールドとして扱います。

関連項目

81 ページの「FREBitmapData」

91 ページの「FREAcquireBitmapData2()」

103 ページの「FREInvalidateBitmapDataRect()」

FREByteArray

AIR 3.0 以降

ActionScript ByteArray クラスオブジェクトのバイトを取得し操作する場合には、FREByteArray 構造体を使用します。構造体は次のように定義されます。

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

FREByteArray のフィールドの意味は次のとおりです。

length uint32_t。bytes 配列内のバイト数を示します。

bytes uint8_t*。ActionScript ByteArray オブジェクト内のバイト数へのポインターです。

関連項目

92 ページの「FREAcquireByteArray()」

FRENamedFunction

AIR 3.0 以降

記述した FREFunction を名前に関連付けるには、FRENamedFunction を使用します。ExtensionContext インスタンスの call() メソッドを使用してネイティブ関数を呼び出すときに、ActionScript コード内でこの名前を使用します。構造体は次のように定義されます。

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

FRENamedFunction のフィールドの意味は次のとおりです。

name uint8_t* の定数。このポインターは、関連付けられた C 関数を呼び出すために ActionScript 側で使用する文字列をポイントします。つまり、この文字列値は、ActionScript ExtensionContext call() メソッドが functionName パラメーターで使用する名前です。この文字列には UTF-8 エンコーディングを使用し、null 文字で終了します。

functionData void*。このポインターは、この FREFunction 関数と関連付ける任意のデータをポイントします。ランタイムが FREFunction 関数を呼び出すときに、関数にこのデータポインターを渡します。

function FRENamedFunction。ランタイムが name フィールドで指定された文字列に関連付ける関数です。FREFunction() のシグネチャを使用してこの関数を定義します。

列挙型

FREObjectType

AIR 3.0 以降

FREObject 変数は、ActionScript クラスオブジェクトまたはプリミティブ型に対応します。FREObjectType 列挙型は、これらの ActionScript クラス型およびプリミティブ型の値を定義します。C API 関数の FREGetObjectType() は、FREObject 変数の対応する ActionScript クラスオブジェクトまたはプリミティブ型を最も適切に表現する、FREObjectType 列挙値を返します。

```
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
};
```

この列挙値の意味は次のとおりです。

FRE_TYPE_OBJECT FREObject 変数は、String オブジェクト、ByteArray オブジェクト、Array オブジェクト、Vector オブジェクトまたは BitmapData オブジェクト以外の ActionScript クラスオブジェクトに対応します。

FRE_TYPE_NUMBER FREObject 変数は、ActionScript Number 変数に対応します。

FRE_TYPE_STRING FREObject 変数は、ActionScript String オブジェクトに対応します。

FRE_TYPE_BYTEARRAY FREObject 変数は、ActionScript ByteArray オブジェクトに対応します。

FRE_TYPE_ARRAY FREObject 変数は、ActionScript Array オブジェクトに対応します。

FRE_TYPE_VECTOR FREObject 変数は、ActionScript Vector オブジェクトに対応します。

FRE_TYPE_BITMAPDATA FREObject 変数は、ActionScript BitmapData オブジェクトに対応します。

FRE_TYPE_BOOLEAN FREObject 変数は、ActionScript Boolean 変数に対応します。

FRE_TYPE_NULL FREObject 変数は、ActionScript 値の Null または undefined に対応します。

FREObjectType_ENUMPADDING この最後の列挙値は、列挙値のサイズが常に 4 バイトになるようにします。

関連項目

21 ページの「FREObject 型」

FREResult

AIR 3.0 以降

FREResult 列挙型は、呼び出すネイティブ拡張 C API 関数の戻り値を定義します。

```
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
};
```

この列挙値の意味は次のとおりです。

FRE_OK 関数は成功しました。

FRE_ACTIONSCRIPT_ERROR ActionScript エラーが発生し、例外がスローされました。このエラーが発生する可能性のある C API 関数では、FREObject を指定して例外に関する情報を取得できます。

FRE_ILLEGAL_STATE ネイティブ拡張 C API 関数に対する呼び出しが、拡張コンテキストがその呼び出しをするには不正な状態であったときに行われました。この戻り値は、次の状況で発生します。コンテキストが、ActionScript BitmapData または ByteArray クラスオブジェクトへのアクセスを取得しました。1 つの例外を除いて、コンテキストはこの BitmapData または ByteArray オブジェクトを解放するまでは、他の C API 関数を呼び出すことができません。例外としては、コンテキストは FREAcquireBitmapData() または FREAcquireBitmapData2() を呼び出した後で、FREInvalidateBitmapDataRect() を呼び出すことができます。

FRE_INSUFFICIENT_MEMORY ランタイムは Array または Vector オブジェクトのサイズを変更するのに十分なメモリを割り当てることができませんでした。

FRE_INVALID_ARGUMENT ポインターパラメーターが NULL です。

FRE_INVALID_OBJECT FREObject パラメーターが無効です。無効な FREObject 変数の例については、22 ページの「FREObject の有効性」を参照してください。

FRE_NO_SUCH_NAME パラメーターとして渡したクラス、プロパティまたはメソッドの名前が、ActionScript クラス名、プロパティまたはメソッドと一致しません。

FRE_READ_ONLY 関数は ActionScript オブジェクトの読み取り専用プロパティを修正しようとしました。

FRE_TYPE_MISMATCH FREObject パラメーターは、呼び出された関数が予期する ActionScript クラスのオブジェクトを表しません。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

FREResult_ENUMPADDING この最後の列挙値は、列挙値のサイズが常に 4 バイトになるようにします。

実装する関数

拡張 C API には、拡張のネイティブ C 実装で実装する関数のシグネチャが用意されています。

FREContextFinalizer()

AIR 3.0 以降

シグネチャ

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

パラメーター

ctx この拡張コンテキストを表す FREContext 変数。80 ページの「FREContext」を参照してください。

戻り値

なし。

説明

ランタイムは、この拡張コンテキストの ExtensionContext インスタンスを破棄するときに、この関数を呼び出します。ランタイムがインスタンスを破棄するのは、次のような場合です。

- ActionScript 側で、ExtensionContext インスタンスの dispose() メソッドを呼び出した場合
- ランタイムのガベージコレクターが、ExtensionContext インスタンスへの参照が存在しないことを検知した場合
- AIR アプリケーションが終了中の場合

拡張のこのコンテキスト固有のリソースをクリーンアップするには、この関数を実装します。ネイティブコンテキストデータおよび ActionScript コンテキストデータに関連付けられたリソースを取得してクリーンアップするには、ctx パラメーターを使用します。18 ページの「コンテキスト固有データ」を参照してください。

ランタイムがこの関数を呼び出した後、この拡張コンテキストの他の関数は呼び出されません。

関連項目

17 ページの「拡張コンテキストの初期化」

18 ページの「拡張コンテキストのファイナライズ処理」

FREContextInitializer()

AIR 3.0 以降

シグネチャ

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

パラメーター

extData ネイティブ拡張の拡張データへのポインター。FREInitializer() 関数がこの拡張データを作成しました。

ctxType コンテキストタイプを特定する文字列。拡張で、必要に応じてこの文字列を定義します。コンテキストタイプでは、拡張の ActionScript 側とネイティブ側とで合意した意味を指定できます。拡張がコンテキストタイプを使用しない場合は、この値に Null を設定できます。この値は、null の終端文字を含む UTF-8 エンコーディングの文字列です。

ctx FREContext 変数。ランタイムがこの値を作成し、FREContextInitializer() に渡します。80 ページの「FREContext」を参照してください。

numFunctionsToSet unint32_t へのポインター。numFunctionsToSet を、functionsToSet パラメーターの関数の数を含む unint32_t 変数に設定します。

functionsToSet FRNamedFunction エレメントの配列へのポインター。各エレメントにはネイティブ関数と、ActionScript 側で ExtensionContext インスタンスの call() メソッド内で使用する文字列へのポインターが含まれています。83 ページの「FRENamedFunction」を参照してください。

戻り値

なし。

説明

ActionScript 側で ExtensionContext.createExtensionContext() を呼び出すと、ランタイムがこのメソッドを呼び出します。このメソッドを実装する目的は次のとおりです。

- 拡張コンテキストを初期化します。この初期化方法は、ctxType パラメーターで渡されたコンテキストタイプによって異なります。
- ctx パラメーターの値を保存して、ネイティブ実装による FREDispatchStatusEventAsync() に対する呼び出しで利用できるようにします。
- ctx パラメーターを使用して、コンテキスト固有データを初期化します。このデータには、コンテキスト固有のネイティブデータおよびコンテキスト固有の ActionScript データが含まれます。
- FRENamedFunction オブジェクトの配列を設定します。functionsToSet パラメーター内の配列へのポインターを返します。numFunctionsToSet パラメーター内の配列エレメント数へのポインターを返します。

FREContextInitializer() メソッドの動作は、ctxType パラメーターによって異なります。ActionScript 側では、ExtensionContext.createExtensionContext() にコンテキストタイプを渡すことができます。次に、ランタイムがこの値を FREContextInitializer() に渡します。通常、この関数はこのコンテキストタイプを使用して、ActionScript 側で呼び出すことのできる、ネイティブ実装内のメソッドセットを選択します。各コンテキストタイプは様々なメソッドセットに対応します。

関連項目

9 ページの「コンテキストタイプ」

17 ページの「拡張コンテキストの初期化」

FREFinalizer()

AIR 3.0 以降

シグネチャ

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

パラメーター
extData 拡張の拡張データへのポインター。

戻り値
なし。

説明
ランタイムは、拡張をアンロードするときにこの関数を呼び出します。ただし、必ずしもランタイムが拡張をアンロードしたり、FREFinalizer() を呼び出したりするとは限りません。

拡張リソースをクリーンアップするには、この関数を実装します。

関連項目
89 ページの「FREInitializer()」

19 ページの「拡張のファイナライズ処理」

FREFunction()

AIR 3.0 以降

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

パラメーター
ctx この拡張コンテキストを表す FREContext 変数。80 ページの「FREContext」を参照してください。

functionData void*。このポインターは、FRENamedFunction 構造体で FREFunction 関数に関連付けたデータをポイントします。FREContextInitializer() の実装は、出力パラメーター内で、FRENamedFunction 構造体のセットをランタイムに渡します。ランタイムが FREFunction 関数を呼び出すときに、関数にこのデータポインターを渡します。83 ページの「FRENamedFunction」を参照してください。

argc argv パラメーター内のエレメント数。

argv FREObject 変数の配列。これらのポインターは、ExtensionContext インスタンスの call() メソッドへの ActionScript の呼び出しで、関数名の後に渡されるパラメーターに対応します。

戻り値
FREObject 変数。デフォルトの戻り値は、FRE_INVALID_OBJECT 型の FREObject です。

説明
拡張内のネイティブ関数ごとに、FREFunction シグネチャを使用して関数を実装します。関数名は、FREContextInitializer() 関数の functionsToSet パラメーターに返される配列内の、FRENamedFunction エレメントの function フィールドに対応します。

ActionScript 側で ExtensionContext インスタンスの call() メソッドを呼び出すと、ランタイムがこの FREFunction 関数を呼び出します。call() の最初のパラメーターは、FRENamedFunction エレメントの name フィールドと同じです。それ以降の call() パラメーターは、argv 配列内の FREObject 変数に対応します。

FREObject 変数を返すには、FREFunction を定義します。FREObject 変数の型は、call() メソッドが返す ActionScript の型に対応します。FREFunction が戻り値を持たない場合、デフォルトの戻り値は FRE_INVALID_OBJECT 型の FREObject 変数です。このデフォルトの戻り値の結果、ActionScript 側では call() が null を返します。

ctx パラメーターは次の場合に使用します。

- 拡張に関連付けるデータを取得または設定する場合。このデータは、ネイティブコンテキストデータと ActionScript コンテキストデータのどちらの可能性もあります。18 ページの「コンテキスト固有データ」を参照してください。
- ActionScript 側の ExtensionContext インスタンスに非同期イベントを送出する場合。94 ページの「FREDispatchStatusEventAsync()」を参照してください。

FREObject パラメーターおよび戻り値を操作する場合は、90 ページの「ユーザー使用関数」の関数を使用します。

関連項目

86 ページの「FREContextInitializer()」

21 ページの「FREObject 型」

FREInitializer()

AIR 3.0 以降

シグネチャ

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

パラメーター

extDataToSet ネイティブ拡張の拡張データに対するポインターへのポインター。拡張固有のデータを保持するには、データ構造体を作成します。例えば、ヒープからデータを割り当てる場合や、グローバルデータを提供する場合があります。extDataToSet を、割り当てられたデータへのポインターに設定します。

ctxInitializerToSet FREContextInitializer() 関数に対するポインターへのポインター。ctxInitializerToSet を、定義した FREContextInitializer() 関数に設定します。

ctxFinalizerToSet FREContextFinalizer() 関数に対するポインターへのポインター。ctxFinalizerToSet を、定義した FREContextFinalizer() 関数に設定します。このポインターを NULL に設定することもできます。

戻り値

なし。

説明

ランタイムは、拡張をロードするときに一度だけこのメソッドを呼び出します。拡張が必要とする初期化処理を実行するには、この関数を実装します。次に出力パラメーターを設定します。

関連項目

86 ページの「FREContextInitializer()」

86 ページの「FREContextFinalizer()」

ユーザー使用関数

ネイティブ拡張 C API では、ActionScript オブジェクトおよびプリミティブデータにアクセスし、これらを操作するための関数が提供されます。

FREAcquireBitmapData()

AIR 3.0 以降

使用方法

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

パラメーター

object FREObject。この FREObject パラメーターは、ActionScript BitmapData クラスオブジェクトを表します。

descriptorToSet FREBitmapData 型の変数へのポインター。ネイティブ C 実装でこのメソッドを呼び出すと、ランタイムがこの構造体のフィールドを設定します。81 ページの「FREBitmapData」を参照してください。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数は成功しました。FREBitmapData パラメーターが設定されています。ActionScript BitmapData オブジェクトを使用して操作できます。

FRE_ILLEGAL_STATE 拡張コンテキストは既に ActionScript BitmapData または ByteArray オブジェクトを取得しています。この BitmapData または ByteArray オブジェクトを解放するまでは、コンテキストはこのメソッドを呼び出すことはできません。

FRE_INVALID_ARGUMENT descriptorToSet パラメーターは NULL です。

FRE_INVALID_OBJECT FREObject object パラメーターは無効です。

FRE_TYPE_MISMATCH FREObject object パラメーターは ActionScript BitmapData クラスオブジェクトを表していません。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明

ActionScript BitmapData クラスオブジェクトのビットマップを取得するには、この関数を呼び出します。この関数の呼び出しに成功すると、FREReleaseBitmapData() を呼び出すまでは、他の C API 関数を正常に呼び出すことはできません。これが禁止されるのは、他の呼び出しの副作用として、ビットマップコンテンツへのポインターを無効にするコードが実行されるからです。

この関数を呼び出した後は、BitmapData オブジェクトのビットマップを操作できます。ビットマップは、ビットマップに関する他の情報と同様に、descriptorToSet パラメーターで使用できます。ビットマップまたは長方形領域が変更されたことをランタイムに通知するには、FREInvalidateBitmapDataRect() を呼び出します。ビットマップの操作を終了したら、FREReleaseBitmapData() を呼び出します。

関連項目

108 ページの「FREReleaseBitmapData()」

103 ページの「FREInvalidateBitmapDataRect()」

FREAcquireBitmapData2()

AIR 3.1 以降

使用方法

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

パラメーター

object FREObject。この FREObject パラメーターは、ActionScript BitmapData クラスオブジェクトを表します。

descriptorToSet FREBitmapData2 型の変数へのポインター。ネイティブ C 実装でこのメソッドを呼び出すと、ランタイムがこの構造体のフィールドを設定します。82 ページの「FREBitmapData2」を参照してください。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数は成功しました。FREBitmapData2 パラメーターが設定されています。ActionScript BitmapData オブジェクトを使用して操作できます。

FRE_ILLEGAL_STATE 拡張コンテキストは既に ActionScript BitmapData または ByteArray オブジェクトを取得しています。この BitmapData または ByteArray オブジェクトを解放するまでは、コンテキストはこのメソッドを呼び出すことはできません。

FRE_INVALID_ARGUMENT descriptorToSet パラメーターは NULL です。

FRE_INVALID_OBJECT FREObject object パラメーターは無効です。

FRE_TYPE_MISMATCH FREObject object パラメーターは ActionScript BitmapData クラスオブジェクトを表していません。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明

ActionScript BitmapData クラスオブジェクトのビットマップを取得するには、この関数を呼び出します。この関数の呼び出しに成功すると、FREReleaseBitmapData() を呼び出すまでは、他の C API 関数を正常に呼び出すことはできません。これが禁止されるのは、他の呼び出しの副作用として、ビットマップコンテンツへのポインターを無効にするコードが実行されるからです。

この関数を呼び出した後は、BitmapData オブジェクトのビットマップを操作できます。ビットマップは、ビットマップに関する他の情報と同様に、descriptorToSet パラメーターで使用できます。ビットマップまたは長方形領域が変更されたことをランタイムに通知するには、FREInvalidateBitmapDataRect() を呼び出します。ビットマップの操作を終了したら、FREReleaseBitmapData() を呼び出します。

関連項目

108 ページの「FREReleaseBitmapData()」

103 ページの「FREInvalidateBitmapDataRect()」

FREAcquireByteArray()

AIR 3.0 以降

使用方法

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

パラメーター

object FREObject。この FREObject パラメーターは、ActionScript ByteArray クラスオブジェクトを表します。

byteArrayToSet FREByteArray 型の変数へのポインター。ネイティブ C 実装でこのメソッドを呼び出すと、ランタイムがこの構造体のフィールドを設定します。83 ページの「FREByteArray」を参照してください。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数は成功しました。FREByteArray パラメーターが設定されています。ActionScript ByteArray オブジェクトを使用して操作できます。

FRE_ILLEGAL_STATE 拡張コンテキストは既に ActionScript BitmapData または ByteArray オブジェクトを取得しています。この BitmapData または ByteArray オブジェクトを解放するまでは、コンテキストはこのメソッドを呼び出すことはできません。

FRE_INVALID_ARGUMENT byteArrayToSet パラメーターは NULL です。

FRE_INVALID_OBJECT FREObject object パラメーターは無効です。

FRE_TYPE_MISMATCH FREObject object パラメーターは ActionScript ByteArray クラスオブジェクトを表していません。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明

ActionScript ByteArray クラスオブジェクトのバイトを取得するには、この関数を呼び出します。この関数の呼び出しに成功すると、FREReleaseByteArray() を呼び出すまでは、他の C API 関数を正常に呼び出すことはできません。これが禁止されるのは、他の呼び出しの副作用として、バイト配列コンテンツへのポインターを無効にするコードが実行されるからです。

この関数を呼び出した後は、ByteArray オブジェクトのバイトを操作できます。このバイトは、バイト数と同様に byteArrayToSet パラメーターで利用できます。バイトの操作を終了したら、FREReleaseByteArray() を呼び出します。

関連項目

109 ページの「FREReleaseByteArray()」

FRECallObjectMethod()

AIR 3.0 以降

使用方法

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

パラメーター

object メソッドを呼び出す ActionScript クラスオブジェクトを表す FREObject。

methodName uint8_t 配列。この配列は、呼び出すメソッドの名前を表す文字列です。この文字列には UTF-8 エンコーディングを使用し、null 文字で終了します。

argc uint32_t。この値は、メソッドに渡されたパラメーターの数です。このパラメーターは、argv 配列パラメーターの長さを表します。呼び出すメソッドにパラメーターがない場合、この値は 0 となります。

argv[] FREObject 配列。各 FREObject エレメントは、呼び出すメソッドにパラメーターとして渡される ActionScript クラスまたはプリミティブ型に対応します。呼び出すメソッドにパラメーターがない場合、この値は NULL となります。

result FREObject へのポインター。この FREObject 変数は、呼び出すメソッドの戻り値を取得するためのものです。FREObject 変数は、呼び出すメソッドが返す ActionScript クラスまたはプリミティブ型を表します。

thrownException FREObject へのポインター。このメソッドを呼び出すことによりランタイムが ActionScript 例外をスローした場合、この FREObject 変数は、ActionScript Error オブジェクトまたは Error サブクラスのオブジェクトを表します。エラーが発生しない場合、ランタイムはこの FREObject 変数を無効に設定します。つまり、thrownException FREObject 変数に FREGetObjectType() を実行すると、FRE_INVALID_OBJECT が返されます。例外情報の処理を行わない場合は、このポインターを NULL に設定できます。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数は成功しました。ActionScript メソッドは、例外をスローせずに処理を完了しました。

FRE_ACTIONSCRIPT_ERROR ActionScript エラーが発生しました。ランタイムは thrownException パラメーターを、ActionScript Error クラスオブジェクトまたは Error サブクラスオブジェクトを表すように設定します。

FRE_ILLEGAL_STATE 拡張コンテキストは既に ActionScript BitmapData または ByteArray オブジェクトを取得しています。この BitmapData または ByteArray オブジェクトを解放するまでは、コンテキストはこのメソッドを呼び出すことはできません。

FRE_INVALID_ARGUMENT method または result パラメーターが NULL であるか、argc が 1 以上かつ argv が NULL です。

FRE_INVALID_OBJECT FREObject パラメーターまたは argv の FREObject エレメントは無効です。

FRE_NO_SUCH_NAME methodName パラメーターは、object パラメーターが表す ActionScript クラスオブジェクトのメソッドと一致しません。可能性は低いですが、この戻り値の理由は他にもあります。具体的には、ActionScript クラスに 2 つの同名のメソッドがあるものの、これらが異なる ActionScript 名前空間のものであるという、稀なケースが考えられます。

FRE_TYPE_MISMATCH FREObject パラメーターは ActionScript クラスオブジェクトを表していません。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明

ActionScript クラスオブジェクトのメソッドを呼び出すには、この関数を呼び出します。

関連項目

21 ページの「FREObject 型」

FREDispatchStatusEventAsync()

AIR 3.0 以降

使用方法

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

パラメーター

ctx FREContext。この値は、拡張コンテキストがコンテキスト初期化関数内で取得した FREContext 変数です。

code uint8_t へのポインター。ランタイムは、StatusEvent オブジェクトの code プロパティをこの値に設定します。この文字列には UTF-8 エンコーディングを使用し、null 文字で終了します。

level uint8_t へのポインター。このパラメーターは、null 終端文字を含む UTF8 エンコーディング文字列です。ランタイムは、StatusEvent オブジェクトの level プロパティをこの値に設定します。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数は成功しました。

FRE_INVALID_ARGUMENT ctx、code または level パラメーターは NULL です。ctx が有効でない場合も、ランタイムはこの値を返します。

説明

ActionScript StatusEvent イベントを送出するには、この関数を呼び出します。イベントターゲットは、ランタイムが ctx パラメーターによって指定されたコンテキストと関連付けた ActionScript ExtensionContext インスタンスです。

通常、この関数が送出するイベントは非同期です。例えば、拡張メソッドから、タスクを実行するための別のスレッドを開始させることができます。他のスレッドのタスクが完了すると、そのスレッドは FREDispatchStatusEventAsync() を呼び出して、ActionScript ExtensionContext インスタンスに通知します。

注意：FREDispatchStatusEventAsync() 関数は、ネイティブ実装のスレッドから呼び出すことのできる唯一の C API です。

無効な引数が 1 つもない限り、FREDispatchStatusEventAsync() は FRE_OK を返します。ただし、FRE_OK が返されたことが、イベントが送出されたことを示すわけではありません。次の場合には、ランタイムはイベントを送出しません。

- ランタイムがすでに ExtensionContext インスタンスを破棄している場合。
- ランタイムが ExtensionContext インスタンスの破棄を処理中である場合。
- ExtensionContext インスタンスが参照を持っていない場合。ランタイムのガベージコレクターがインスタンスを破棄できる状態にあります。

code および level パラメーターを、null 終端文字を含む UTF8 エンコーディング文字列値に設定します。これらの値は任意に設定できますが、ActionScript 側の拡張に合わせて調整してください。

例

関連項目

80 ページの「FREContext」

FREGetArrayElementAt()

AIR 3.0 以降

使用方法

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

パラメーター

arrayOrVector ActionScript Array または Vector クラスオブジェクトを表す FREObject。

index 取得するための Array または Vector エレメントのインデックスを含む uint32_t。Array または Vector オブジェクトの最初のエレメントでは、index は 0 です。

value FREObject へのポインター。このメソッドは、このパラメーターがポイントする FREObject 変数を設定します。メソッドは FREObject 変数を、要求されたインデックス位置にある Array または Vector エレメントに対応するように設定します。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数は成功しました。value パラメーターは、要求された Array または Vector エレメントに設定されます。

FRE_ILLEGAL_STATE 拡張コンテキストは既に ActionScript BitmapData または ByteArray オブジェクトを取得しています。この BitmapData または ByteArray オブジェクトを解放するまでは、コンテキストはこのメソッドを呼び出すことはできません。

FRE_INVALID_ARGUMENT arrayOrVector パラメーターは ActionScript Vector オブジェクトに対応していますが、index が、最後のエレメントのインデックスよりも大きくなっています。この戻り値となる他の理由は、value パラメーターが NULL の場合です。

FRE_INVALID_OBJECT FREObject arrayOrVector パラメーターが無効です。

FRE_TYPE_MISMATCH FREObject arrayOrVector パラメーターは、ActionScript Array または Vector クラスオブジェクトを表していません。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明

ActionScript Array または Vector クラスオブジェクトの指定したインデックス位置にある、ActionScript クラスオブジェクトまたはプリミティブ値を取得するには、この関数を呼び出します。FREObject arrayOrVector パラメーターは、Array または Vector オブジェクトを表します。ランタイムは、value パラメーターがポイントする FREObject 変数を設定します。FREObject 変数を、適切な Array または Vector エレメントに対応するように設定します。

ActionScript Array オブジェクトの要求されたインデックス位置に値がない場合、ランタイムが FREObject value パラメーターを無効に設定しますが、FRE_OK を返します。

関連項目

110 ページの「FRESetArrayElementAt()」

111 ページの「FRESetArrayLength()」

FREGetArrayLength()

AIR 3.0 以降

使用方法

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

パラメーター

arrayOrVector ActionScript Array または Vector クラスオブジェクトを表す FREObject。

length uint32_t へのポインター。このメソッドは、Array または Vector クラスオブジェクトの長さを使用して、このパラメーターがポイントする uint32_t 変数を設定します。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数は成功しました。メソッドは、length パラメーターがポイントする uint32_t 変数を設定しました。このメソッドは、変数を Array または Vector オブジェクトの長さに設定します。

FRE_ILLEGAL_STATE 拡張コンテキストは既に ActionScript BitmapData または ByteArray オブジェクトを取得しています。この BitmapData または ByteArray オブジェクトを解放するまでは、コンテキストはこのメソッドを呼び出すことはできません。

FRE_INVALID_ARGUMENT length パラメーターは NULL です。

FRE_INVALID_OBJECT FREObject arrayOrVector パラメーターが無効です。

FRE_TYPE_MISMATCH FREObject arrayOrVector パラメーターは、ActionScript Array または Vector クラスオブジェクトを表していません。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明

Array または Vector クラスオブジェクトの長さを取得するには、この関数を呼び出します。FREObject arrayOrVector パラメーターは、Array または Vector オブジェクトを表します。ランタイムは、length パラメーターがポイントする uint32_t 変数内に、その長さを返します。

関連項目

111 ページの「FRESetArrayLength()」

110 ページの「FRESetArrayElementAt()」

FREGetContextActionScriptData()

AIR 3.0 以降

使用方法

```
FREResult FREGetContextActionScriptData( FREContext ctx, FREObject *actionScriptData);
```

パラメーター

ctx FREContext 変数。ランタイムはこの値を FREContextInitializer() に渡しました。86 ページの「FREContextInitializer()」を参照してください。

actionScriptData FREObject 変数へのポインター。FREGetContextActionScriptData() が成功すると、ランタイムはこのパラメーターを設定します。この値は、以前に FRESetContextActionScriptData() を使用して保存された ActionScript オブジェクトに対応します。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数は成功しました。actionScriptData パラメーターは、以前に FRESetContextActionScriptData() を使用して保存された ActionScript オブジェクトに対応します。

FRE_INVALID_ARGUMENT actionScriptData パラメーターは NULL です。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明

拡張コンテキストの ActionScript データを取得するには、この関数を呼び出します。

関連項目

112 ページの「FRESetContextActionScriptData()」

18 ページの「コンテキスト固有データ」

FREGetContextNativeData()

AIR 3.0 以降

使用方法

```
FREResult FREGetContextNativeData( FREContext ctx, void** nativeData );
```

パラメーター

ctx FREContext 変数。ランタイムはこの値を FREContextInitializer() に渡しました。86 ページの「FREContextInitializer()」を参照してください。

nativeData ネイティブデータに対するポインターへのポインター。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数は成功しました。nativeData パラメーターがコンテキストのネイティブデータをポイントするように設定されています。

FRE_INVALID_ARGUMENT nativeData パラメーターは NULL です。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明

拡張コンテキストのネイティブデータを取得するには、この関数を呼び出します。

関連項目

112 ページの「FRESetContextNativeData()」

18 ページの「コンテキスト固有データ」

FREGetObjectAsBool()

AIR 3.0 以降

使用方法

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

パラメーター

object FREObject。

value uint32_t へのポインター。関数はこの値を、FREObject 変数が表す ActionScript Boolean 変数の値に対応するように設定します。ゼロ以外の値は true に対応します。ゼロの値は false に対応します。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数が成功し、value パラメーターが正しく設定されています。

FRE_TYPE_MISMATCH FREObject パラメーターに、ActionScript Boolean 値が含まれていません。

FRE_INVALID_OBJECT FREObject パラメーターが無効です。

FRE_INVALID_ARGUMENT value パラメーターは NULL です。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明

C の uint32_t 変数の値を、ActionScript Boolean 変数の値に設定するには、この関数を呼び出します。

関連項目

105 ページの「FRENewObjectFromBool()」

21 ページの「FREObject 型」

FREGetObjectAsDouble()

AIR 3.0 以降

使用方法

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

パラメーター

object FREObject。

value double へのポインター。関数はこの値を、FREObject 変数が表す ActionScript Boolean、int または Number 変数に対応するように設定します。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数が成功し、value パラメーターが正しく設定されています。

FRE_TYPE_MISMATCH FREObject パラメーターに、ActionScript Boolean、int または Number 値が含まれていません。

FRE_INVALID_OBJECT FREObject パラメーターが無効です。

FRE_INVALID_ARGUMENT value パラメーターは NULL です。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明

C の double 変数の値を、ActionScript Boolean、int または Number 変数の値に設定するには、この関数を呼び出します。

関連項目

106 ページの「FRENewObjectFromDouble()」

21 ページの「FREObject 型」

FREGetObjectAsInt32()

AIR 3.0 以降

使用方法

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

パラメーター

object FREObject。

value int32_t へのポインター。関数はこの値を、FREObject 変数が表す ActionScript Boolean または int 変数の値に対応するように設定します。

戻り値
FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数が成功し、value パラメーターが正しく設定されています。

FRE_TYPE_MISMATCH FREObject パラメーターには、ActionScript Boolean または int 値が含まれていません。

FRE_INVALID_OBJECT FREObject パラメーターが無効です。

FRE_INVALID_ARGUMENT value パラメーターは NULL です。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明
C の int32_t 変数を、ActionScript int または Boolean 変数の値に設定するには、この関数を呼び出します。

関連項目
107 ページの「FRENewObjectFromInt32()」

21 ページの「FREObject 型」

FREGetObjectAsUint32()

AIR 3.0 以降

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

パラメーター
object FREObject。

value uint32_t へのポインター。関数はこの値を、FREObject 変数が表す ActionScript Boolean または int 変数の値に対応するように設定します。

戻り値
FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数が成功し、value パラメーターが正しく設定されています。

FRE_TYPE_MISMATCH FREObject パラメーターには、ActionScript Boolean または int 値が含まれていません。ActionScript int 値が負の場合も、この戻り値となります。

FRE_INVALID_OBJECT FREObject パラメーターが無効です。

FRE_INVALID_ARGUMENT value パラメーターは NULL です。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明
C の uint32_t 変数の値を、ActionScript Boolean または int 変数の値に設定するには、この関数を呼び出します。

関連項目

107 ページの「FRENewObjectFromUint32()」

21 ページの「FREObject 型」

FREGetObjectAsUTF8()

AIR 3.0 以降

使用方法

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

パラメーター

object FREObject。

length uint32_t へのポインター。length の値は、value 配列のバイト数です。length には null 終端文字が含まれます。length は、FREObject 変数が表す ActionScript String 変数の長さに対応します。

value uint8_t 配列へのポインター。関数は、FREObject 変数が表す ActionScript String 変数の文字を使用して、この配列に値を入力します。この文字列では UTF-8 エンコーディングを使用し、null 終端文字を含めます。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数が成功し、value および length パラメーターが正しく設定されています。

FRE_TYPE_MISMATCH FREObject パラメーターには、ActionScript String 値が含まれていません。

FRE_INVALID_OBJECT FREObject パラメーターが無効です。

FRE_INVALID_ARGUMENT value または length パラメーターは NULL です。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明

uint8_t 配列の値を ActionScript String オブジェクトの文字列値に設定するには、この関数を呼び出します。

value パラメーターに返される文字列について、次のことを考慮してください。

- 文字列は変更できません。
- 文字列が有効なのは、ランタイムが呼び出したネイティブ拡張関数が完了するまでの間だけです。
- 他の C API 関数を呼び出すと、文字列は無効になります。

したがって、後で文字列を操作する場合または文字列にアクセスする場合は、すぐに独自に作成した配列に文字列をコピーしてください。

関連項目

108 ページの「FRENewObjectFromUTF8()」

21 ページの「FREObject 型」

FREGetObjectProperty()

AIR 3.0 以降

使用方法

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

パラメーター

object プロパティの値を取得する対象となる ActionScript クラスオブジェクトを表す FREObject。

propertyName uint8_t 配列。この配列にはプロパティの名前を表す文字列が含まれます。この文字列には UTF-8 エンコーディングを使用し、null 文字で終了します。

propertyValue FREObject へのポインター。このメソッドは、この FREObject パラメーターを、要求されたプロパティである ActionScript オブジェクトを表すように設定します。

thrownException FREObject へのポインター。このメソッドを呼び出すことによりランタイムが ActionScript 例外をスローした場合、この FREObject 変数は、ActionScript Error オブジェクトまたは Error サブクラスのオブジェクトを表します。エラーが発生しない場合、ランタイムはこの FREObject 変数を無効に設定します。つまり、thrownException FREObject 変数に FREGetObjectType() を実行すると、FRE_INVALID_OBJECT が返されます。例外情報の処理を行わない場合は、このポインターを NULL に設定できます。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数が成功し、propertyValue パラメーターが正しく設定されています。

FRE_ACTIONSCRIPT_ERROR ActionScript エラーが発生しました。ランタイムは thrownException パラメーターを、ActionScript Error クラスオブジェクトまたは Error サブクラスオブジェクトを表すように設定します。

FRE_ILLEGAL_STATE 拡張コンテキストは既に ActionScript BitmapData または ByteArray オブジェクトを取得しています。この BitmapData または ByteArray オブジェクトを解放するまでは、コンテキストはこのメソッドを呼び出すことはできません。

FRE_INVALID_ARGUMENT propertyName または propertyValue パラメーターは NULL です。

FRE_INVALID_OBJECT FREObject パラメーターが無効です。

FRE_NO_SUCH_NAME propertyName パラメーターは object パラメーターが表す ActionScript クラスオブジェクトのプロパティと一致しません。可能性は低いですが、この戻り値の理由は他にもあります。具体的には、ActionScript クラスに 2 つの同名のプロパティがあるものの、これらが異なる ActionScript 名前空間のものであるという、稀なケースが考えられます。

FRE_TYPE_MISMATCH FREObject パラメーターは ActionScript クラスオブジェクトを表していません。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明

object パラメーターで指定した ActionScript クラスオブジェクトの public プロパティに対応するデータを示す FREObject 変数を取得するには、この関数を呼び出します。

関連項目

113 ページの「FRESetObjectProperty()」

21 ページの「FREObject 型」

FREGetObjectType()

AIR 3.0 以降

使用方法

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

パラメーター

object FREObject 変数。

objectType FREObjectType 変数へのポインター。FREGetObjectType() はこの変数を、FREObjectType 列挙値の 1 つに設定します。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数が成功し、objectType パラメーターが正しく設定されています。

FRE_INVALID_ARGUMENT objectType パラメーターは NULL です。

FRE_INVALID_OBJECT FREObject パラメーターが無効です。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明

FREObject 変数に対応する ActionScript クラスオブジェクトまたはプリミティブ型を最も適切に表現する FREObjectType 列挙値を取得するには、この関数を呼び出します。

関連項目

84 ページの「FREObjectType」

21 ページの「FREObject 型」

FREInvalidateBitmapDataRect()

AIR 3.0 以降

使用方法

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

パラメーター

object 以前取得した ActionScript BitmapData クラスオブジェクトを表す FREObject。

x uint32_t。この値は、x 座標をピクセル単位で表します。この値はビットマップの左上隅からの相対位置です。この値は、y パラメーターと一緒に使用して、無効にする長方形領域の左上隅を示します。

y uint32_t。この値は、y 座標をピクセル単位で表します。この値はビットマップの左上隅からの相対位置です。この値は、x パラメーターと一緒に使用して、無効にする長方形領域の左上隅を示します。

width uint32_t。この値は、無効にする長方形領域のピクセル単位の幅です。

height uint32_t。この値は、無効にする長方形領域のピクセル単位の高さです。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数は成功しました。指定した長方形領域が無効になりました。

FRE_ILLEGAL_STATE 拡張コンテキストは、ActionScript BitmapData オブジェクトを取得していません。

FRE_INVALID_OBJECT FREObject object パラメーターは無効です。

FRE_TYPE_MISMATCH FREObject object パラメーターは ActionScript BitmapData クラスオブジェクトを表していません。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明

ActionScript BitmapData クラスオブジェクトの長方形領域を無効にするには、この関数を呼び出します。この関数を呼び出す前に、FREAcquireBitmapData() または FREAcquireBitmapData2() を呼び出します。ビットマップの操作と無効化が完了した後に、FREReleaseBitmapData() を呼び出します。

BitmapData オブジェクトの長方形領域を無効にすると、ランタイムはその長方形領域を再描画することが必要になります。

関連項目

90 ページの「FREAcquireBitmapData()」

91 ページの「FREAcquireBitmapData2()」

108 ページの「FREReleaseBitmapData()」

FRENewObject()

AIR 3.0 以降

使用方法

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

パラメーター

className uint8_t 配列。オブジェクトを作成する ActionScript クラスの名前を示す文字列です。この文字列には UTF-8 エンコーディングを使用し、null 文字で終了します。

argc uint32_t。ActionScript クラスのコンストラクターに渡すパラメーター数。このパラメーターは、argv 配列パラメーターの長さを表します。コンストラクターにパラメーターがない場合、この値は 0 となります。

argv[] FREObject 配列。各 FREObject エレメントは、コンストラクターにパラメーターとして渡された ActionScript クラスまたはプリミティブ型に対応します。コンストラクターにパラメーターがない場合は、この値は NULL となります。

object FREObject へのポインター。このメソッドが成功した場合、この FREObject 変数は新しい ActionScript クラスオブジェクトを表します。

thrownException FREObject へのポインター。このメソッドを呼び出すことによりランタイムが ActionScript 例外をスローした場合、この FREObject 変数は、ActionScript Error オブジェクトまたは Error サブクラスのオブジェクトを表します。エラーが発生しない場合、ランタイムはこの FREObject 変数を無効に設定します。つまり、thrownException FREObject 変数に FREGetObjectType() を実行すると、FRE_INVALID_OBJECT が返されます。例外情報の処理を行わない場合は、このポインターを NULL に設定できます。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数は成功しました。object パラメーターは、新しい ActionScript クラスオブジェクトを表します。

FRE_ACTIONSCRIPT_ERROR ActionScript エラーが発生しました。ランタイムは thrownException パラメーターを、ActionScript Error クラスオブジェクトまたは Error サブクラスオブジェクトを表すように設定します。

FRE_ILLEGAL_STATE 拡張コンテキストは既に ActionScript BitmapData または ByteArray オブジェクトを取得しています。この BitmapData または ByteArray オブジェクトを解放するまでは、コンテキストはこのメソッドを呼び出すことはできません。

FRE_INVALID_ARGUMENT className または object パラメーターが NULL であるか、argc が 1 以上かつ argv が NULL または空です。

FRE_NO_SUCH_NAME className パラメーターは、ActionScript クラス名と一致しません。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明

ActionScript クラスのオブジェクトを作成するには、この関数を呼び出します。ランタイムが呼び出すコンストラクターは、argv 配列に渡すパラメーターによって異なります。

関連項目

21 ページの「FREObject 型」

FRENewObjectFromBool()

AIR 3.0 以降

使用方法

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

パラメーター

value uint32_t。新しい ActionScript Boolean インスタンスの値となります。

object FREObject へのポインター。ActionScript Boolean 変数を表すデータをポイントします。ゼロ以外の値は true に対応します。ゼロの値は false に対応します。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数が成功し、object パラメーターが正しく設定されています。

FRE_INVALID_ARGUMENT FREObject パラメーターは NULL です。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明

value パラメーターを使用して ActionScript Boolean インスタンスを作成するには、この関数を呼び出します。ランタイムは FREObject 変数を、新しい ActionScript インスタンスに対応するデータに設定します。

関連項目

98 ページの「FREGetObjectAsBool()」

21 ページの「FREObject 型」

FRENewObjectFromDouble()

AIR 3.0 以降

使用方法

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

パラメーター

value double。新しい ActionScript Number インスタンスの値となります。

object FREObject へのポインター。Number ActionScript 変数を表すデータをポイントします。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数が成功し、object パラメーターが正しく設定されています。

FRE_INVALID_ARGUMENT FREObject パラメーターは NULL です。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明

value パラメーターを使用して ActionScript Number インスタンスを作成するには、この関数を呼び出します。ランタイムは FREObject 変数を、新しい ActionScript インスタンスに対応するデータに設定します。

関連項目

99 ページの「FREGetObjectAsDouble()」

21 ページの「FREObject 型」

FRENewObjectFromInt32()

AIR 3.0 以降

使用方法

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

パラメーター

value int32_t。新しい ActionScript int インスタンスの値となります。

object FREObject へのポインター。ActionScript int インスタンスを表します。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数が成功し、object パラメーターが正しく設定されています。

FRE_INVALID_ARGUMENT FREObject パラメーターは NULL です。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明

value パラメーターを使用して ActionScript int インスタンスを作成するには、この関数を呼び出します。ランタイムは FREObject 変数を、新しい ActionScript インスタンスに対応するように設定します。

関連項目

99 ページの「FREGetObjectAsInt32()」

21 ページの「FREObject 型」

FRENewObjectFromUint32()

AIR 3.0 以降

使用方法

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

パラメーター

value uint32_t。新しい ActionScript int インスタンスの値（0 以上）となります。

object FREObject へのポインター。ActionScript int インスタンスを表します。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数が成功し、object パラメーターが正しく設定されています。

FRE_INVALID_ARGUMENT FREObject パラメーターは NULL です。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明
value パラメーターを使用して ActionScript int インスタンスを作成するには、この関数を呼び出します。ランタイムは FREObject 変数を、新しい ActionScript int インスタンスに対応するように設定します。

関連項目
100 ページの「FREGetObjectAsUint32()」
21 ページの「FREObject 型」

FRENewObjectFromUTF8()

AIR 3.0 以降

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

パラメーター
length uint32_t。value パラメーターの、null 終端文字を含む文字列の長さを示します。

value uint8_t エレメントの配列。配列は、新しい ActionScript String オブジェクトの値です。FREObject 変数は ActionScript String 変数を表します。この文字列には UTF-8 エンコーディングを使用し、null 文字で終了します。

object FREObject へのポインター。ActionScript String オブジェクトを表すデータをポイントします。

戻り値
FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数が成功し、object パラメーターが正しく設定されています。

FRE_INVALID_ARGUMENT object または value パラメーターは NULL です。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明
value に指定した文字列値を使用して ActionScript String オブジェクトを作成するには、この関数を呼び出します。このメソッドは、新しい ActionScript String インスタンスに対応するように、FREObject object 出力パラメーターを設定します。

関連項目
101 ページの「FREGetObjectAsUTF8()」
21 ページの「FREObject 型」

FREReleaseBitmapData()

AIR 3.0 以降

使用方法
```
FREResult FREReleaseBitmapData (FREObject object);
```

パラメーター
object ActionScript BitmapData クラスオブジェクトに対応する FREObject。

戻り値
FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数は成功しました。ActionScript BitmapData オブジェクトは、これ以降は操作できません。

FRE_ILLEGAL_STATE 拡張コンテキストは、ActionScript BitmapData オブジェクトを取得していません。

FRE_INVALID_OBJECT FREObject object パラメーターは無効です。

FRE_TYPE_MISMATCH FREObject object パラメーターは ActionScript BitmapData クラスオブジェクトを表していません。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明
ActionScript BitmapData クラスオブジェクトを解放するには、この関数を呼び出します。この関数を呼び出す前に、FREAcquireBitmapData() または FREAcquireBitmapData2() および FREInvalidateBitmapDataRect() を呼び出します。この関数を呼び出すと、以降はビットマップを操作できなくなりますが、他のネイティブ拡張 C API 関数を再度呼び出すことは可能です。

関連項目
90 ページの「FREAcquireBitmapData()」

91 ページの「FREAcquireBitmapData2()」

103 ページの「FREInvalidateBitmapDataRect()」

FREReleaseByteArray()

AIR 3.0 以降

使用方法
```
FREResult FREReleaseByteArray (FREObject object);
```

パラメーター
object FREObject。ActionScript ByteArray クラスオブジェクトに対応します。

戻り値
FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数は成功しました。ActionScript ByteArray オブジェクトは、これ以降は操作できません。

FRE_ILLEGAL_STATE 拡張コンテキストは、ActionScript ByteArray オブジェクトを取得していません。

FRE_INVALID_OBJECT FREObject object パラメーターは無効です。

FRE_TYPE_MISMATCH FREObject object パラメーターは ActionScript ByteArray オブジェクトに対応しません。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明

ActionScript ByteArray クラスオブジェクトを解放するには、この関数を呼び出します。この関数を呼び出す前に、FREAcquireByteArray() を呼び出します。この関数を呼び出すと、以降は ByteArray のバイトを操作できなくなりますが、他のネイティブ拡張 C API 関数を再度呼び出すことは可能です。

関連項目

92 ページの「FREAcquireByteArray()」

FRESetArrayElementAt()

AIR 3.0 以降

使用方法

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

パラメーター

arrayOrVector FREObject。ActionScript Array または Vector クラスオブジェクトを表すデータをポイントします。

index uint32_t。設定する Array または Vector エレメントのインデックスが含まれます。Array または Vector オブジェクトの最初のエレメントでは、index は 0 です。

value FREObject。このメソッドは、index が指定する Array または Vector エレメントを、FREObject value パラメーターが表す ActionScript オブジェクトに設定します。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数は成功しました。Array または Vector エレメントは、FREObject value パラメーターに設定されます。

FRE_ILLEGAL_STATE 拡張コンテキストは既に ActionScript BitmapData または ByteArray オブジェクトを取得しています。この BitmapData または ByteArray オブジェクトを解放するまでは、コンテキストはこのメソッドを呼び出すことはできません。

FRE_INVALID_OBJECT FREObject arrayOrVector または value パラメーターは無効です。

FRE_TYPE_MISMATCH FREObject arrayOrVector パラメーターは、ActionScript Array または Vector クラスオブジェクトを表すデータをポイントしていません。この戻り値は、arrayOrVector パラメーターが Vector オブジェクトを表し、value パラメーターは Vector オブジェクトの正しい型ではないことを示すこともあります。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明

ActionScript Array または Vector クラスオブジェクトの指定したインデックス位置に、ActionScript クラスオブジェクトまたはプリミティブ値を設定するには、この関数を呼び出します。FREObject arrayOrVector パラメーターは、Array または Vector オブジェクトに対応します。FREObject value パラメーターは、配列エレメントの値に対応します。

関連項目

95 ページの「FREGetArrayElementAt()」

96 ページの「FREGetArrayLength()」

FRESetArrayLength()

AIR 3.0 以降

使用方法

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

パラメーター

arrayOrVector ActionScript Array または Vector クラスオブジェクトを表す FREObject。

length uint32_t。このメソッドは Array または Vector クラスオブジェクトの長さを、このパラメーターの値に設定します。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数は成功しました。ランタイムは Array または Vector オブジェクトのサイズを変更しました。

FRE_ILLEGAL_STATE 拡張コンテキストは既に ActionScript BitmapData または ByteArray オブジェクトを取得しています。この BitmapData または ByteArray オブジェクトを解放するまでは、コンテキストはこのメソッドを呼び出すことはできません。

FRE_INSUFFICIENT_MEMORY ランタイムは Array または Vector オブジェクトのサイズを変更するのに十分なメモリを割り当てることができませんでした。

FRE_INVALID_ARGUMENT length パラメーターは 2^{32} よりも大きな値です。

FRE_INVALID_OBJECT FREObject arrayOrVector パラメーターが無効です。

FRE_READ_ONLY FREObject arrayOrVector パラメーターは、固定サイズの ActionScript Vector オブジェクトを表しています（fixed プロパティが true です）。

FRE_TYPE_MISMATCH FREObject arrayOrVector パラメーターは、ActionScript Array または Vector クラスオブジェクトを表していません。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明

ActionScript Array または Vector クラスオブジェクトの長さを設定するには、この関数を呼び出します。FREObject arrayOrVector パラメーターは、Array または Vector オブジェクトに対応します。ランタイムは、length パラメーターが指定した値のとおりに、Array または Vector オブジェクトのサイズを変更します。

関連項目

95 ページの「FREGetArrayElementAt()」

96 ページの「FREGetArrayLength()」

FRESetContextActionScriptData()

AIR 3.0 以降

使用方法

```
FREResult FRESetContextActionScriptData( FREContext ctx, FREObject actionScriptData);
```

パラメーター

ctx FREContext 変数。ランタイムはこの値を FREContextInitializer() に渡しました。86 ページの「FREContextInitializer()」を参照してください。

actionScriptData FREObject 変数。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数は成功しました。

FRE_INVALID_OBJECT actionScriptData パラメーターは無効な FREObject 変数です。

FRE_INVALID_ARGUMENT 引数が無効です。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明

拡張コンテキストの ActionScript データを設定するには、この関数を呼び出します。

関連項目

97 ページの「FREGetContextActionScriptData()」

18 ページの「コンテキスト固有データ」

FRESetContextNativeData()

AIR 3.0 以降

使用方法

```
FREResult FRESetContextNativeData( FREContext ctx, void* nativeData );
```

パラメーター

ctx FREContext 変数。ランタイムはこの値を FREContextInitializer() に渡しました。86 ページの「FREContextInitializer()」を参照してください。

nativeData ネイティブデータへのポインター。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数は成功しました。

FRE_INVALID_ARGUMENT nativeData パラメーターは NULL です。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明

拡張コンテキストのネイティブデータを設定するには、この関数を呼び出します。

関連項目

97 ページの「FREGetContextNativeData()」

18 ページの「コンテキスト固有データ」

FRESetObjectProperty()

AIR 3.0 以降

使用方法

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

パラメーター

object FREObject。プロパティが設定される ActionScript クラスオブジェクトを表します。

propertyName uint8_t 配列。この配列には設定するプロパティの名前を表す文字列が含まれます。この文字列には UTF-8 エンコーディングを使用し、null 文字で終了します。

propertyValue FREObject。プロパティが設定される値を表します。

thrownException FREObject へのポインター。このメソッドを呼び出すことによりランタイムが ActionScript 例外をスローした場合、この FREObject 変数は、ActionScript Error オブジェクトまたは Error サブクラスのオブジェクトを表します。エラーが発生しない場合、ランタイムはこの FREObject 変数を無効に設定します。つまり、thrownException FREObject 変数に FREGetObjectType() を実行すると、FRE_INVALID_OBJECT が返されます。例外情報の処理を行わない場合は、このポインターを NULL に設定できます。

戻り値

FREResult。戻り値には、以下が含まれます（ただし、これらに限定されません）。

FRE_OK 関数が成功し、ActionScript クラスオブジェクトのプロパティが正しく設定されています。

FRE_ACTIONSCRIPT_ERROR ActionScript エラーが発生しました。ランタイムは thrownException パラメーターを、ActionScript Error クラスオブジェクトまたは Error サブクラスオブジェクトを表すように設定します。

FRE_ILLEGAL_STATE 拡張コンテキストは既に ActionScript BitmapData または ByteArray オブジェクトを取得しています。この BitmapData または ByteArray オブジェクトを解放するまでは、コンテキストはこのメソッドを呼び出すことはできません。

FRE_INVALID_ARGUMENT propertyName パラメーターは NULL です。

FRE_INVALID_OBJECT object または propertyValue パラメーターは無効な FREObject 変数です。

FRE_NO_SUCH_NAME propertyName パラメーターは object パラメーターが表す ActionScript クラスオブジェクトのプロパティと一致しません。可能性は低いですが、この戻り値の理由は他にもあります。具体的には、ActionScript クラスに

2 つの同名のプロパティがあるものの、これらが異なる ActionScript 名前空間のものであるという、稀なケースが考えられます。

FRE_READ_ONLY 設定するプロパティは読み取り専用プロパティです。

FRE_TYPE_MISMATCH FREObject object パラメーターは ActionScript クラスオブジェクトを表していません。

FRE_WRONG_THREAD ランタイムにネイティブ拡張関数に対する未実行の呼び出しがあるスレッドとは別のスレッドから、メソッドが呼び出されました。

説明

FREObject 変数が表す ActionScript クラスオブジェクトの public プロパティの値を設定するには、この関数を呼び出します。propertyName パラメーターに、設定するプロパティの名前を渡します。propertyValue パラメーターに、新しいプロパティ値を渡します。

関連項目

102 ページの「FREGetObjectProperty()」

21 ページの「FREObject 型」

第 9 章：Android Java API リファレンス

Android Java API を使用したネイティブ拡張の例については、「Adobe AIR 用のネイティブ拡張」を参照してください。

インターフェイス

AIR ネイティブ拡張用の Java API には 2 つのインターフェイスが定義されています。拡張にはこれら 2 つのインターフェイスを実装することが必須です。

FREExtension

パッケージ： com.adobe.fre

ランタイムバージョン AIR 3

FREExtension インターフェイスには、ネイティブ拡張内の Java コードを AIR ランタイムがインスタンス化するときに使用するインターフェイスが定義されています。

メソッド

メソッド	説明
void initialize()	拡張の初期化処理時にランタイムから呼び出されます。
FREContext createContext(String contextType)	FREContext オブジェクトを作成します。
void dispose()	拡張が破棄されるときにランタイムから呼び出されます。

すべてのネイティブ拡張ライブラリは FREExtension インターフェイスを実装する必要があります。拡張記述ファイルの <initializer> エレメントには、実装するクラスの完全修飾名を指定する必要があります。

例えば、拡張の Java コードが ExampleExtension.jar という JAR ファイルにパッケージされている場合、FREExtension を実装するクラスの名前が com.example.ExampleExtension だとすると、拡張記述ファイルに指定するエントリは次のようになります。

```
<platform name="Android-ARM">
    <applicationDeployment>
        <nativeLibrary>ExampleExtension.jar</nativeLibrary>
        <initializer>com.example.ExampleExtension</initializer>
    </applicationDeployment>
</platform>
```

<finalizer> エレメントは、Java 拡張インターフェイスを使用する場合には不要です。

メソッドの詳細

createContext

```
FREContext createContext( String contextType )
```

FREContext オブジェクトを作成します。

拡張によって ActionScript ExtensionContext.createExtensionContext() メソッドが呼び出されると、AIR ランタイムによって Java createContext() メソッドが呼び出されます。このメソッドから返されたコンテキストは、以後、ランタイムによるメソッド呼び出しに使用されます。

この関数では、普通、ActionScript 側から呼び出すことができるネイティブ実装内のメソッドセットを指定するために contextType パラメーターを使用します。各コンテキストタイプは様々なメソッドセットに対応します。拡張内では、単一のコンテキストを作成することも、同一の機能セットを備えた複数のコンテキストインスタンスを作成することも、また、異なる機能セットを備えた複数のコンテキストインスタンスを作成することもできます。

パラメーター：

contextType コンテキストタイプを特定する文字列。拡張で、必要に応じてこの文字列を定義します。コンテキストタイプでは、拡張の ActionScript 側とネイティブ側とで合意した意味を指定できます。コンテキストタイプを使用しない拡張の場合は、この値に Null を設定することもできます。この値は、null の終端文字を含む UTF-8 エンコーディングの文字列です。

戻り値：

FREContext 拡張コンテキストオブジェクト。

例：

次の例は、contextType パラメーターに応じたコンテキストオブジェクトを返します。contextType が文字列「TypeA」の場合、この関数は、呼び出されるたびに異なる一意の FREContext インスタンスを返します。contextType がその他の値の場合は、FREContext インスタンスを 1 つだけ作成し、プライベート変数 bContext に格納します。

```
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()
```

dispose() メソッドは、その FREExtension 実装で作成されたリソースをクリーンアップするために使用します。関連付けられている ActionScript ExtensionContext オブジェクトが破棄されるかガベージコレクションの対象になると、AIR ランタイムによってこのメソッドが呼び出されます。

initialize

```
void initialize()
```

拡張の初期化処理時に AIR ランタイムから呼び出されます。

クラスの例

次に示す例は、FREContext オブジェクトを 1 つ作成する単純な FREExtension です。また、この例では、Android Log クラスを使用して情報メッセージを Android システムログに出力する方法がわかります（このログは adb logcat コマンドで参照できます）。

```
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

パッケージ： com.adobe.fre

ランタイムバージョン AIR 3

FREFunction インターフェイスには、ネイティブ拡張内に定義された Java 関数をランタイムが呼び出すときに使用するインターフェイスが定義されています。

メソッド

メソッド	説明
FREObject call(FREContext ctx, FREObject[] args)	拡張の ActionScript 側からこの関数が呼び出されたときに、AIR ランタイムによって呼び出されます。

FREFunction インターフェイスは、拡張内にある Java 関数のうち、ネイティブ拡張ライブラリの ActionScript 側から呼び出せるものすべてに実装します。この機能を提供する FREContext インスタンスの getFunctions() から返される Java Map オブジェクトに、このクラスを追加します。

ActionScript 側にある ExtensionContext インスタンスの call() メソッドが実行されると、ランタイムによって call() メソッドが呼び出されます。

FREFunction をコンテキストの関数マップに追加するときには、文字列値をキーとして指定します。関数を ActionScript から呼び出すときは、それと同じ値を指定します。

メソッドの詳細

call

```
FREObject call( FREContext ctx, FREObject[] args )
```

この関数が ActionScript から呼び出されたときに、ランタイムによって呼び出されます。call() メソッドでは、FREFunction インスタンスによって提供される機能を実装（または起動）します。

パラメーター：

ctx この拡張コンテキストを表す FREContext 変数。

ctx パラメーターは次の場合に使用します。

- 拡張コンテキストに関連付けたデータを取得または設定する場合。
- FREContext dispatchStatusEventAsync() メソッドを使用して、ActionScript 側の ExtensionContext インスタンスに非同期イベントを送出する場合。

args FREObject 変数の配列として関数に渡す引数。配列内のオブジェクトは、Java 関数の呼び出しに使用された ActionScript ExtensionContext call() メソッドの引数です。

戻り値：

FREObject 結果。拡張の Java 側と ActionScript 側で共有されるオブジェクトは、すべて、1 個の FREObject または FREObject サブクラス内にカプセル化されます。

クラスの例

次に示す例は、1 個の文字列引数を取り、その中の文字を逆に並べた新しい文字列を返す関数です。call() 関数では、関数呼び出しが行われたコンテキストから ID 文字列を取得するために ctx パラメーターが使用されています。

```
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();
    }
}
```

クラス

AIR ネイティブ拡張用の Java API に定義されている抽象クラス FREContext は、拡張コンテキストを定義するために使用します。また、この API には、具象クラスとして FREObject および数種類の FREObject サブクラスが定義されています。これらは、ネイティブ拡張内の Java 側と ActionScript 側で共有されるオブジェクトを表現するものです。拡張は、FREContext の具象サブクラスのうち少なくとも 1 つを実装する必要があります。

FREArray

パッケージ： com.adobe.fre

継承 FREObject

ランタイムバージョン AIR 3

FREArray クラスは、ActionScript の Array または Vector オブジェクトを表します。

メソッド

メソッド	説明
public static FREArray newArray (String classname, int numElements, boolean fixed)	ActionScript Vector 配列オブジェクトを作成します。
public static FREArray newArray (int numElements)	ActionScript Array オブジェクトを作成します。
public long getLength()	配列要素の数を取得します。
public void setLength(long length)	配列の長さを変更します。
public FREObject getObjectAt(long index)	指定したインデックス位置のオブジェクトを取得します。
public void setObjectAt(long index, FREObject value)	配列内の指定したインデックス位置にオブジェクトを格納します。

FREArray オブジェクトに対する操作には、FREArray クラスに定義されているメソッドと、FREObject クラス（FREArray のスーパークラス）に定義されているメソッドを使用できます。FREObject の getProperty() メソッドと setProperty() メソッドを使用すると、Array および Vector クラスの、ActionScript で定義されたプロパティに対するアクセスまたは変更操作ができます。callMethod() を使用すると、ActionScript で定義されたメ ッドを呼び出すことができます。

メソッドの詳細

newArray

```
public static FREArray newArray (String classname, int numElements, boolean fixed)
```

ActionScript Vector 配列オブジェクトを作成します。

パラメーター：

classname Vector 配列のメンバーが属する ActionScript クラスの完全修飾名。

numElements その配列のために確保する要素の数。

fixed true の場合、その Vector の長さは変更できません。

戻り値：

FREArray ActionScript Vector 配列オブジェクトに関連付けられた FREArray オブジェクト。

例：

```
FREArray vector = FREArray.newArray( "flash.geom.Matrix3D", 4, true );
```

newArray

```
public static FREArray newArray (int numElements)
```

ActionScript Array オブジェクトを作成します。

パラメーター：

numElements その配列のために確保する要素の数。各要素は未定義です。

戻り値：

FREArray ActionScript Array オブジェクトに関連付けられた FREArray オブジェクト。

例：

```
FREArray array = FREArray.newArray( 4 );
```

getLength

```
public long getLength()
```

配列要素の数を取得します。

戻り値：

long 配列の長さ。

例：

```
long length = asArray.getLength();
```

setLength

```
public void setLength( long length )
```

配列の長さを変更します。変更後の長さが現在よりも短い場合は、配列が切り捨てられます。

パラメーター：

length 配列の新しい長さ。

例：

```
asArray.setLength( 4 );
```

getObjectAt

```
public FREObject getObjectAt( long index )
```

配列要素 1 つを取得します。

パラメーター：

index 取得する配列要素の位置（先頭は 0）。

戻り値：

FREObject 配列内の ActionScript オブジェクトに関連付けられた FREObject インスタンス。

例：

```
FREObject element = asArray.getObjectAt( 2 );
```

setObjectAt

```
public void setObjectAt( long index, FREObject value )
```

配列内の指定したインデックス位置にオブジェクトを格納します。

パラメーター：

index 配列内にオブジェクトを格納する位置（先頭は 0）。

value 挿入するプリミティブ値または ActionScript オブジェクトを含んだ FREObject。

例：

```
FREObject stringElement = FREObject.newObject("String element value");
FREArray asVector = FREArray.newArray( "String", 1, false );
asVector.setObjectAt( 0, stringElement );
```

FREByteArray

パッケージ： com.adobe.fre

継承 FREObject

ランタイムバージョン AIR 3

FREByteArray クラスは、ActionScript ByteArray オブジェクトを表します。

メソッド

メソッド	説明
public static FREByteArray newByteArray()	空の ActionScript ByteArray オブジェクトを作成します。
public long getLength()	バイト配列の長さをバイト単位で返します。
public ByteBuffer getBytes()	ActionScript ByteArray オブジェクトの内容を Java ByteBuffer として取得します。
public void acquire()	ActionScript オブジェクトに対するロックを取得します。
public void release()	ActionScript オブジェクトに対するロックを解放します。

ByteArray オブジェクト内のデータにアクセスするには、getBytes() を呼び出します。ActionScript によって参照されている配列内のバイトデータにアクセスする場合は、前もって acquire() を呼び出してオブジェクトをロックする必要があります。データのアクセスまたは変更が完了した後は、release() を呼び出してロックを解放します。

配列に対するロックを保持している間は、そのバッファー内のデータに変更を加えることができますが、配列のサイズは変更できません。配列のサイズを変更するには、ロックを解放し、setProperty() メソッド（FREObject スーパークラス内に定義されています）を使用して、ActionScript で定義された length プロパティを変更します。getProperty()、setProperty() および callMethod() 関数を使用すると、ActionScript ByteArray クラスに定義されたすべてのプロパティやメソッドにアクセスできます。

メソッドの詳細

newByteArray

```
public static FREByteArray newByteArray()
```

空の ActionScript ByteArray オブジェクトと、それに関連付けられた Java FREByteArray インスタンスを作成します。

戻り値：

FREByteArray ActionScript ByteArray を表す FREByteArray オブジェクト。

例：

```
FREBytearray bytearray = FREByteArray.newByteArray();
```

acquire

```
public void acquire()
```

そのオブジェクトに対するロックを取得します。ロックを取得すると、アクセス中にランタイムやアプリケーションコードがそのオブジェクトに変更を加えたり、破棄したりすることはできなくなります。ロックを保持している間、ActionScript で定義されたそのオブジェクトのプロパティの読み取りや変更は一切できません。

getLength

```
public long getLength()
```

バイト配列内のバイト数を返します。このメソッドを呼び出す場合は、前もって、そのオブジェクトの acquire() 関数を呼び出しておく必要があります。

戻り値：

long バイト配列内のバイト数。

getBytes

```
public ByteBuffer getBytes()
```

配列内のバイトデータを返します。このメソッドを呼び出す場合は、前もって acquire() を呼び出してオブジェクトをロックする必要があります。バッファーはロックを保持している間のみ有効です。

java.nio.ByteBuffer バイト配列内のデータ。

例：

```
FREByteArray bytearray = FREByteArray.newByteArray();
bytearray.acquire();
ByteBuffer bytes = bytearray.getBytes();
bytes.putFloat( 16.3 );
bytearray.release();
```

release

```
public void release()
```

acquire() で取得したロックを解放します。getBytes() から返されるデータ以外の FREByteArray プロパティにアクセスするには、ロックが解放されている必要があります。

FREBitmapData

パッケージ： com.adobe.fre

継承 FREObject

ランタイムバージョン AIR 3

FREBitmapData クラスは、ActionScript BitmapData オブジェクトを表します。

メソッド

メソッド	説明
public static FREBitmapData newBitmapData (int width, int height, boolean transparent, Byte[] fillColor)	ActionScript BitmapData オブジェクトを作成します。
public int getWidth()	ビットマップの幅を取得します。
public int getHeight()	ビットマップの高さを取得します。
public boolean hasAlpha()	ビットマップにアルファチャンネルが含まれるかどうかを示します。
public boolean isPremultiplied()	ビットマップのカラーにあらかじめアルファ値が乗算されているかどうかを示します。
public int getLineStride32()	ビットマップのスキャンライン 1 本に含まれる 32 bit 値の個数を取得します。
ByteBuffer getBits()	ビットマップのピクセルデータを取得します。
public void acquire()	ActionScript オブジェクトに対するロックを取得します。
void invalidateRect(int x, int y, int width, int height)	ビットマップ内の長方形領域を、更新が必要な部分としてマークします。
public void release()	ActionScript オブジェクトに対するロックを解放します。
AIR 3.1 の追加項目：	
public boolean isInvertedY	画像データの行が格納される順番を示します。

FREBitmapData オブジェクトのプロパティは、ほとんどが読み取り専用です。例えば、ビットマップの幅や高さを変更することはできません（これらのプロパティを変更する必要がある場合は、変更後のサイズで FREBitmapData オブジェクトを新規作成します）。ただし、ビットマップ内に既にあるピクセルの値に対しては、getBits() メソッドを使用してアクセスまたは変更ができます。また、BitmapData オブジェクト内の ActionScript で定義されたプロパティには FREObjectgetProperty() メソッドと setProperty() メソッドでアクセスでき、オブジェクトの ActionScript メソッドは FREObject callMethod() 関数を使用して呼び出すことができます。

ActionScript コードによって参照されている FREBitmapData オブジェクトに対して getBits() を呼び出す場合は、acquire() メソッドでビットマップをロックしておく必要があります。ロックすると、その関数を実行する間、AIR ランタイムやアプリケーション（別のスレッドで動作している可能性もあります）によって、そのビットマップオブジェクトが変更または破棄されることはなくなります。ビットマップに変更を加えた後は、ビットマップが変化したことをランタイムに伝えるために invalidateRect() を呼び出し、release() を呼び出してロックを解放します。ロックを保持している間は、ActionScript で定義されたプロパティやメソッドを使用することはできません。

AIR ランタイムの定義により、ピクセルデータは、3 原色とアルファの成分を ARGB の順に並べた 32 bit 値として表現されています。データ内の各成分は 1 バイトです。有効なビットマップデータの場合、格納された各カラー成分の値にはアルファ成分の値があらかじめ乗算されています（ただし、場合によっては、アルファが乗算されていない技術的に無効な BitmapData を受け取る可能性があります。例えば、Context3D クラスを使用してビットマップのレンダリングを実行すると、そのような無効なビットマップが生成されることがあります。こうした場合にも、isPremultiplied() メソッドの戻り値は true になります）。

メソッドの詳細

newBitmapData

```
public static FREBitmapData newBitmapData (int width, int height, boolean transparent, Byte[] fillColor)
```

FREBitmapData オブジェクトを作成します。このオブジェクトは、BitmapData インスタンスとして拡張の ActionScript 側に返すことができます。

パラメーター：

width ピクセル単位の幅。AIR 3+ では、ビットマップの幅と高さに恣意的な制限はありません（ActionScript の符号付き整数の上限値のみ）。ただし、実際に使用できるメモリの制約は存在します。ビットマップデータでは、1 ピクセルにつき 32 bit のメモリが使用されます。

height ピクセル単位の高さ。

transparent そのビットマップでアルファチャンネルが使用されているかどうかを示します。このパラメーターは、作成した FREBitmapData の hasTransparency() メソッドと、ActionScript BitmapData オブジェクトの transparent プロパティに対応します。

fillColor 新しいビットマップの全体に設定する 32 bit の ARGB カラー値。transparent パラメーターが false の場合、このカラーのアルファ成分は無視されます。

戻り値：

FREBitmapData ActionScript BitmapData オブジェクトに関連付けられている FREBitmapData オブジェクト。

例：

```
Byte[] fillColor = {0xf,0xf,0xf,0xf,0xf,0xf,0xf,0xf};
FREBitmapData bitmap = FREBitmapData.newBitmapData( 320, 200, true, fillColor );
```

getWidth

```
public int getWidth()
```

ビットマップの幅をピクセル単位で返します。この値は、ActionScript BitmapData クラスオブジェクトの width プロパティに対応します。このメソッドを呼び出す場合は、前もって、そのオブジェクトの acquire() 関数を呼び出しておく必要があります。

戻り値：

int ビットマップの横軸方向のサイズをピクセル単位で表した数値。

getHeight

```
public int getHeight()
```

ビットマップのピクセル単位の高さ。この値は、ActionScript BitmapData クラスオブジェクトの height プロパティに対応します。このメソッドを呼び出す場合は、前もって、そのオブジェクトの acquire() 関数を呼び出しておく必要があります。

戻り値：

int ビットマップの縦軸方向のサイズをピクセル単位で表した数値。

hasAlpha

```
public boolean hasAlpha()
```

そのビットマップでピクセル単位の透明効果がサポートされているかどうかを示します。この値は、ActionScript BitmapData クラスオブジェクトの transparent プロパティに対応します。この値が true の場合、ピクセルカラー値のアルファ成分が使用されます。false の場合、カラーのアルファ成分は使用されません。このメソッドを呼び出す場合は、前もって、そのオブジェクトの acquire() 関数を呼び出しておく必要があります。

戻り値：

boolean true の場合、そのビットマップではアルファチャンネルが使用されています。

isPremultiplied

```
public boolean isPremultiplied()
```

ビットマップのピクセルに乗算済みカラー値が格納されているかどうかを示します。true の場合は、ピクセルカラーの赤、緑、青のそれぞれの成分にアルファ成分の値があらかじめ乗算されています。現時点では、ActionScript BitmapData オブジェクトのデータは常に乗算済みです（将来的には、乗算済みでないビットマップを作成できるような変更がランタイムに加えられる可能性があります）。乗算済みカラー値について詳しくは、『Adobe Flash Platform 用 ActionScript 3.0 リファレンスガイド』で BitmapData.getPixel() の項を参照してください。このメソッドを呼び出す場合は、前もって、そのオブジェクトの acquire() 関数を呼び出しておく必要があります。

戻り値：

boolean true の場合、ピクセルの各カラー成分にはアルファ成分の値があらかじめ乗算されています。

getLineStride32

```
public int getLineStride32()
```

ビットマップの水平ライン 1 本あたりのデータ量を示します。この値は、getBits() から返されるバイト配列を使用して画像を解析する場合に使用します。例えば、バイト配列内の位置 n にあるピクセルのすぐ下にあるピクセルを表すデータの位置は、n + getLineStride32() です。このメソッドを呼び出す場合は、前もって、そのオブジェクトの acquire() 関数を呼び出しておく必要があります。

戻り値：

int 画像の水平ライン 1 本あたりのデータ量。

例：

次の例では、ピクセルのカラー値を含んだバイト配列の位置を、ビットマップ内の 3 本目のラインの先頭に移動します。

```
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()
```

ピクセルカラー値の配列を返します。このメソッドを使用する前には acquire() を呼び出し、データの変更が完了した後には release() を呼び出す必要があります。ピクセルデータに変更を加えた場合は、ビットマップが変化したことを AIR ランタイムに伝えるために invalidateRect() を呼び出します。そうしないとグラフィックの表示は更新されません。

戻り値：

java.nio.ByteBuffer 画像データ。

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()
```

その画像に対するロックを取得します。ロックを取得すると、アクセス中にランタイムやアプリケーションコードがその画像に変更を加えたり、破棄したりすることはできなくなります。ロックを取得する必要があるのは、拡張の ActionScript 側から関数に渡されたビットマップを操作する場合のみです。

ロックを保持している間は、getBits() 関数から返されたデータに対するアクセスのみ可能です。FREBitmapData の他のプロパティに対してアクセスや変更を試みると、例外が発生します。

invalidateRect

```
void invalidateRect( int x, int y, int width, int height )
```

画像内の何らかの領域に変更を加えたことをランタイムに伝えます。このメソッドを呼び出さないと、ビットマップデータオブジェクトの変更内容が画面に反映されません。このメソッドを呼び出す場合は、前もって、そのオブジェクトの acquire() 関数を呼び出しておく必要があります。

パラメーター：

x 現在の表示を無効にする長方形領域の左上隅を表す座標。

y 現在の表示を無効にする長方形領域の左上隅を表す座標。

width 現在の表示を無効にする長方形領域の幅。

height 現在の表示を無効にする長方形領域の高さ。

release

```
public void release()
```

acquire() で取得したロックを解放します。getBits() から返されるピクセルデータ以外のビットマッププロパティにアクセスするには、ロックが解放されている必要があります。

isInvertedY

```
public boolean isInvertedY()
```

画像データの行が格納される順番を示します。true の場合、画像データの最後の行は getBits() メソッドから返されるバッファの最初に表示されます。Android の画像は一般的にはデータの最初の行から順番に格納されるので、Android プラットフォームではこのプロパティは通常 false になります。

AIR 3.1 の追加項目です。

FREContext

パッケージ： com.adobe.fre

継承 java.lang.Object

ランタイムバージョン AIR 3

FREContext クラスは、AIR ネイティブ拡張で定義された Java 実行コンテキストを表します。

メソッド

メソッド	説明
public abstract Map<String,FREFunction> getFunctions()	そのコンテキストでどのような関数が提供されているかを調べるために、AIR ランタイムによって呼び出されます。
public FREObject getActionScriptData()	FREContext に関連付けられている ActionScript ExtensionContext オブジェクトの actionScriptData プロパティからデータを取得します。
public void setActionScriptData(FREObject)	FREContext に関連付けられている ActionScript ExtensionContext オブジェクトの actionScriptData プロパティを設定します。
public abstract void dispose()	その FREContext インスタンスに関連付けられている ActionScript ExtensionContext オブジェクトが破棄されたときに AIR ランタイムによって呼び出されます。
public android.app.Activity getActivity()	アプリケーション Activity を返します。
public native int getResourceId(String resourceString)	Android リソースの ID を返します。
public void dispatchStatusEventAsync(String code, String level)	その FREContext インスタンスに関連付けられている ExtensionContext オブジェクトを介して、アプリケーションに StatusEvent を送出します。

拡張内には、FREContext クラスの具象サブクラスを 1 つまたは複数定義する必要があります。拡張の ActionScript 側で ExtensionContext.createExtensionContext() メソッドが呼び出されると、FREExtension クラスの実装に含まれる createContext() メソッドにより、それらのクラスのインスタンスが作成されます。

拡張によって提供されるコンテキストの整理は様々な方法で行うことができます。例えば、コンテキストクラスのインスタンスを 1 つ作成し、アプリケーションが存続する期間中ずっとそれを保持することができます。または、存続期間の限られている 1 セットのネイティブオブジェクトと緊密に結びついたコンテキストクラスを 1 つ作成することもできます。その場合は、個々のオブジェクトセットごとに別々のコンテキストインスタンスを作成し、ネイティブオブジェクトとそれに関連付けられているコンテキストインスタンスを一緒に破棄するという使用方法になります。

注意： 何らかの FREByteArray または FREBitmapData オブジェクトに対して acquire() メソッドを呼び出した後は、どのオブジェクトの FREObject クラスで定義されたメソッドを呼び出すこともできません。

メソッドの詳細

getFunctions

```
public abstract Map<String,FREFunction> getFunctions()
```

この関数には、そのコンテキストでアクセス可能とされるすべての関数を含んだ Java Map インスタンスを返すコードを実装します。マップのキー値は、何らかの関数が呼び出されたときに FREFunction オブジェクトを検索するために使用されます。

戻り値：

Map<String, FREFunction> 文字列のキー値 1 つとそれに対応する FREFunction オブジェクト 1 つを含んだ Map オブジェクト。この Map オブジェクトは、拡張の ActionScript 側で ExtensionContext クラスの call() メソッドが呼び出されたとき、該当する関数を AIR ランタイムが検索するために使用されます。

例：

次に示す getFunctions() の例では、架空の関数 3 つを含んだ Java HashMap を作成します。これらの関数を拡張の ActionScript 側から呼び出すには、文字列「negate」、「invert」、「reverse」のいずれかを ExtensionContext call() メソッドに渡します。

```
@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()
```

そのコンテキストに関連付けられている ActionScript データオブジェクトを取得します。このデータオブジェクトに対しては、Java コードと ActionScript コードのどちらからでも読み取りと変更ができます。

戻り値：

FREObject データオブジェクトを表す FREObject。

例：

```
FREObject data = context.getActionScriptData();
```

setActionScriptData

public void setActionScriptData(FREObject object)

そのコンテキストに関連付けられている ActionScript データオブジェクトを設定します。このデータオブジェクトに対しては、Java コードと ActionScript コードのどちらからでも読み取りと変更ができます。

パラメーター：

value プリミティブ値または ActionScript オブジェクトを含んだ FREObject。

例：

```
FREObject data = FREObject.newObject( "A string" );
context.setActionScriptData( data );
```

getActivity

```
public android.app.Activity getActivity()
```

アプリケーション Activity オブジェクトへの参照を取得します。この参照は、Android SDK の多くの API を使用するために必要です。

戻り値：

android.app.Activity AIR アプリケーションの Activity オブジェクト。

例：

```
Activity activity = context.getActivity();
```

getResourceID

```
public native int getResourceId( String resourceString )
```

ネイティブ Android リソースのリソース ID を取得します。

resourceString パラメーターには、リソースを特定する文字列を指定します。Android 上のリソースにアクセスする場合の命名規則に従います。例えば、あるリソースに対し、Android ネイティブコードでは次のようにアクセスするとします。

```
R.string.my_string
```

このリソースの resourceString は次のとおりです。

```
"string.my_string"
```

このメソッドでは、resourceString で指定されたリソースが見つからない場合は Resources.NotFoundException をスローします。

パラメーター：

resourceString Android リソースを特定する文字列。

戻り値：

int Android リソースの ID。

例：

```
int myResourceID = context.getResourceID( "string.my_string" );
```

dispatchStatusEventAsync

```
public void dispatchStatusEventAsync( String code, String level )
```

ActionScript StatusEvent イベントを送出するには、この関数を呼び出します。イベントのターゲットは、ランタイムによって FREContext オブジェクトに関連付けられた ActionScript ExtensionContext インスタンスです。

通常、この関数が送出するイベントは非同期です。例えば、拡張メソッドから、タスクを実行するための別のスレッドを開始させることができます。他のスレッドのタスクが完了したときには、そのスレッドから、ActionScript ExtensionContext インスタンスに完了を知らせるために dispatchStatusEventAsync() が呼び出されます。

注意：dispatchStatusEventAsync() 関数は、ネイティブ実装内のいずれのスレッドからも呼び出すことができる唯一の Java API です。

次の場合には、ランタイムはイベントを送出しません。

- ランタイムがすでに ExtensionContext インスタンスを破棄している場合。
- ランタイムが ExtensionContext インスタンスの破棄を処理中である場合。
- ExtensionContext インスタンスが参照を持っていない場合。ランタイムのガベージコレクターがインスタンスを破棄できる状態にあります。

code および level パラメーターに、任意の文字列値を設定します。必要に応じてどのような値を設定することもできますが、拡張の ActionScript 側とは合意が成立している必要があります。

パラメーター：

code 報告されるステータスの説明。

level 拡張によって定義された、メッセージのカテゴリー。

例：

```
context.dispatchStatusEventAsync( "Processing finished", "progress" );
```

dispose

```
public abstract void dispose()
```

関連付けられている ActionScript ExtensionContext オブジェクトが破棄され、その FREContext オブジェクトに他の参照が何もない場合に AIR ランタイムによって呼び出されます。このメソッドには、コンテキストリソースをクリーンアップするコードを実装できます。

クラスの例

次の例では、コンテキスト情報を返す FREContext のサブクラスを作成します。

```
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

パッケージ： com.adobe.fre

継承 java.lang.Object

サブクラス FREArray、FREBitmapData、FREByteArray

ランタイムバージョン AIR 3

FREObject クラスは、Java コード内において ActionScript オブジェクトを表します。

メソッド

メソッド	説明
public static FREObject newObject(int value)	32 bit の符号付き整数値を含んだ FREObject を作成します。
public static FREObject newObject(double value)	Java の double 値（ActionScript の Number 型に相当する値）を含んだ FREObject を作成します。
public static FREObject newObject(boolean value)	boolean 値を含んだ FREObject を作成します。
public static FREObject newObject(String value)	文字列値を含んだ FREObject を作成します。
public int getAsInt()	FREObject 内のデータに Java の int 値としてアクセスします。
public double getAsDouble()	FREObject 内のデータに Java の double 値としてアクセスします。
public Boolean getAsBool()	FREObject 内のデータに Java の boolean 値としてアクセスします。
public String getAsString()	FREObject 内のデータに Java の String 値としてアクセスします。
public static native FREObject newObject(String className, FREObject[] constructorArgs)	ActionScript クラスの新規インスタンスを参照する FREObject を作成します。
public FREObject getProperty(String propertyName)	ActionScript プロパティの値を取得します。
public void setProperty(String propertyName, FREObject propertyValue)	ActionScript プロパティの値を設定します。
public FREObject callMethod(String methodName, FREObject[] methodArgs)	ActionScript メソッドを呼び出します。

拡張内の Java コードと ActionScript コードでデータを共有するには、FREObject を使用します。FREObject 変数とそれに対応する ActionScript オブジェクトとの関連付けはランタイムによって行われます。FREObject に関連付けられた ActionScript オブジェクトのプロパティやメソッドにアクセスするには、getProperty()、setProperty() および callMethod() 関数を使用します。

FREObject は汎用的なオブジェクトで、プリミティブ値を表現することも、クラスタイプを表現することもできます。オブジェクト内のデータと互換性のない方法でオブジェクトにアクセスした場合は、FRETypeMismatchException がスローされます。

FREObject は、任意の ActionScript オブジェクトを表現するために使用できます。また、サブクラスの FREArray、FREBitmapData および FREByteArray には、特定の種類のデータを取り扱うためのメソッドが追加されています。

Java から ActionScript コードにデータを渡すには、FREFunction インスタンスの call() メソッドから取得した FREObject を返します。そのデータは、ActionScript コードには ActionScript Object として渡され、必要に応じて、適切なプリミティブまたはクラスタイプにキャストされます。また、FREObject 参照によって示される ActionScript オブジェクトに対してプロパティの設定やメソッドの呼び出しを行うこともできます。ActionScript メソッドのプロパティまたはパラメーターを設定するには FREObject を使用します。

ActionScript から Java コードにデータを渡すには、FREFunction インスタンスの call() メソッドにパラメーターとして渡します。この引数は、call() メソッドが呼び出されたとき FREObject インスタンス内にカプセル化されます。引数が何らかの ActionScript オブジェクトに対する参照である場合、Java コードでは、そのオブジェクトに対してプロパティ値の読み

取りやメソッドの呼び出しを実行できます。それらのプロパティおよび関数戻り値は、Java コードには FREObject として渡されます。FREObject の各種メソッドを使用すると、データに Java タイプとしてアクセスすることや、該当する場合にはオブジェクトを FREObject サブクラス（FREBitmapData など）にダウンキャストすることができます。

メソッドの詳細

newObject(int)

```
public static FREObject newObject( int value )
```

32 bit の符号付き整数値を含んだ FREObject を作成します。

パラメーター：

value 符号付き整数。

戻り値：

FREObject FREObject。

例：

```
FREObject value = FREObject.newObject( 4 );
```

newObject(double)

```
public static FREObject newObject( double value )
```

Java の double 値（ActionScript の Number 型に相当する値）を含んだ FREObject を作成します。

パラメーター：

value double 値。

戻り値：

FREObject FREObject。

例：

```
FREObject value = FREObject.newObject( 3.14156d );
```

newObject(boolean)

```
public static FREObject newObject( boolean value )
```

boolean 値を含んだ FREObject を作成します。

パラメーター：

value true または false。

戻り値：

FREObject FREObject。

例：

```
FREObject value = FREObject.newObject( true );
```

newObject(String)

```
public static FREObject newObject( String value )
```

文字列値を含んだ FREObject を作成します。

パラメーター：

value 文字列。

戻り値：

FREObject FREObject。

例：

```
FREObject value = FREObject.newObject( "A string value" );
```

getAsInt

```
public int getAsInt()
```

FREObject 内のデータに Java の int 値としてアクセスします。

戻り値：

int 整数値。

例：

```
int value = FREObject.getAsInt();
```

getAsDouble

```
public double getAsDouble()
```

FREObject 内のデータに Java の double 値としてアクセスします。

戻り値：

double double 値。

例：

```
double value = FREObject.getAsInt();
```

getAsBool

```
public Boolean getAsBool()
```

FREObject 内のデータに Java の boolean 値としてアクセスします。

戻り値：

boolean true または false

例：

```
boolean value = FREObject.getAsInt();
```

getAsString

```
public String getAsString()
```

FREObject 内のデータに Java の String 値としてアクセスします。

戻り値：

String 文字列値。

例：

```
String value = FREObject.getAsInt();
```

newObject(String, FREObject[])

```
public static native FREObject newObject( String className, FREObject[] constructorArgs )
```

ActionScript クラスの新規インスタンスを参照する FREObject を作成します。

パラメーター：

className ActionScript クラスの完全修飾クラス名。

constructorArgs ActionScript クラスのコンストラクターに渡す引数（FREObject の配列）。クラスのコンストラクターにパラメーターがない場合は、null を指定します。

戻り値：

FREObject ActionScript クラスの新規インスタンスを表す FREObject。

例：

```
FREObject matrix = FREObject.newObject( "flash.geom.Matrix", null );
```

getProperty

```
public FREObject getProperty( String propertyName )
```

ActionScript プロパティの値を取得します。

パラメーター：

propertyName アクセスするプロパティの名前。

戻り値：

FREObject 指定したプロパティの値（FREObject）。

例：

```
FREObject isDir = fileobject.getProperty( "isDirectory" );
```

setProperty

```
public void setProperty( String propertyName, FREObject propertyValue )
```

ActionScript プロパティの値を設定します。

パラメーター：

propertyName 設定するプロパティの名前。

propertyValue 設定するプロパティ値を含んだ FREObject。

例：

```
fileobject.setProperty( "url", FREObject.newObject( "app://file.txt" ) );
```

callMethod

```
public FREObject callMethod( String methodName, FREObject[] methodArgs )
```

ActionScript メソッドを呼び出します。

パラメーター：

methodName 呼び出すメソッドの名前。

methodArgs メソッド呼び出しの引数を含んだ FREObject 配列。配列要素はメソッドパラメーターの宣言順に並べる必要があります。

戻り値：

FREObject メソッドの結果。ActionScript メソッドから Array、Vector または BitmapData のオブジェクトが返された場合は、結果を適切な FREObject サブクラスにキャストして使用します。

例：

```
FREObject[] args = new FREObject[1]
args[0] = FREObject.newObject( "assets/image.jpg" );
FREObject imageFile = directoryobject.callMethod( "resolvePath", args );
```