ADOBE® AIR® 用 HTML 開発ガイド

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。

最終更新日 2011/10/12

目次

第 1 章：HTML 環境について

Adobe AIR 1.0 およびそれ以降

AIR では、Safari Web ブラウザーでも使用される WebKit（**www.webkit.org**）を使用して、HTML および JavaScript のコンテンツの解析、配置およびレンダリングを行います。AIR のビルトインのホストクラスとオブジェクトには、デスクトップアプリケーションに関連付けられている従来の機能のための API が用意されています。このような機能には、ファイルの読み取りと書き込みおよびウィンドウの管理があります。また、Adobe AIR は Adobe® Flash® Player から API を継承しており、これらの API には、サウンドおよびバイナリソケットなどの機能があります。

重要：Adobe AIR ランタイムの新しいバージョンには、WebKit の更新されたバージョンが含まれている場合があります。新しいバージョンの AIR に含まれる WebKit の更新により、デプロイされた AIR アプリケーションに予期しない変更が生じる可能性があります。そのような変更は、アプリケーションの HTML コンテンツの動作や外観に影響を与える場合があります。例えば、WebKit のレンダリングの改善または修正によって、アプリケーションのユーザーインターフェイスの要素のレイアウトが変化する場合があります。そのため、アプリケーションには更新メカニズムを用意することを強くお勧めします。AIR に含まれる WebKit のバージョンに加えられた変更により、アプリケーションを更新する必要がある場合は、AIR の更新メカニズムによって、アプリケーションの新しいバージョンをインストールするかどうかを確認するメッセージが表示されます。

次の表に、AIR で使用されている WebKit と同等のバージョンを使用する Safari Web ブラウザーのバージョンを示します。

AIR バージョン	Safari バージョン
1.0	2.04
1.1	3.04
1.5	4.0 Beta
2.0	4.03
2.5	4.03
2.6	4.03
2.7	4.03
3	5.0.3

インストールされている WebKit のバージョンは、HTMLLoader オブジェクトによって返されるデフォルトのユーザーエージェント文字列を調べることにより、いつでも確認できます。

```
air.trace( window.htmlLoader.userAgent );
```

AIR で使用されている WebKit のバージョンは、オープンソースバージョンと同じであるとは限りません。一部の機能は AIR でサポートされていません。また、AIR バージョンには、対応する WebKit バージョンではまだ利用できないセキュリティ修正やバグ修正が含まれていることがあります。16 ページの「AIR でサポートされていない WebKit の機能」を参照してください。

HTML コンテンツでの AIR API の使用は完全にオプションです。HTML および JavaScript を使用して AIR アプリケーション全体をプログラムできます。既存の HTML アプリケーションの多くは、ほとんど変更することなく実行できます（WebKit との互換性がある HTML、CSS、DOM および JavaScript の機能を使用している場合）。

AIR では、アプリケーションのルックアンドフィールを完全に制御できます。アプリケーションをネイティブデスクトップアプリケーションのようにすることができます。オペレーティングシステムによって提供されるウィンドウクロムをオフにし、ウィンドウの移動、サイズ変更および終了を行うための独自のコントロールを実装できます。ウィンドウなしで実行することもできます。

AIR アプリケーションはファイルシステムへのフルアクセスを使用してデスクトップで直接実行されるので、セキュリティモデルは、一般的な Web ブラウザーのセキュリティモデルより厳格です。AIR では、アプリケーションのインストールディレクトリから読み込まれたコンテンツのみ、アプリケーションサンドボックスに配置されます。アプリケーションサン

ドボックスには最高レベルの権限が設定されており、このサンドボックスから AIR API にアクセスできます。AIR では、その他のコンテンツは、そのコンテンツの生成元に基づいて個別のサンドボックスに配置されます。ファイルシステムから読み込まれたファイルは、ローカルサンドボックスに含められます。http: プロトコルまたは https: プロトコルを使用してネットワークから読み込まれたファイルは、リモートサーバーのドメインに基づいてサンドボックスに含められます。これらの非アプリケーションサンドボックス内のコンテンツは、AIR API へのアクセスが禁止され、一般的な Web ブラウザーでの場合とほぼ同様に実行されます。

AIR の HTML コンテンツでは、アルファ、拡大／縮小または透明度の設定が適用されている場合、SWF コンテンツや PDF コンテンツは表示されません。詳しくは、49 ページの「HTML ページに SWF コンテンツまたは PDF コンテンツを読み込む場合の考慮事項」および 87 ページの「ウィンドウの透明度」を参照してください。

関連項目

Webkit DOM リファレンス

Safari HTML リファレンス

Safari CSS リファレンス

www.webkit.org

HTML 環境の概要

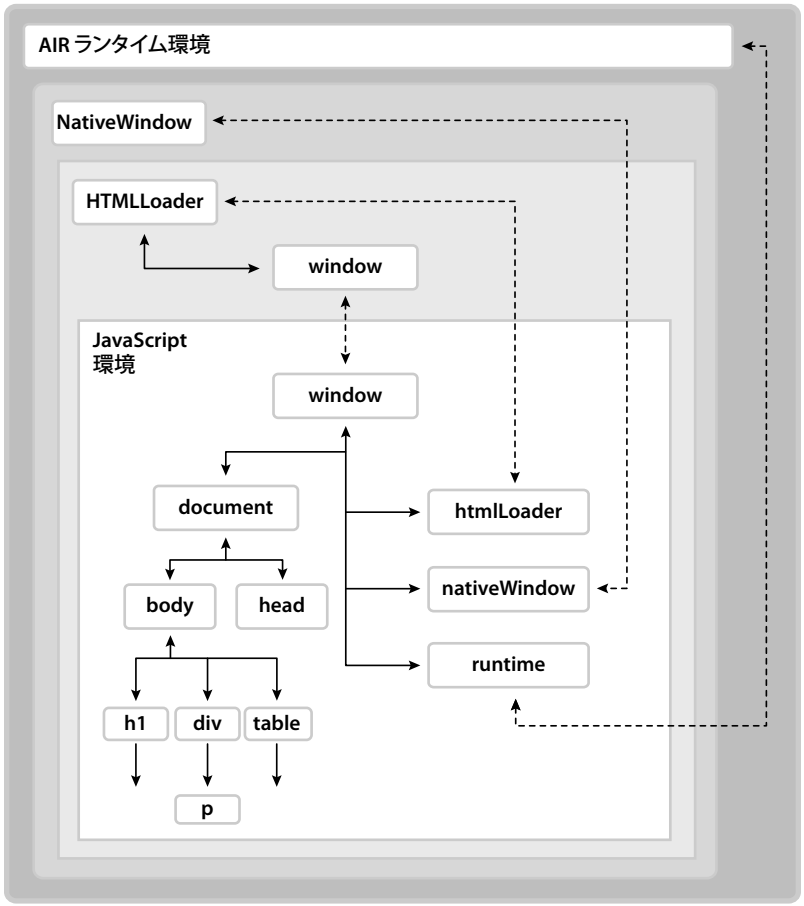
Adobe AIR 1.0 およびそれ以降

Adobe AIR では、HTML レンダラー、ドキュメントオブジェクトモデルおよび JavaScript インタプリタを備えた、完全なブラウザー形式の JavaScript 環境を提供します。この JavaScript 環境を表すのが、AIR の HTMLLoader クラスです。HTML ウィンドウでは、HTMLLoader オブジェクトはすべての HTML コンテンツを格納し、そのオブジェクト自体が NativeWindow オブジェクト内に格納されます。NativeWindow オブジェクトを使用すると、アプリケーションで、ユーザーのデスクトップに表示されるオペレーティングシステムのネイティブウィンドウのプロパティおよび動作のスクリプトを作成できます。

JavaScript 環境および AIR ホストとの関係について

Adobe AIR 1.0 およびそれ以降

次の図に、JavaScript 環境と AIR ランタイム環境の関係を示します。1 つのネイティブウィンドウしか示されていませんが、AIR アプリケーションには複数のウィンドウを含めることができます（また、1 つのウィンドウに複数の HTMLLoader オブジェクトを含めることができます）。



JavaScript 環境には、独自の Document オブジェクトと Window オブジェクトがあります。JavaScript コードでは、runtime、nativeWindow、htmlLoader の各プロパティを使用して、AIR ランタイム環境を操作できます。ActionScript コードでは、JavaScript の Window オブジェクトを参照する、HTMLLoader オブジェクトの window プロパティを使用して、**JavaScript** 環境を操作できます。また、**ActionScript** オブジェクトと **JavaScript** オブジェクトの両方で、**AIR** オブジェクトと **JavaScript** オブジェクトから送出されたイベントをリッスンできます。

runtime プロパティは AIR API クラスへのアクセスを提供します。このプロパティを使用すると、クラスメンバー（静的メンバー）にアクセスできるだけでなく、新しい AIR オブジェクトを作成することもできます。AIR API にアクセスするには、package を使用して、クラスの名前を runtime プロパティに追加します。例えば、File オブジェクトを作成するには、次のステートメントを使用します。

```
var file = new window.runtime.filesystem.File();
```

注意：AIR SDK には、AIRAliases.js という JavaScript ファイルが付属しており、このファイルでは、最もよく使用される AIR クラスの便利なエイリアスが定義されています。このファイルを読み込むと、window.runtime.package.Class の代わりに、短い形式の air.Class を使用できます。例えば、new air.File() を使用して File オブジェクトを作成することもできます。

NativeWindow オブジェクトには、デスクトップウィンドウを制御するためのプロパティが用意されています。HTML ページ内から、window.nativeWindow プロパティを使用して、コンテナとなっている NativeWindow オブジェクトにアクセスできます。

HTMLLoader オブジェクトには、コンテンツを読み込んでレンダリングする方法を制御するためのプロパティ、メソッドおよびイベントが用意されています。HTML ページ内から、window.htmlLoader プロパティを使用して、親 HTMLLoader オブジェクトにアクセスできます。

重要：htmlLoader、nativeWindow、runtime の各プロパティを使用できるのは、アプリケーションの一部としてインストールされたページのみで、トップレベルドキュメントとして読み込まれた場合だけです。これらのプロパティは、ドキュメントがフレームまたはインラインフレームに読み込まれた場合には追加されません（子ドキュメントは、同じセキュリティサンドボックスにある限り、親ドキュメントのこれらのプロパティにアクセスできます。例えば、フレームに読み込まれたドキュメントでは、parent.runtime を使用して、その親の runtime プロパティにアクセスできます）。

セキュリティについて

Adobe AIR 1.0 およびそれ以降

AIR では、すべてのコードは、元のドメインに基づいてセキュリティサンドボックス内で実行されます。アプリケーションコンテンツ（アプリケーションのインストールディレクトリから読み込まれたコンテンツに限定）は、アプリケーションサンドボックスに配置されます。ランタイム環境および AIR API へのアクセスは、このサンドボックス内で実行される HTML および JavaScript からのみ可能です。その際、ページの load イベントのすべてのハンドラーから処理が戻った後、JavaScript の動的な評価と実行の多くはアプリケーションサンドボックス内でブロックされます。

アプリケーションページを非アプリケーションサンドボックスにマップするには、ページをフレームまたはインラインフレームに読み込み、そのフレームの AIR 固有の sandboxRoot 属性と documentRoot 属性を設定します。sandboxRoot 値を実際のリモートドメインに設定すると、サンドボックスに割り当てたコンテンツで、そのドメインのコンテンツをクロススクリプトできるようになります。この方法によるページのマッピングは、マッシュアップアプリケーションなどで、リモートコンテンツの読み込みおよびスクリプト作成を行う場合に役立ちます。

アプリケーションおよび非アプリケーションのコンテンツで互いにクロススクリプトできるようにするもう 1 つの方法、また非アプリケーションコンテンツから AIR API にアクセスできるようにする唯一の方法は、サンドボックスブリッジを作成することです。親から子へのブリッジを使用すると、アプリケーションサンドボックスで定義されている指定のメソッドとプロパティに、子のフレーム、インラインフレームまたはウィンドウのコンテンツからアクセスできます。逆に、子から親へのブリッジを使用すると、子のサンドボックスで定義されている指定のメソッドとプロパティに、アプリケーションコンテンツからアクセスできます。サンドボックスブリッジを確立するには、ウィンドウオブジェクトの parentSandboxBridge プロパティと childSandboxBridge プロパティを設定します。詳しくは、72 ページの「Adobe AIR の HTML セキュリティ」および 12 ページの「HTML の frame エレメントと iframe エレメント」を参照してください。

プラグインと埋め込みオブジェクトについて

Adobe AIR 1.0 およびそれ以降

AIR では、Adobe® Acrobat® プラグインをサポートしています。PDF コンテンツを表示するには、ユーザーは、Acrobat または Adobe® Reader® 8.1 以上を使用する必要があります。HTMLLoader オブジェクトには、ユーザーのシステムで PDF を表示できるかどうかをチェックするためのプロパティが用意されています。SWF ファイルコンテンツも HTML 環境内で表示できますが、この機能は AIR に組み込まれており、外部プラグインは必要ありません。

その他の Webkit プラグインは、AIR ではサポートされていません。

関連項目

72 ページの「Adobe AIR の HTML セキュリティ」

5 ページの「HTML サンドボックス」

12 ページの「HTML の frame エレメントと iframe エレメント」

10 ページの「JavaScript の Window オブジェクト」

6 ページの「XMLHttpRequest オブジェクト」

264 ページの「AIR での PDF コンテンツの追加」

AIR および WebKit

Adobe AIR 1.0 およびそれ以降

Adobe AIR では、Safari Web ブラウザーでも使用されるオープンソースの Webkit エンジンを使用します。AIR では、セキュリティ目的のほかに、ランタイムクラスとオブジェクトにアクセスできるようにするため、いくつかの拡張機能を追加します。また、Webkit 自体でも、HTML、CSS および JavaScript の W3C 標準には含まれていない機能を追加します。

ここでは、AIR の追加機能と Webkit の最も注目される拡張機能についてのみ説明します。非標準の HTML、CSS および JavaScript に関する他のドキュメントについては、www.webkit.org および developer.apple.com/jp を参照してください。標準規格については、W3C Web サイトを参照してください。Mozilla にも、HTML、CSS および DOM のトピックに関する有用な一般リファレンスがあります（ただし、WebKit エンジンと Mozilla エンジンは同一ではありません）。

AIR での JavaScript

Flash Player 9 以降、Adobe AIR 1.0 以降

AIR では、一般的な JavaScript オブジェクトの通常動作にいくつか変更が加えられます。これらの変更の多くは、AIR で安全なアプリケーションを容易に作成できるようにするために行われます。また、この動作の違いによって、JavaScript の一般的なコーディングパターンの一部と、そのパターンを使用している既存の Web アプリケーションが、AIR では期待どおりに実行されない場合があります。このような問題の修正については、23 ページの「セキュリティ関連の JavaScript エラーの回避」を参照してください。

HTML サンドボックス

Adobe AIR 1.0 およびそれ以降

AIR では、コンテンツは、そのコンテンツの生成元に従って個別のサンドボックスに配置されます。サンドボックス規則は、Adobe Flash Player で実装されているサンドボックスの規則と同様、ほとんどの Web ブラウザーで実装されている同一生成元ポリシーに従っています。また、AIR には、アプリケーションコンテンツを格納して保護する、新しいアプリケーションサンドボックスタイプが用意されています。AIR アプリケーションの開発時に扱う場合があるサンドボックスのタイプについて詳しくは、セキュリティサンドボックスを参照してください。

ランタイム環境および AIR API へのアクセスは、アプリケーションサンドボックス内で実行される HTML および JavaScript からのみ可能です。ただし、その際、JavaScript の動的な評価と実行は、様々な形で、セキュリティ上の理由からアプリケーションサンドボックス内で大きく制限されます。これらの制限は、実際にアプリケーションでサーバーから直接情報を読み込むかどうかに関わらず適用されます（ファイルコンテンツ、ペーストされたストリングおよび直接ユーザー入力であっても、信頼できない可能性があります）。

ページ内のコンテンツの生成元によって、割り当て先のサンドボックスが決まります。アプリケーションディレクトリ（app: URL スキームで参照するインストールディレクトリ）から読み込まれたコンテンツのみ、アプリケーションサンドボックスに配置されます。ファイルシステムから読み込まれたコンテンツは、**local-with-file system** サンドボックスまたは **local-trusted** サンドボックスに配置されます。このサンドボックスでは、ローカルファイルシステム上のコンテンツへのアクセスおよび操作は許可されますが、リモートコンテンツへのアクセスおよび操作は許可されません。ネットワークから読み込まれたコンテンツは、元のドメインに対応するリモートサンドボックスに配置されます。

リモートサンドボックス内のコンテンツをアプリケーションページから自由に操作できるようにするには、リモートコンテンツと同じドメインにページをマップします。例えば、インターネットサービスからマップデータを表示するアプリケーションを作成する場合は、そのサービスからコンテンツを読み込んで表示するアプリケーションのページをサービスドメインにマップします。リモートサンドボックスおよびドメインにページをマップするための属性は、HTML の frame エレメントと iframe エレメントに追加された新しい属性です。

非アプリケーションサンドボックス内のコンテンツで安全に AIR 機能を使用できるようにするには、親サンドボックスブリッジを設定します。アプリケーションコンテンツで他のサンドボックス内のコンテンツのメソッドの呼び出しおよびプロパティへのアクセスを安全に行えるようにするには、子サンドボックスブリッジを設定します。この場合の安全とは、明示的に公開していないオブジェクト、プロパティまたはメソッドに、リモートコンテンツから誤って参照しないようにすることを指します。単純データ型、関数および匿名オブジェクトのみ、ブリッジを介して渡すことができます。ただし、危険性のある関数は明示的に公開しないでください。例えば、ユーザーのシステム上の任意の場所にあるファイルの読み取りと書き込みをリモートコンテンツに許可するインターフェイスを公開すると、ユーザーに多大な被害を与える可能性をリモートコンテンツが持つことになります。

JavaScript の eval() 関数

Adobe AIR 1.0 およびそれ以降

ページの読み込みが完了すると、eval() 関数の使用はアプリケーションサンドボックス内で制限されます。JSON 形式のデータを安全に解析できるように一部の使用は許可されますが、実行可能ステートメントを生成する評価はすべてエラーになります。75 ページの「異なるサンドボックス内のコンテンツに対するコードの制限」で、eval() 関数の許可される使用について説明しています。

関数コンストラクター

Adobe AIR 1.0 およびそれ以降

アプリケーションサンドボックスでは、ページの読み込みが完了する前に関数コンストラクターを使用できます。ページの load イベントハンドラーがすべて終了した後は、新しい関数は作成できません。

外部スクリプトの読み込み

Adobe AIR 1.0 およびそれ以降

アプリケーションサンドボックス内の HTML ページでは、script タグを使用してアプリケーションディレクトリ外から JavaScript ファイルを読み込むことはできません。アプリケーションのページでアプリケーションディレクトリ外からスクリプトを読み込むには、そのページを非アプリケーションサンドボックスにマップする必要があります。

XMLHttpRequest オブジェクト

Adobe AIR 1.0 およびそれ以降

AIR には、アプリケーションでデータ要求を行うために使用できる XMLHttpRequest（XHR）オブジェクトが用意されています。次の例に、簡単なデータ要求を示します。

```
xmlhttp = new XMLHttpRequest();
xmlhttp.open("GET", "http:/www.example.com/file.data", true);
xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4) {
        //do something with data...
    }
}
xmlhttp.send(null);
```

ブラウザーとは対照的に、AIR では、アプリケーションサンドボックス内で実行されるコンテンツは任意のドメインのデータを要求できます。JSON ストリングを含む XHR の結果は、実行可能コードもその結果に含まれる場合を除き、データオブジェクトになります。実行可能ステートメントが XHR 結果で表される場合、エラーがスローされ、評価の試行は失敗します。

リモートソースからコードが意図せず挿入されないようにするため、同期 XHR は、ページの読み込みが完了する前に行われた場合は空の結果を返します。非同期 XHR は、常に、ページが読み込まれた後に返します。

デフォルトでは、AIR は、非アプリケーションサンドボックスでのクロスドメイン XMLHttpRequest をブロックします。アプリケーションサンドボックス内の親ウィンドウでは、非アプリケーションサンドボックス内のコンテンツを含む子フレームでのクロスドメイン要求を許可することを選択できます。それには、コンテナとなっている frame エレメントまたは iframe エレメントで、allowCrossDomainXHR（AIR で追加された属性）を true に設定します。

```
<iframe id="mashup"
    src="http://www.example.com/map.html"
    allowCrossDomainXHR="true"
</iframe>
```

注意：必要に応じて、AIR の URLStream クラスを使用してデータをダウンロードすることもできます。

リモートサンドボックスにマップされているアプリケーションコンテンツを含むフレームまたはインラインフレームからリモートサーバーに XMLHttpRequest を送出する場合は、マッピング URL で、XHR で使用するサーバーアドレスをマスクしていないことを確認してください。例えば、example.com ドメインのリモートサンドボックスにアプリケーションコンテンツをマップする、次の iframe 定義について考えます。

```
<iframe id="mashup"
    src="http://www.example.com/map.html"
    documentRoot="app:/sandbox/"
    sandboxRoot="http://www.example.com/"
    allowCrossDomainXHR="true"
</iframe>
```

sandboxRoot 属性は www.example.com アドレスのルート URL を再マップするので、すべての要求はアプリケーションディレクトリから読み込まれ、リモートサーバーからは読み込まれません。要求は、ページナビゲーションと XMLHttpRequest のいずれから発生するかに関わらず、再マップされます。

リモートサーバーへのデータ要求が誤ってブロックされないようにするには、sandboxRoot を、ルートではなくリモート URL のサブディレクトリにマップします。このディレクトリは存在していなくてもかまいません。例えば、www.example.com への要求でアプリケーションディレクトリではなくリモートサーバーからの読み込みを許可するには、上述の iframe を次のように変更します。

```
<iframe id="mashup"
    src="http://www.example.com/map.html"
    documentRoot="app:/sandbox/"
    sandboxRoot="http://www.example.com/air/"
    allowCrossDomainXHR="true"
</iframe>
```

この場合、air サブディレクトリ内のコンテンツのみローカルに読み込まれます。

サンドボックスのマッピングについて詳しくは、12 ページの「HTML の frame エレメントと iframe エレメント」および 72 ページの「Adobe AIR の HTML セキュリティ」を参照してください。

Cookie

Adobe AIR 1.0 およびそれ以降

AIR アプリケーションでは、リモートサンドボックス内のコンテンツ（http: ソースおよび https: ソースから読み込まれたコンテンツ）のみ、Cookie（document.cookie プロパティ）を使用できます。アプリケーションサンドボックスには、EncryptedLocalStore、SharedObject および FileStream クラスなど、永続データを格納するための他の手段が用意されています。

Clipboard オブジェクト

Adobe AIR 1.0 およびそれ以降

WebKit Clipboard API は、copy、cut、paste の各イベントによって動作します。これらのイベントで渡されるイベントオブジェクトでは、clipboardData プロパティを使用してクリップボードにアクセスできます。クリップボードのデータを読み取ったり、書き込んだりするには、clipboardData オブジェクトの次のメソッドを使用します。

メソッド	説明
clearData(mimeType)	クリップボードのデータを消去します。mimeType パラメーターを、消去するデータの MIME タイプに設定します。
getData(mimeType)	クリップボードのデータを取得します。このメソッドは、paste イベントのハンドラーでのみ呼び出すことができます。mimeType パラメーターを、取得するデータの MIME タイプに設定します。
setData(mimeType, data)	データをクリップボードにコピーします。mimeType パラメーターをデータの MIME タイプに設定します。

アプリケーションサンドボックス外の JavaScript コードからクリップボードにアクセスするには、これらのイベントを使用する必要があります。ただし、アプリケーションサンドボックス内のコンテンツでは、AIR の Clipboard クラスを使用してシステムクリップボードに直接アクセスできます。例えば、次のステートメントを使用すると、クリップボードのテキスト形式のデータを取得できます。

```
var clipping = air.Clipboard.generalClipboard.getData("text/plain",
                                air.ClipboardTransferMode.ORIGINAL_ONLY);
```

データの有効な MIME タイプは、次のとおりです。

MIME タイプ	値
テキスト	"text/plain"
HTML	"text/html"
URL	"text/uri-list"
ビットマップ	"image/x-vnd.adobe.air.bitmap"
ファイルリスト	"application/x-vnd.adobe.air.file-list"

重要：アプリケーションサンドボックス内のコンテンツからのみ、クリップボード上にあるファイルデータにアクセスできます。非アプリケーションコンテンツからクリップボードのファイルオブジェクトにアクセスしようとすると、セキュリティエラーがスローされます。

クリップボードの使用について詳しくは、192 ページの「コピー＆ペースト」および「JavaScript からのペーストボードの使用（Apple デベロッパーセンター）」を参照してください。

ドラッグ＆ドロップ

Adobe AIR 1.0 およびそれ以降

HTML への、また HTML からのドラッグ＆ドロップ操作は、dragstart、drag、dragend、dragenter、dragover、dragleave、drop の各 DOM イベントを発生させます。これらのイベントで渡されるイベントオブジェクトでは、dataTransfer プロパティを使用してドラッグしたデータにアクセスできます。dataTransfer プロパティは、クリップボードイベントに関連付けられている clipboardData オブジェクトと同じメソッドを備えたオブジェクトを参照します。例えば、次の関数を使用すると、drop イベントからテキスト形式のデータを取得できます。

```
function onDrop(dragEvent){
    return dragEvent.dataTransfer.getData("text/plain",
            air.ClipboardTransferMode.ORIGINAL_ONLY);
}
```

dataTransfer オブジェクトには、次の重要なメンバーがあります。

メンバー	説明
clearData(mimeType)	データを消去します。mimeType パラメーターを、消去するデータ表現の MIME タイプに設定します。
getData(mimeType)	ドラッグするデータを取得します。このメソッドは、drop イベントのハンドラーでのみ呼び出すことができます。mimeType パラメーターを、取得するデータの MIME タイプに設定します。
setData(mimeType, data)	ドラッグするデータを設定します。mimeType パラメーターをデータの MIME タイプに設定します。
types	dataTransfer オブジェクトで現在使用できるすべてのデータ表現の MIME タイプを含む、ストリングの配列です。
effectsAllowed	ドラッグするデータのコピー、移動、リンクまたはそれらを組み合わせた操作を行えるかどうかを指定します。dragstart イベントのハンドラーで effectsAllowed プロパティを設定します。
dropEffect	可能なドロップ操作のうち、ドラッグターゲットでサポートするものを指定します。dragEnter イベントのハンドラーで dropEffect プロパティを設定します。ドラッグ時、ユーザーがマウスを放すと発生する操作を示すよう、カーソルが変化します。dropEffect を指定していない場合は、effectsAllowed プロパティの操作が選択されます。コピー操作は移動操作より優先度が高く、移動操作はリンク操作より優先度が高いです。ユーザーは、キーボードを使用して、デフォルトの優先度を変更できます。

AIR アプリケーションにドラッグ& ドロップのサポートを追加する方法について詳しくは、175 ページの「AIR でのドラッグ& ドロップ」および「JavaScript からのドラッグ& ドロップの使用（Apple デベロッパーセンター）」を参照してください。

innerHTML プロパティと outerHTML プロパティ

Adobe AIR 1.0 およびそれ以降

AIR では、アプリケーションサンドボックス内で実行されるコンテンツの innerHTML プロパティと outerHTML プロパティの使用に対して、セキュリティ制限が適用されます。ページの load イベントの前、また load イベントハンドラーの実行中は、innerHTML プロパティと outerHTML プロパティの使用は制限されません。ただし、ページが読み込まれると、innerHTML プロパティまたは outerHTML プロパティを使用できるのは、ドキュメントへの静的コンテンツの追加に限定されます。実行可能コードになる、innerHTML または outerHTML に割り当てられたストリング内のステートメントは、すべて無視されます。例えば、イベントのコールバック属性をエレメント定義に含めると、イベントリスナーは追加されません。同様に、埋め込みの <script> タグは評価されません。詳しくは、72 ページの「Adobe AIR の HTML セキュリティ」を参照してください。

Document.write() メソッドと Document.writeln() メソッド

Adobe AIR 1.0 およびそれ以降

ページの load イベントの前は、write() メソッドと writeln() メソッドの使用はアプリケーションサンドボックスで制限されません。ただし、ページが読み込まれると、これらのメソッドのいずれかを呼び出しても、ページがクリアされることも新しいページが作成されることもありません。非アプリケーションサンドボックスでは、ほとんどの Web ブラウザーの場合と同様に、ページの読み込み完了後に document.write() または writeln() を呼び出すと、現在のページがクリアされ、新しい空白のページが開かれます。

Document.designMode プロパティ

Adobe AIR 1.0 およびそれ以降

ドキュメント内のすべてのエレメントを編集可能にするには、document.designMode プロパティを値 on に設定します。ビルトインエディターのサポートに含まれるのは、テキストの編集、コピー、ペーストおよびドラッグ＆ドロップです。designMode を on に設定することは、body エレメントの contentEditable プロパティを true に設定することと同じです。編集可能にするドキュメントのセクションを定義するため、HTML エレメントのほとんどで contentEditable プロパティを使用できます。詳しくは、15 ページの「HTML の contentEditable 属性」を参照してください。

unload イベント（body オブジェクトおよび frameset オブジェクトの場合）

Adobe AIR 1.0 およびそれ以降

ウィンドウ（アプリケーションのメインウィンドウを含む）のトップレベルの frameset タグまたは body タグでは、閉じられるウィンドウ（またはアプリケーション）への応答に unload イベントを使用しないでください。代わりに、NativeApplication オブジェクトの exiting イベントを使用します（アプリケーションが終了するときを検出するために使用）。または、NativeWindow オブジェクトの closing イベントを使用します（ウィンドウが閉じるときを検出するために使用）。例えば、次の JavaScript コードでは、ユーザーがアプリケーションを閉じるときにメッセージ（「Goodbye.」）を表示します。

```
var app = air.NativeApplication.nativeApplication;
app.addEventListener(air.Event.EXITING, closeHandler);
function closeHandler(event)
{
    alert("Goodbye.");
}
```

ただし、スクリプトでは、フレーム、インラインフレームまたはトップレベルウィンドウのコンテンツのナビゲーションによって発生した unload イベントに正常に応答できます。

注意：これらの制限は、Adobe AIR の将来のバージョンで取り除かれる可能性があります。

JavaScript の Window オブジェクト

Adobe AIR 1.0 およびそれ以降

Window オブジェクトは、JavaScript 実行コンテキストではグローバルオブジェクトのままです。アプリケーションサンドボックスでは、重要なホストオブジェクトと同様、AIR のビルトインクラスへのアクセスを提供するため、新しいプロパティが JavaScript の Window オブジェクトに追加されます。また、一部のメソッドとプロパティは、アプリケーションサンドボックス内であるかどうかによって動作が異なります。

Window.runtime プロパティ runtime プロパティを使用すると、アプリケーションサンドボックス内から、ビルトインのランタイムクラスをインスタンス化して使用できます。これらのクラスには、AIR および Flash Player の API が含まれます（ただし、Flex フレームワークなどは含まれません）。例えば、次のステートメントでは、AIR ファイルオブジェクトを作成します。

```
var preferencesFile = new window.runtime.flash.filesystem.File();
```

AIR SDK に付属する AIRAliases.js ファイルには、このような参照を短くすることができるエイリアス定義が含まれています。例えば、AIRAliases.js をページに読み込むと、次のステートメントで File オブジェクトを作成できます。

```
var preferencesFile = new air.File();
```

window.runtime プロパティは、アプリケーションサンドボックス内のコンテンツと、フレームまたはインラインフレームがあるページの親ドキュメントのためだけに定義されています。

29 ページの「AIRAliases.js ファイルの使用」を参照してください。

Window.nativeWindow プロパティ nativeWindow プロパティは、基になるネイティブウィンドウオブジェクトへの参照を提供します。このプロパティを使用すると、画面の位置、サイズ、可視性など、ウィンドウの関数とプロパティのスクリプトを作成することも、終了、サイズ変更、移動などのウィンドウイベントを処理することもできます。例えば、次のステートメントではウィンドウを閉じます。

```
window.nativeWindow.close();
```

注意：NativeWindow オブジェクトが提供するウィンドウ制御機能は、JavaScript の Window オブジェクトが提供する機能と重複します。そのような場合は、使いやすい方のメソッドを使用できます。

window.nativeWindow プロパティは、アプリケーションサンドボックス内のコンテンツと、フレームまたはインラインフレームがあるページの親ドキュメントのためだけに定義されています。

Window.htmlLoader プロパティ htmlLoader プロパティは、HTML コンテンツを含む AIR の HTMLLoader オブジェクトへの参照を提供します。このプロパティを使用すると、HTML 環境の外観と動作のスクリプトを作成できます。例えば、htmlLoader.paintsDefaultBackground プロパティを使用して、コントロールでデフォルトの白い背景を描画するかどうかを指定できます。

```
window.htmlLoader.paintsDefaultBackground = false;
```

注意：HTMLLoader オブジェクト自体に、格納している HTML コンテンツの JavaScript Window オブジェクトを参照する window プロパティがあります。このプロパティを使用すると、コンテナとなっている HTMLLoader への参照を通じて JavaScript 環境にアクセスできます。

window.htmlLoader プロパティは、アプリケーションサンドボックス内のコンテンツと、フレームまたはインラインフレームがあるページの親ドキュメントのためだけに定義されています。

Window.parentSandboxBridge プロパティと Window.childSandboxBridge プロパティ parentSandboxBridge プロパティと childSandboxBridge プロパティを使用すると、親と子のフレーム間のインターフェイスを定義できます。詳しくは、35 ページの「異なるセキュリティサンドボックス内のコンテンツのクロススクリプト」を参照してください。

Window.setTimeout() 関数と Window.setInterval() 関数 AIR では、アプリケーションサンドボックス内での setTimeout() 関数と setInterval() 関数の使用に対して、セキュリティ制限が適用されます。setTimeout() または setInterval() を呼び出すときに、ストリングとして実行されるコードを定義することはできません。関数参照を使用する必要があります。詳しくは、26 ページの「setTimeout() および setInterval()」を参照してください。

Window.open() 関数 非アプリケーションサンドボックス内で実行されるコードから呼び出す場合、open() メソッドは、ユーザー操作（マウスクリックやキー入力など）の結果として呼び出されたときにのみウィンドウを開きます。また、ウィンドウのタイトルには、アプリケーションタイトルが前に付けられます（リモートコンテンツによって開かれたウィンドウが、アプリケーションによって開かれたウィンドウを偽装しないようにするため）。詳しくは、77 ページの「JavaScript window.open() メソッドの呼び出しに関する制限」を参照してください。

air.NativeApplication オブジェクト

Adobe AIR 1.0 およびそれ以降

NativeApplication オブジェクトは、アプリケーションの状態に関する情報を提供し、いくつかの重要なアプリケーションレベルのイベントを送出し、アプリケーション動作の制御に役立つ関数を提供します。NativeApplication オブジェクトの単一インスタンスは、自動的に作成され、クラスで定義されている NativeApplication.nativeApplication プロパティを使用してアクセスできます。

このオブジェクトに JavaScript コードからアクセスするには、次のステートメントを使用できます。

```
var app = window.runtime.flash.desktop.NativeApplication.nativeApplication;
```

または、AIRAliases.js スクリプトが読み込まれている場合は、次の短い形式を使用できます。

```
var app = air.NativeApplication.nativeApplication;
```

NativeApplication オブジェクトには、アプリケーションサンドボックス内からのみアクセスできます。
NativeApplication オブジェクトについて詳しくは、297 ページの「AIR ランタイムとオペレーティングシステムに関する情報の操作」を参照してください。

JavaScript URL スキーム
Adobe AIR 1.0 およびそれ以降

JavaScript URL スキーム（href="javascript:alert('Test')" など）で定義されたコードの実行は、アプリケーションサンドボックス内でブロックされます。エラーはスローされません。

AIR での HTML
Adobe AIR 1.0 およびそれ以降

AIR および WebKit では、次に示す、非標準の HTML エレメントと属性を定義します。

12 ページの「HTML の frame エレメントと iframe エレメント」

14 ページの「HTML エレメントのイベントハンドラー」

HTML の frame エレメントと iframe エレメント
Adobe AIR 1.0 およびそれ以降

AIR では、アプリケーションサンドボックス内のコンテンツの frame エレメントと iframe エレメントに新しい属性を追加します。

sandboxRoot 属性 sandboxRoot 属性は、フレームの src 属性で指定されているファイルの元の場所の代わりとなる非アプリケーションドメインを指定します。ファイルは、指定したドメインに対応する非アプリケーションサンドボックスに読み込まれます。ファイル内のコンテンツと、指定したドメインから読み込まれたコンテンツでは、互いにクロススクリプトできます。

重要：sandboxRoot の値をドメインのベース **URL** に設定すると、そのドメインからのコンテンツに対するすべての要求は、リモートサーバーではなく、アプリケーションディレクトリから読み込まれます（その要求が、ページナビゲーション、**XMLHttpRequest**、コンテンツを読み込むその他の手段のいずれによって発生したかは関係ありません）。

documentRoot 属性 documentRoot 属性は、sandboxRoot で指定されている場所にあるファイルに解決される URL の読み込み元となるローカルディレクトリを指定します。

フレームの src 属性またはフレームに読み込まれたコンテンツのいずれかで、URL を解決するとき、sandboxRoot で指定されている値に一致する URL の部分は、documentRoot で指定されている値によって置き換えられます。したがって、次のフレームタグでは、後述の動作になります。

```
<iframe src="http://www.example.com/air/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="http://www.example.com/air/"/>
```

child.html は、アプリケーションのインストールフォルダーの sandbox サブディレクトリから読み込まれます。child.html の相対 URL は、sandbox ディレクトリに基づいて解決されます。www.example.com/air のリモートサーバー上のいずれのファイルにもフレームでアクセスできません。これは、AIR が app:/sandbox/ ディレクトリからファイルを読み込もうとするためです。

allowCrossDomainXHR 属性 フレーム内のコンテンツで任意のリモートドメインに対する XMLHttpRequest を生成することを許可するには、フレームの開始タグに allowCrossDomainXHR="allowCrossDomainXHR" を含めます。デフォルトでは、非アプリケーションコンテンツは、元の個々のドメインに対してのみこの要求を行うことができます。クロスドメイン

XHR を許可する場合、重大なセキュリティ上の影響を伴います。ページのコードで任意のドメインとデータを交換できるようになります。悪意のあるコンテンツが何らかの形でページに挿入されている場合、現在のサンドボックス内のコードからアクセスできるあらゆるデータが、漏洩してしまう可能性があります。クロスドメイン XHR は、自分で作成して制御するページに対してのみ有効にし、クロスドメインのデータ読み込みが本当に必要な場合にのみ有効にしてください。また、コードの挿入や他の形での攻撃を避けるため、ページで読み込まれるすべての外部データを慎重に検証してください。

重要：allowCrossDomainXHR 属性を frame エレメントまたは iframe エレメントに含めると、クロスドメイン XHR が有効になります（割り当てられた値が「0」であるか、「f」または「n」で始まる場合を除く）。例えば、allowCrossDomainXHR を "deny" に設定した場合は、クロスドメイン **XHR** は有効のままになります。クロスドメイン要求を有効にしない場合は、エレメントの宣言からこの属性全体を除外してください。

ondominitialize 属性 [ondominitialize ぞくせい] フレームの dominitialize イベントのイベントハンドラーを指定します。このイベントは、AIR 固有のイベントで、フレームのウィンドウオブジェクトおよびドキュメントオブジェクトが作成されたときに発生します。ただし、スクリプトが解析されたり、ドキュメントエレメントが作成されたりする前に発生します。

フレームは、dominitialize ハンドラーによって子ドキュメントに追加されたオブジェクト、変数および関数を子ページのスクリプトで参照できる読み込み順序で、dominitialize イベントを十分早期に送出します。子ドキュメントのオブジェクトについて、直接追加したり、アクセスしたりするには、親ページが子と同じサンドボックスに含まれている必要があります。ただし、アプリケーションサンドボックス内の親で、サンドボックスブリッジを確立して、非アプリケーションサンドボックス内のコンテンツと通信できます。

次の例に、AIR での iframe タグの使用を示します。

child.html を、リモートサーバーの実際のドメインにマップせずに、リモートサンドボックスに配置します。

```
<iframe src="http://localhost/air/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="http://localhost/air/"/>
```

child.html をリモートサンドボックスに配置し、www.example.com への XMLHttpRequest のみ許可します。

```
<iframe src="http://www.example.com/air/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="http://www.example.com/air/"/>
```

child.html をリモートサンドボックスに配置し、任意のリモートドメインへの XMLHttpRequest を許可します。

```
<iframe src="http://www.example.com/air/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="http://www.example.com/air/"
            allowCrossDomainXHR="allowCrossDomainXHR"/>
```

child.html を local-with-file-system サンドボックスに配置します。

```
<iframe     src="file:///templates/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="app-storage:/templates/"/>
```

child.html をリモートサンドボックスに配置し、dominitialize イベントを使用してサンドボックスブリッジを確立します。

```
<html>
<head>
<script>
var bridgeInterface = {};
bridgeInterface.testProperty = "Bridge engaged";
function engageBridge(){
    document.getElementById("sandbox").parentSandboxBridge = bridgeInterface;
}
</script>
</head>
<body>
<iframe id="sandbox"
            src="http://www.example.com/air/child.html"
            documentRoot="app:/"
            sandboxRoot="http://www.example.com/air/"
            ondominitialize="engageBridge()"/>
</body>
</html>
```

次の child.html ドキュメントは、子コンテンツから親サンドボックスブリッジにアクセスする方法を示しています。

```
<html>
    <head>
        <script>
            document.write(window.parentSandboxBridge.testProperty);
        </script>
    </head>
    <body></body>
</html>
```

詳しくは、35 ページの「異なるセキュリティサンドボックス内のコンテンツのクロススクリプト」および 72 ページの「Adobe AIR の HTML セキュリティ」を参照してください。

HTML エレメントのイベントハンドラー

Adobe AIR 1.0 およびそれ以降

AIR および Webkit の DOM オブジェクトは、標準の DOM イベントモデルにはないイベントを送出します。次の表に、これらのイベントのハンドラーを指定するために使用できる、関連するイベント属性を示します。

コールバック属性名	説明
oncontextmenu	選択されたテキストでの右クリックまたはコマンドクリックなどによってコンテキストメニューが呼び出されるときに呼び出されます。
oncopy	エレメントでの選択内容がコピーされるときに呼び出されます。
oncut	エレメントでの選択内容がカットされるときに呼び出されます。
ondominitialize	フレームまたはインラインフレームに読み込まれたドキュメントの DOM が作成されるときに呼び出されます。ただし、DOM エレメントが作成されたり、スクリプトが解析されたりする前に呼び出されます。
ondrag	エレメントがドラッグされるときに呼び出されます。
ondragend	ドラッグが放されるときに呼び出されます。
ondragenter	ドラッグ操作がエレメントの境界に入るときに呼び出されます。
ondragleave	ドラッグ操作がエレメントの境界から出るときに呼び出されます。
ondragover	ドラッグ操作がエレメントの境界内にある間、連続して呼び出されます。

コールバック属性名	説明
ondragstart	ドラッグ操作が開始されるときに呼び出されます。
ondrop	ドラッグ操作がエレメント上で放されるときに呼び出されます。
onerror	エレメントの読み込み中にエラーが発生したときに呼び出されます。
oninput	テキストがフォームエレメントに入力されるときに呼び出されます。
onpaste	アイテムがエレメントにペーストされるときに呼び出されます。
onscroll	スクロールできるエレメントのコンテンツがスクロールされるときに呼び出されます。
onselectstart	選択が開始されるときに呼び出されます。

HTML の contentEditable 属性

Adobe AIR 1.0 およびそれ以降

contentEditable 属性を任意の HTML エレメントに追加して、そのエレメントのコンテンツをユーザーが編集できるようにすることができます。例えば、次の例の HTML コードでは、最初の p エレメントを除き、ドキュメント全体を編集可能として設定します。

```
<html>
<head/>
<body contentEditable="true">
    <h1>de Finibus Bonorum et Malorum</h1>
    <p contentEditable="false">Sed ut perspiciatis unde omnis iste natus error.</p>
    <p>At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis.</p>
</body>
</html>
```

注意：document.designMode プロパティを on に設定すると、個々のエレメントに対する contentEditable の設定に関わらず、ドキュメント内のすべてのエレメントが編集可能になります。ただし、designMode を off に設定しても、contentEditable が true になっているエレメントの編集は無効になりません。詳しくは、10 ページの「Document.designMode プロパティ」を参照してください。

data: URL

Adobe AIR 2 以降

AIR は、次のエレメントの data: URL をサポートしています。

- img
- input type="image"
- CSS ルールで許可されているイメージ（background-image など）

data: URL を使用すると、CSS または HTML ドキュメントにバイナリイメージデータを直接 base64 エンコード文字列として挿入できます。次の例では、data: URL を繰り返す背景として使用しています。

```
<html>
<head>
<style>
body {
background-
image:url('
ZSBJbWFnZVJlYWR5ccllPAAAAAZQTFRF%2F6cA%2F%2F%2F%2Fgxp3lwAAAAJ0Uk5T%2FwDltzBKAAABF0lEQVR42uzZQQ7CMAxE0e%2F
7X5oNCyRocWzPiJbMBZ6qpIljE%2BnwklgKG7kwUjc2IkIaxkY0CPdEsCCasws6ShXBgmBBmEagpXQQLAgWBAuSY2gaKaWPYEGwIEwg0F
RmECwIFoQeQjJlhJWUEFazjFDJCkI5WYRWMgjtfEGYyQnCXD4jTCdm1zmngFpBFznwVNi5RPSbwbWnpYr%2BBHi%2FtCTfgPLEPL7jBct
AKBRptXJ8M%2BprIuZKu%2BUKcg4YK1PLz7kx4bSqHyPaT4d%2B28OCJJiRBo4FCQsSA0bziT3XubMgYUG6fc5fatmGBQkL0hoJ1IaZMi
QsSFiQ8vRscTjlQOI2iHZwtpHuf%2BJAYiOiJSkj8Z%2FIQ4ABANvXGLd3%2BZMrAAAAAElFTkSuQmCC');
background-repeat:repeat;
}
</style>
</head>
<body>
</body>
</html>
```

data: URL を使用するときは、空白の有無に意味があることに留意してください。例えば、データ文字列は改行されていない単一行として入力する必要があります。改行すると、改行がデータの一部として処理され、イメージがデコードされません。

AIR での CSS

Adobe AIR 1.0 およびそれ以降

WebKit では、様々な拡張 CSS プロパティをサポートしています。これらの拡張機能の多くでは、接頭辞 -webkit が使用されます。これらの拡張機能の一部は実験的なものであり、将来のバージョンの WebKit からは除外される可能性があります。CSS に対する WebKit のサポートと拡張機能について詳しくは、「Safari CSS リファレンス」を参照してください。

AIR でサポートされていない WebKit の機能

Adobe AIR 1.0 およびそれ以降

AIR は、WebKit または Safari 4 で利用できる次の機能をサポートしていません。

- window.postMessage 経由のクロスドメインメッセージング（AIR は独自のクロスドメイン通信 API を提供）。
- CSS 変数
- Web Open Font Format（WOFF）および SVG フォント
- HTML ビデオタグおよびオーディオタグ
- メディアデバイスキュー
- オフラインアプリケーションキャッシュ
- プリント（AIR は独自の PrintJob API を提供）
- スペルチェッカーおよび文法チェッカー
- SVG
- WAI-ARIA
- WebSocket（AIR は独自のソケット API を提供）
- Web ワーカー
- WebKit SQL API（AIR は独自の API を提供）

- WebKit ジオロケーション API（API はサポートされているデバイスで独自のジオロケーション API を提供）
- WebKit マルチファイルアップロード API
- WebKit タッチイベント（AIR は独自のタッチイベントを提供）
- Wireless Markup Language（WML）

次のリストは、AIR がサポートしていない特定の JavaScript API、HTML エレメント、CSS プロパティと値を示しています。

サポートされない JavaScript Window オブジェクトメンバー：

- applicationCache()
- console
- openDatabase()
- postMessage()
- document.print()

サポートされない HTML タグ：

- audio
- video

サポートされない HTML 属性：

- aria-*
- draggable
- formnovalidate
- list
- novalidate
- onbeforeload
- onhashchange
- onorientationchange
- onpagehide
- onpageshow
- onpopstate
- ontouchstart
- ontouchmove
- ontouchend
- ontouchcancel
- onwebkitbeginfullscreen
- onwebkitendfullscreen
- pattern
- required
- sandbox

サポートされない JavaScript イベント：

- beforeload
- hashchange
- orientationchange
- pagehide
- pageshow
- popstate
- touchstart
- touchmove
- touchend
- touchcancel
- webkitbeginfullscreen
- webkitendfullscreen

サポートされない CSS プロパティ：

- background-clip
- background-origin（-webkit-background-origin を使用）
- background-repeat-x
- background-repeat-y
- background-size（-webkit-background-size を使用）
- border-bottom-left-radius
- border-bottom-right-radius
- border-radius
- border-top-left-radius
- border-top-right-radius
- text-rendering
- -webkit-animation-play-state
- -webkit-background-clip
- -webkit-color-correction
- -webkit-font-smoothing

サポートされない CSS 値：

- 外観プロパティの値：
  - media-volume-slider-container
  - media-volume-slider
  - media-volume-sliderthumb
  - outer-spin-button

- border-box（background-clip および background-origin）
- contain（background-size）
- content-box（background-clip および background-origin）
- cover（background-size）
- リストプロパティの値：
  - afar
  - amharic
  - amharic-abegede
  - cjk-earthly-branch
  - cjk-heavenly-stem
  - ethiopic
  - ethiopic-abegede
  - ethiopic-abegede-am-et
  - ethiopic-abegede-gez
  - ethiopic-abegede-ti-er
  - ethiopic-abegede-ti-et
  - ethiopic-halehame-aa-er
  - ethiopic-halehame-aa-et
  - ethiopic-halehame-am-et
  - ethiopic-halehame-gez
  - ethiopic-halehame-om-et
  - ethiopic-halehame-sid-et
  - ethiopic-halehame-so-et
  - ethiopic-halehame-ti-er
  - ethiopic-halehame-ti-et
  - ethiopic-halehame-tig
  - hangul
  - hangul-consonant
  - lower-norwegian
  - oromo
  - sidama
  - somali
  - tigre
  - tigrinya-er
  - tigrinya-er-abegede

  - tigrinya-et
  - tigrinya-et-abegede
  - upper-greek
  - upper-norwegian
- -wap-marquee（表示プロパティ）

第 2 章：AIR での HTML および JavaScript のプログラミング

Adobe AIR 1.0 およびそれ以降

HTML および JavaScript を使用した Adobe® AIR® アプリケーション開発に特有のプログラミングトピックがいくつかあります。以降の情報は、HTML ベースの AIR アプリケーションをプログラミングする場合にも、HTMLLoader クラス（または mx:HTML Flex™ コンポーネント）を使用して HTML および JavaScript を実行する SWF ベースの AIR アプリケーションをプログラミングする場合にも重要です。

HTML ベースの AIR アプリケーションの作成

Adobe AIR 1.0 およびそれ以降

AIR アプリケーションの開発プロセスは、HTML ベースの Web アプリケーションの開発プロセスにかなり類似しています。アプリケーション構造はページベースのままで、HTML がドキュメントの構造を規定し、JavaScript がアプリケーションロジックを規定します。さらに、AIR アプリケーションはアプリケーション記述ファイルを必要とします。アプリケーション記述ファイルは、アプリケーションに関するメタデータを含み、アプリケーションのルートファイルを指定します。

Adobe® Dreamweaver® を使用する場合、Dreamweaver ユーザーインターフェイスから直接 AIR アプリケーションをテストおよびパッケージ化できます。AIR SDK を使用する場合、コマンドラインの ADL ユーティリティで AIR アプリケーションをテストできます。ADL はアプリケーション記述子を読み込み、アプリケーションを起動します。コマンドラインの ADT ユーティリティを使用して、アプリケーションを AIR インストールファイルにパッケージ化できます。

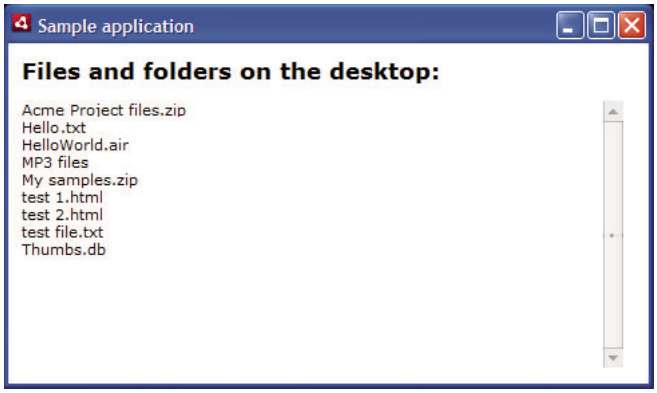
AIR アプリケーションを作成する基本の手順は次のとおりです。

1. アプリケーション記述ファイルを作成します。content エレメントでアプリケーションのルートページを指定します。ルートページはアプリケーションが起動されるときに自動的に読み込まれます
2. アプリケーションのページおよびコードを作成します。
3. ADL ユーティリティまたは Dreamweaver を使用してアプリケーションをテストします。
4. ADT ユーティリティまたは Dreamweaver でアプリケーションを AIR インストールファイルにパッケージ化します。

サンプルアプリケーションとセキュリティ上の影響

Adobe AIR 1.0 およびそれ以降

次の HTML コードでは、ファイルシステム API を使用して、ユーザーのデスクトップディレクトリ内のファイルとディレクトリを一覧表示します。



アプリケーションの HTML コードを次に示します。

```
<html>
    <head>
        <title>Sample application</title>
        <script type="text/javascript" src="AIRAliases.js"></script>
        <script>
            function getDesktopFileList()
            {
                var log = document.getElementById("log");
                var files = air.File.desktopDirectory.getDirectoryListing();
                for (i = 0; i < files.length; i++)
                {
                    log.innerHTML += files[i].name + "<br/>";
                }
            }
        </script>
    </head>
    <body onload="getDesktopFileList();" style="padding: 10px">
        <h2>Files and folders on the desktop:</h2>
        <div id="log" style="width: 450px; height: 200px; overflow-y: scroll;" />
    </body>
</html>
```

また、アプリケーション記述ファイルを設定し、AIR Debug Launcher（ADL）アプリケーションを使用してアプリケーションをテストする必要もあります

サンプルコードの大部分を Web ブラウザーで使用することができます。ただし、コードの一部はランタイム固有です。

getDesktopFileList() メソッドでは、ランタイム API で定義されている File クラスを使用しています。アプリケーションの最初の script タグで、AIR API へのアクセスを容易にする AIRAliases.js ファイル（AIR SDK が提供）が読み込まれます（例えば、サンプルコードでは、シンタックス air.File で AIR File クラスにアクセスしています）。詳しくは、29 ページの「AIRAliases.js ファイルの使用」を参照してください。

File.desktopDirectory プロパティは、File オブジェクト（ランタイムによって定義されるオブジェクトの型）です。File オブジェクトは、ユーザーのコンピューター上のファイルまたはディレクトリへの参照です。File.desktopDirectory プロパティは、ユーザーのデスクトップディレクトリへの参照です。getDirectoryListing() メソッドは、File オブジェクトに対して定義され、File オブジェクトの配列を返します。File.desktopDirectory.getDirectoryListing() メソッドは、ユーザーのデスクトップ上のファイルとディレクトリを表す File オブジェクトの配列を返します。

各 File オブジェクトには、ファイル名のストリングを持つ name プロパティがあります。getDesktopFileList() メソッドの for ループは、ユーザーのデスクトップのディレクトリ上のファイルとディレクトリの処理を繰り返し行い、アプリケーションの div オブジェクトの innerHTML プロパティにファイル名とディレクトリ名を付加していきます。

AIR アプリケーションでの HTML 使用時の重要なセキュリティ規則

Adobe AIR 1.0 およびそれ以降

AIR アプリケーションと共にインストールするファイルから、AIR API にアクセスできます。セキュリティ上の理由で、他のソースのコンテンツからはアクセスできません。例えば、この制限により、リモートドメイン（http://example.com など）のコンテンツからユーザーのデスクトップディレクトリを読み込むこと（またはさらに悪い状況）はできません。

eval() 関数（および関連する API）を呼び出すことによって攻撃される可能性があるセキュリティの抜け穴があるので、アプリケーションと共にインストールされるコンテンツは、これらのメソッドの使用がデフォルトで制限されています。しかし、一部の Ajax フレームワークは、eval() 関数および関連する API の呼び出しを使用します。

AIR アプリケーションで動作するコンテンツを適切に構成するには、異なるソースのコンテンツに対するセキュリティ制限の規則を考慮する必要があります。異なるソースのコンテンツは、サンドボックスと呼ばれる個別のセキュリティ区分に配置されます（セキュリティサンドボックスを参照）。デフォルトで、アプリケーションと共にインストールされるコンテンツは、アプリケーションサンドボックスというサンドボックスにインストールされ、AIR API へのアクセス権が付与されます。アプリケーションサンドボックスは、信頼できないコードの実行を防止するよう設計された制限があり、一般に最も安全性の高いサンドボックスです。

ランタイムでは、アプリケーションと共にインストールされたコンテンツをアプリケーションサンドボックス以外のサンドボックスに読み込むことができます。非アプリケーションサンドボックス内のコンテンツは、一般的な Web ブラウザーのコンテンツと同様のセキュリティ環境で動作します。例えば、非アプリケーションサンドボックス内のコードは、eval() メソッドおよび関連するメソッドを使用できます（ただし、同時に AIR API にアクセスすることはできません）。ランタイムには、（AIR API を例えば非アプリケーションコンテンツに公開せずに）異なるサンドボックス内のコンテンツが安全に通信できる方法が備えられています。詳しくは、35 ページの「異なるセキュリティサンドボックス内のコンテンツのクロススクリプト」を参照してください。

サンドボックスで使用が制限されているコードを呼び出すと、セキュリティ上の理由から、ランタイムは JavaScript エラー「Adobe AIR security violation for JavaScript code in the application security sandbox.」を送出します。

このエラーを回避するには、次節、23 ページの「セキュリティ関連の JavaScript エラーの回避」で説明するコーディングの手法を参照してください。

詳しくは、72 ページの「Adobe AIR の HTML セキュリティ」を参照してください。

セキュリティ関連の JavaScript エラーの回避

Adobe AIR 1.0 およびそれ以降

セキュリティ制限により、サンドボックスで使用が制限されているコードを呼び出すと、ランタイムは JavaScript エラー「Adobe AIR security violation for JavaScript code in the application security sandbox.」を送出します。このエラーを回避するには、ここで説明するコーディングの手法を参考にしてください。

セキュリティ関連の JavaScript エラーの原因

Adobe AIR 1.0 およびそれ以降

アプリケーションサンドボックスで実行するコードでは、ドキュメントの load イベントが開始し、任意の load イベントハンドラーが終了すると、ストリングの評価および実行を伴うほとんどの操作が制限されます。潜在的に安全でないストリングを評価および実行する次のタイプの JavaScript ステートメントの使用を試みると、JavaScript エラーが発生します。

- eval() 関数
- setTimeout() および setInterval()
- Function コンストラクター

  さらに、次のタイプの JavaScript ステートメントは、安全でない JavaScript エラーを発生させずにエラーになります。

- javascript: URL
- innerHTML および outerHTML ステートメントの onevent 属性によって割り当てられるイベントコールバック
- アプリケーションインストールディレクトリの外部からの JavaScript ファイルの読み込み
- document.write() および document.writeln()
- load イベントの前または load イベントハンドラー実行中に同期する XMLHttpRequests
- 動的に作成されるスクリプトエレメント

  注意：制限されるケースの一部では、ストリングの評価は許可されます。詳しくは、75 ページの「異なるサンドボックス内のコンテンツに対するコードの制限」を参照してください。

  アドビ システムズ社では、アプリケーションセキュリティサンドボックスをサポートする Ajax フレームワークのリストを http://www.adobe.com/go/airappsandboxframeworks_jp で管理しています。

  以降の節では、アプリケーションサンドボックスで実行するコードについて、安全でない JavaScript エラーおよびメッセージを出さない障害を回避するようスクリプトを書き直す方法について説明します。

異なるサンドボックスへのアプリケーションコンテンツのマッピング

Adobe AIR 1.0 およびそれ以降

ほとんどの場合、アプリケーションを書き直したり再構築したりすることで、セキュリティ関連の JavaScript エラーを回避できます。しかし、書き直しまたは再構築時ができない場合には、36 ページの「アプリケーションコンテンツの非アプリケーションサンドボックスへの読み込み」で説明する方法で、アプリケーションコンテンツを別のサンドボックスに読み込めます。該当のコンテンツからも AIR API にアクセスする必要がある場合、37 ページの「サンドボックスブリッジインターフェイスの設定」で説明するようにして、サンドボックスブリッジを作成できます。

eval() 関数

Flash Player 9 以降、Adobe AIR 1.0 以降

アプリケーションサンドボックスでは、eval() 関数は、ページの load イベントの前または load イベントハンドラー実行中にのみ使用できます。ページが読み込まれた後、eval() を呼び出してもコードは実行されません。しかし、次の場合は、コードを書き直すことで eval() の使用を避けることができます。

プロパティのオブジェクトへの割り当て

Adobe AIR 1.0 およびそれ以降

ストリングを解析してプロパティアクセサを構築する次のようなコードにしないでください。

```
eval("obj." + propName + " = " + val);
```

次のように角括弧表記でプロパティにアクセスします。

```
obj[propName] = val;
```

コンテキストで使用可能な変数を持つ関数の作成

Adobe AIR 1.0 およびそれ以降

次のようなステートメントは、

```
function compile(var1, var2){
    eval("var fn = function(){ this."+var1+"(var2) }");
    return fn;
}
```

次のように置き換えます。

```
function compile(var1, var2){
    var self = this;
    return function(){ self[var1](var2) };
}
```

クラス名をストリングのパラメーターとして使用するオブジェクトの作成

Adobe AIR 1.0 およびそれ以降

次のコードで定義する仮定の JavaScript クラスを考えます。

```
var CustomClass =
    {
        Utils:
        {
            Parser: function(){ alert('constructor') }
        },
        Data:
        {

        }
    };
var constructorClassName = "CustomClass.Utils.Parser";
```

インスタンスを最も簡単に作成するには、次のように eval() を使用します。

```
var myObj;
eval('myObj=new ' + constructorClassName +'()')
```

しかし、クラス名の各コンポーネントを解析し、角括弧表記を使用して新規のオブジェクトを作成すれば、eval() の呼び出しを避けることができます。

```
function getter(str)
{
    var obj = window;
    var names = str.split('.');
    for(var i=0;i<names.length;i++){
        if(typeof obj[names[i]]=='undefined'){
            var undefstring = names[0];
            for(var j=1;j<=i;j++)
                undefstring+="."+names[j];
            throw new Error(undefstring+" is undefined");
        }
        obj = obj[names[i]];
    }
    return obj;
}
```

インスタンスを作成するには、次のようにします。

```
try{
    var Parser = getter(constructorClassName);
    var a = new Parser();
    }catch(e){
        alert(e);
}
```

setTimeout() および setInterval()

Adobe AIR 1.0 およびそれ以降

関数参照またはオブジェクトと共にハンドラー関数として渡されるストリングを置き換えます。次に例を示します。

```
setTimeout("alert('Timeout')", 100);
```

次のように置き換えます。

```
setTimeout(function(){alert('Timeout')}, 100);
```

または、関数に必要な this オブジェクトが呼び出し元によって設定される場合に、次のようなステートメントを

```
this.appTimer = setInterval("obj.customFunction();", 100);
```

次のステートメントに置き換えます。

```
var _self = this;
this.appTimer = setInterval(function(){obj.customFunction.apply(_self);}, 100);
```

Function コンストラクター

Adobe AIR 1.0 およびそれ以降

new Function(param, body) の呼び出しは、インライン関数宣言に置き換えるか、ページの load イベントの処理が完了する前にのみ使用します。

javascript: URL

Adobe AIR 1.0 およびそれ以降

javascript: URL スキームを使用してリンクに定義されるコードは、アプリケーションサンドボックス内では無視されます。安全でない JavaScript エラーは生成されません。javascript: URL を使用するリンクを置き換えることができます。次に例を示します。

```
<a href="javascript:code()">Click Me</a>
```

次のように置き換えます。

```
<a href="#" onclick="code()">Click Me</a>
```

innerHTML および outerHTML ステートメントの onevent 属性によって割り当てられるイベントコールバック

Adobe AIR 1.0 およびそれ以降

innerHTML または outerHTML を使用してドキュメントの DOM にエレメントを追加するとき、onclick または onmouseover などのステートメント内で割り当てられたイベントコールバックは無視されます。セキュリティエラーは生成されません。代わりに、id 属性を新しいエレメントに割り当て、addEventListener() メソッドを使用してイベントハンドラーコールバック関数を設定できます。

例えば、次のようなドキュメントのターゲットエレメントがあるとします。

```
<div id="container"></div>
```

次のようなステートメントを置き換えます。

```
document.getElementById('container').innerHTML =
    '<a href="#" onclick="code()">Click Me.</a>';
```

次のように置き換えます。

```
document.getElementById('container').innerHTML = '<a href="#" id="smith">Click Me.</a>';
document.getElementById('smith').addEventListener("click", function() { code(); });
```

アプリケーションインストールディレクトリの外部からの JavaScript ファイルの読み込み

Adobe AIR 1.0 およびそれ以降

アプリケーションサンドボックス外部からのスクリプトファイルの読み込みは許可されていません。セキュリティエラーは生成されません。アプリケーションサンドボックス内で実行されるすべてのスクリプトファイルがアプリケーションディレクトリにインストールされている必要があります。ページ内で外部スクリプトを使用するには、ページを別のサンドボックスにマップする必要があります。36 ページの「アプリケーションコンテンツの非アプリケーションサンドボックスへの読み込み」を参照してください。

document.write() および document.writeln()

Adobe AIR 1.0 およびそれ以降

document.write() または document.writeln() の呼び出しは、ページの load イベントの処理が完了した後は無視されます。セキュリティエラーは生成されません。代わりに、新しいファイルを読み込むか、DOM 操作のテクニックを使用してドキュメントの body を置き換えることができます。

load イベントの前または load イベントハンドラー実行中に同期する XMLHttpRequests

Adobe AIR 1.0 およびそれ以降

ページの load イベントまたは load イベントハンドラー実行中に開始された同期の XMLHttpRequests はコンテンツを返しません。非同期の XMLHttpRequests を開始することはできますが、load イベントの後まで制御が戻りません。load イベントの処理が完了した後は、同期の XMLHttpRequests は正常に動作します。

動的に作成されるスクリプトエレメント

Adobe AIR 1.0 およびそれ以降

innerHTML や document.createElement() メソッドなどで動的に作成されたスクリプトエレメントは無視されます。

JavaScript からの AIR API クラスのアクセス

Adobe AIR 1.0 およびそれ以降

HTML および JavaScript のコードでは、Webkit の標準的なエレメントおよび拡張されたエレメントに加えて、ランタイムが提供するホストクラスにアクセスできます。これらのクラスによって、AIR が備える次の拡張機能にアクセスできます。

- ファイルシステムへのアクセス
- ローカル SQL データベースの使用
- アプリケーションのコントロールおよびウィンドウメニュー
- ネットワーキング用ソケットへのアクセス
- ユーザー定義クラスおよびオブジェクトの使用
- サウンド機能

  例えば、AIR ファイル API には、flash.filesystem パッケージ内に File クラスが含まれます。JavaScript で次のようにして File オブジェクトを作成できます。

  ```
  var myFile = new window.runtime.flash.filesystem.File();
  ```

  runtime オブジェクトは、特殊な JavaScript オブジェクトで、アプリケーションサンドボックスの AIR で実行される HTML コンテンツから使用できます。runtime オブジェクトによって、JavaScript からランタイムクラスにアクセスできます。runtime オブジェクトの flash プロパティによって、flash パッケージにアクセスできます。また、runtime オブジェクトの flash.filesystem プロパティでは、flash.filesystem パッケージにアクセスできます（このパッケージは File クラスを含みます）。パッケージは、ActionScript で使用されるクラスを編成する手段です。

  注意：runtime プロパティは、フレームまたはインラインフレームに読み込まれるウィンドウに自動的には追加されません。しかし、子ドキュメントがアプリケーションサンドボックス内にある限り、子は親の runtime プロパティにアクセスできます。

  ランタイムクラスのパッケージ構造は、各クラスにアクセスする JavaScript コードストリングの長いストリングを開発者が入力する必要があるので（window.runtime.flash.desktop.NativeApplication など）、AIR SDK には、ランタイムクラスにもっと容易にアクセスできるようにする（例えば、簡単に air.NativeApplication と入力するなど）ための AIRAliases.js ファイルが含まれています。

AIR API クラスについては、このガイド全体を通じて随所で言及します。HTML 開発に役立つ Flash Player API の他のクラスについては、『HTML 開発者用 Adobe AIR API リファレンスガイド』を参照してください。ActionScript は、SWF（Flash Player）コンテンツで使用される言語です。ただし、JavaScript と ActionScript のシンタックスは似ています（この 2 つは、いずれも ECMAScript 言語を基にしています）。すべてのビルトインクラスが JavaScript（HTML コンテンツ内）と ActionScript（SWF コンテンツ内）の両方で使用できます。

注意：Dictionary クラス、XML クラスおよび XMLList クラスは、JavaScript コードでは使用できませんが、ActionScript では使用できます。

注意：詳しくは、330 ページの「ActionScript 3.0 のクラス、パッケージおよび名前空間」および 328 ページの「JavaScript 開発者のための ActionScript の基礎」を参照してください。

AIRAliases.js ファイルの使用

Adobe AIR 1.0 およびそれ以降

ランタイムクラスは、次のようなパッケージ構造に編成されます。

- window.runtime.flash.desktop.NativeApplication
- window.runtime.flash.desktop.ClipboardManager
- window.runtime.flash.filesystem.FileStream
- window.runtime.flash.data.SQLDatabase

AIR SDK には、少ない入力でランタイムクラスへのアクセスを可能にする「エイリアス」定義を指定する AIRAliases.js ファイルが含まれています。例えば、次のように入力するだけで上記のクラスにアクセスできます。

- air.NativeApplication
- air.Clipboard
- air.FileStream
- air.SQLDatabase

これは、AIRAliases.js ファイル内のクラスのほんの一部です。クラスおよびパッケージレベルの関数の完全な一覧については、『HTML 開発者用 Adobe AIR API リファレンスガイド』を参照してください。

AIRAliases.js ファイルには、一般に使用されるランタイムクラスに加えて、よく使用されるパッケージレベルの関数のエイリアスが含まれています。例えば、window.runtime.trace()、window.runtime.flash.net.navigateToURL() および window.runtime.flash.net.sendToURL() のエイリアスは、air.trace()、air.navigateToURL() および air.sendToURL() です。

AIRAliases.js ファイルを使用するには、HTML ページに次の script 参照が必要です。

```
<script src="AIRAliases.js"></script>
```

必要に応じて src 参照のパスを調整します。

重要：特に記載がない限り、このマニュアルの JavaScript サンプルコードは、HTML ページに AIRAliases.js ファイルを組み込むことを想定しています。

AIR での URL について

Adobe AIR 1.0 およびそれ以降

AIR で実行する HTML コンテンツでは、img タグ、frame タグ、iframe タグ、script タグの src 属性の定義で、link タグの href 属性で、または URL を指定できる任意の場所で、次の URL スキームのいずれかを使用できます。

URL スキーム	説明	例
file	ファイルシステムのルートを基準とした相対的なパス。	`file:///c:/AIR Test/test.txt`
app	インストールされたアプリケーションのルートディレクトリを基準とした相対的なパス。	`app:/images`
app-storage	アプリケーション記憶領域ディレクトリを基準とした相対的なパス。AIR では、インストールされたアプリケーションごとに、一意のアプリケーション記憶領域ディレクトリが定義されます。このディレクトリは、そのアプリケーション固有のデータを格納するための便利な場所として使用できます。	`app-storage:/settings/prefs.xml`
http	標準の HTTP リクエスト。	`http://www.adobe.com`
https	標準の HTTPS リクエスト。	`https://secure.example.com`

AIR の URL スキームの使用法について詳しくは、311 ページの「URI スキーム」を参照してください。

File クラス、Loader クラス、URLStream クラスおよび Sound クラスなど、AIR API の多くは、URL を含むストリングではなく、URLRequest オブジェクトを使用します。URLRequest オブジェクト自体は、同じ URL スキームで使用できるストリングで初期化されます。例えば、次のステートメントは、アドビ システムズ社の Web サイトの要求に使用できる URLRequest オブジェクトを作成します。

```
var urlReq = new air.URLRequest("http://www.adobe.com/");
```

URLRequest オブジェクトについて詳しくは、309 ページの「HTTP 通信」を参照してください。

HTML への SWF コンテンツの埋め込み

Adobe AIR 1.0 およびそれ以降

ブラウザーでの埋め込みと同じ方法で、AIR アプリケーション内で HTML コンテンツに SWF コンテンツを埋め込むことができます。SWF コンテンツを埋め込むには、object タグまたは embed タグを使用するか、両方のタグを使用します。

注意：Web 開発の一般的な方法では、object タグと embed タグの両方を使用して HTML ページ内に SWF コンテンツを表示します。この方法は AIR では利点がありません。AIR では、W3C 標準の object タグを単独で使用してコンテンツを表示できます。同時に、HTML コンテンツをブラウザーにも表示する場合は、必要に応じて object タグと embed タグの両方を使用することもできます。

HTML コンテンツと SWF コンテンツを表示する NativeWindow オブジェクトで透明度を有効にしている場合、コンテンツを埋め込むために使用されるウィンドウモード（wmode）が値 window に設定されているとき、AIR は SWF コンテンツを表示しません。透明ウィンドウの HTML ページで SWF コンテンツを表示するには、wmode パラメーターを opaque または transparent に設定します。wmode のデフォルト値は window なので、値を指定しなかった場合、コンテンツは表示されない可能性があります。

次の例では、HTML の object タグを使用して SWF ファイルを HTML コンテンツ内に表示する方法を示します。基になる NativeWindow オブジェクトが透明の場合でもコンテンツが表示されるように、wmode パラメーターが opaque に設定されています。SWF ファイルはアプリケーションディレクトリから読み込まれますが、AIR でサポートされている任意の URL スキームから読み込むことができます。どこから SWF ファイルを読み込むかによって、コンテンツの挿入先のセキュリティサンドボックスが決まります。

```
<object type="application/x-shockwave-flash" width="100%" height="100%">
    <param name="movie" value="app:/SWFFile.swf"></param>
    <param name="wmode" value="opaque"></param>
</object>
```

スクリプトを使用してコンテンツを動的に読み込むこともできます。次の例では、object ノードを作成して urlString パラメーターに指定された SWF ファイルを表示します。この例では、elementID パラメーターによって指定された ID を使用してページエレメントの子としてノードを追加します。

```
<script>
function showSWF(urlString, elementID){
    var displayContainer = document.getElementById(elementID);
    var flash = createSWFObject(urlString, 'opaque', 650, 650);
    displayContainer.appendChild(flash);
}
function createSWFObject(urlString, wmodeString, width, height){
    var SWFObject = document.createElement("object");
    SWFObject.setAttribute("type","application/x-shockwave-flash");
    SWFObject.setAttribute("width","100%");
    SWFObject.setAttribute("height","100%");
    var movieParam = document.createElement("param");
    movieParam.setAttribute("name","movie");
    movieParam.setAttribute("value",urlString);
    SWFObject.appendChild(movieParam);
    var wmodeParam = document.createElement("param");
    wmodeParam.setAttribute("name","wmode");
    wmodeParam.setAttribute("value",wmodeString);
    SWFObject.appendChild(wmodeParam);
    return SWFObject;
}
</script>
```

HTMLLoader オブジェクトが拡大／縮小または回転されている場合、または alpha プロパティが 1.0 以外の値に設定されている場合、SWF コンテンツは表示されません。AIR 1.5.2 より前のバージョンでは、どの wmode 値が設定されていようと、SWF コンテンツは透明ウィンドウに表示されませんでした。

注意：埋め込まれた SWF オブジェクトがビデオファイルなどの外部アセットを読み込もうとすると、ビデオファイルの絶対パスが HTML ファイルで指定されていない場合に SWF コンテンツが正しくレンダリングされないことがあります。ただし、埋め込まれた SWF オブジェクトでは、相対パスを使用して外部イメージファイルを読み込むことができます。

次の例では、HTML コンテンツに埋め込まれた SWF オブジェクトを通じて外部アセットを読み込む方法を示します。

```
var imageLoader;

function showSWF(urlString, elementID){
    var displayContainer = document.getElementById(elementID);
    imageLoader = createSWFObject(urlString,650,650);
    displayContainer.appendChild(imageLoader);
}

function createSWFObject(urlString, width, height){

    var SWFObject = document.createElement("object");
    SWFObject.setAttribute("type","application/x-shockwave-flash");
    SWFObject.setAttribute("width","100%");
    SWFObject.setAttribute("height","100%");

    var movieParam = document.createElement("param");
    movieParam.setAttribute("name","movie");
    movieParam.setAttribute("value",urlString);
    SWFObject.appendChild(movieParam);

    var flashVars = document.createElement("param");
    flashVars.setAttribute("name","FlashVars");

    //Load the asset inside the SWF content.
    flashVars.setAttribute("value","imgPath=air.jpg");
    SWFObject.appendChild(flashVars);

    return SWFObject;
}
function loadImage()
{
  showSWF("ImageLoader.swf", "imageSpot");

}
```

次に示す ActionScript の例では、HTML ファイルによって渡されるイメージのパスが読み取られ、そのイメージがステージに読み込まれます。

```
package
{
      import flash.display.Sprite;
      import flash.display.LoaderInfo;
      import flash.display.StageScaleMode;
      import flash.display.StageAlign;
      import flash.display.Loader;
      import flash.net.URLRequest;

      public class ImageLoader extends Sprite
      {
            public function ImageLoader()
            {

                  var flashvars = LoaderInfo(this.loaderInfo).parameters;

                  if(flashvars.imgPath){
                        var imageLoader = new Loader();
                        var image = new URLRequest(flashvars.imgPath);
                        imageLoader.load(image);
                        addChild(imageLoader);
                        imageLoader.x = 0;
                        imageLoader.y = 0;
                        stage.scaleMode=StageScaleMode.NO_SCALE;
                        stage.align=StageAlign.TOP_LEFT;
                  }
            }
      }
}
```

HTML ページ内での ActionScript ライブラリの使用

Adobe AIR 1.0 およびそれ以降

AIR では、コンパイルされた SWF ファイルの ActionScript クラスをページがインポートできるように HTML の script エレメントを拡張します。例えば、ルートアプリケーションフォルダーの **lib** サブディレクトリにある myClasses.swf という名前のライブラリを読み込むには、HTML ファイル内に次の script タグを記述します。

```
<script src="lib/myClasses.swf" type="application/x-shockwave-flash"></script>
```

重要：ライブラリが適切に読み込まれるには、type 属性を type="application/x-shockwave-flash" と指定する必要があります。

SWF コンテンツが Flash Player 10 または AIR 1.5 SWF としてコンパイルされる場合、アプリケーション記述ファイルの XML 名前空間を AIR 1.5 名前空間に設定する必要があります。

AIR ファイルをパッケージ化するときには、lib ディレクトリと myClasses.swf ファイルも含める必要があります。

JavaScript Window オブジェクトの runtime プロパティを通じて、読み込まれたクラスにアクセスします。

```
var libraryObject = new window.runtime.LibraryClass();
```

SWF ファイル内のクラスをパッケージに編成する場合、パッケージ名も指定する必要があります。例えば、LibraryClass 定義が **utilities** という名前のパッケージ内にある場合、次のようにしてクラスのインスタンスを作成します。

```
var libraryObject = new window.runtime.utilities.LibraryClass();
```

注意：AIR で HTML ページの一部として使用する ActionScript SWF ライブラリをコンパイルするには、acompc コンパイラーを使用します。acompc ユーティリティは、Flex SDK の一部です。詳しくは、Flex SDK ドキュメントを参照してください。

読み込まれた ActionScript ファイルからの HTML DOM オブジェクトおよび JavaScript オブジェクトへのアクセス

Adobe AIR 1.0 およびそれ以降

<script> タグを使用してページに読み込まれた SWF ファイルの ActionScript から HTML ページのオブジェクトにアクセスするには、window や document などの JavaScript オブジェクトへの参照を、ActionScript コードに定義された関数に渡します。JavaScript オブジェクト（または渡された参照からアクセスできる他のオブジェクト）にアクセスする関数内の参照を使用します。

例えば、次のような HTML ページがあるとします。

```
<html>
    <script src="ASLibrary.swf" type="application/x-shockwave-flash"></script>
    <script>
        num = 254;
        function getStatus() {
            return "OK.";
        }
        function runASFunction(window){
            var obj = new runtime.ASClass();
            obj.accessDOM(window);
        }
    </script>
    <body onload="runASFunction">
        <p id="p1">Body text.</p>
    </body>
</html>
```

この簡単な HTML ページには、**num** という名前の JavaScript 変数と **getStatus()** という名前の JavaScript 関数があります。2 つともページの window オブジェクトのプロパティです。また、window.document オブジェクトには名前付きの P エレメント（ID は **p1**）があります。

ページは、ASClass というクラスを含む ActionScript ファイル「ASLibrary.swf」を読み込みます。ASClass には、JavaScript オブジェクトの値を単純に追跡する accessDOM() という名前の関数が定義されています。accessDOM() メソッドは、JavaScript Window オブジェクトをパラメーターとして使用します。次の定義に示すように、この Window 参照を使用すると、変数、関数および DOM エレメントなど、ページ内の他のオブジェクトにアクセスできます。

```
public class ASClass{
    public function accessDOM(window:*):void {
        trace(window.num); // 254
        trace(window.document.getElementById("p1").innerHTML); // Body text..
        trace(window.getStatus()); // OK.
    }
}
```

読み込まれた ActionScript クラスから HTML ページのプロパティを取得することも設定することもできます。例えば、次の関数はページ上の p1 エレメントのコンテンツを設定し、ページ上の foo JavaScript 変数の値を設定します。

```
public function modifyDOM(window:*):void {
    window.document.getElementById("p1").innerHTML = "Bye";
    window.foo = 66;
```

Date オブジェクトと RegExp オブジェクトの変換

Adobe AIR 1.0 およびそれ以降

JavaScript 言語と ActionScript 言語はいずれも Date クラスと RegExp クラスを定義していますが、これらの型のオブジェクトは 2 つの実行コンテキスト間で自動的に変換されません。Date オブジェクトと RegExp オブジェクトを代わりの実行コンテキストのプロパティまたは関数パラメーターの設定に使用する前に、対応する型に変換する必要があります。

例えば、次の ActionScript コードは、jsDate という名前の JavaScript の Date オブジェクトを ActionScript の Date オブジェクトに変換します。

```
var asDate:Date = new Date(jsDate.getMilliseconds());
```

次の ActionScript コードは、jsRegExp という名前の JavaScript の RegExp オブジェクトを ActionScript の RegExp オブジェクトに変換します。

```
var flags:String = "";
if (jsRegExp.dotAll) flags += "s";
if (jsRegExp.extended) flags += "x";
if (jsRegExp.global) flags += "g";
if (jsRegExp.ignoreCase) flags += "i";
if (jsRegExp.multiline) flags += "m";
var asRegExp:RegExp = new RegExp(jsRegExp.source, flags);
```

異なるセキュリティサンドボックス内のコンテンツのクロススクリプト

Adobe AIR 1.0 およびそれ以降

ランタイムセキュリティモデルでは、発信元が異なるコードを分離します。異なるセキュリティサンドボックス内のコンテンツをクロススクリプトすると、あるセキュリティサンドボックス内のコンテンツから、別のサンドボックス内の選択されたプロパティおよびメソッドにアクセスできるようになります。

AIR セキュリティサンドボックスおよび JavaScript コード

Adobe AIR 1.0 およびそれ以降

AIR では、あるドメインのコードが別のドメインのコンテンツを操作するのを避けるために、同一生成元のポリシーを適用します。すべてのファイルが生成元に基づいたサンドボックス内に配置されます。通常、アプリケーションサンドボックス内のコンテンツは、同一生成元の原則を破って、アプリケーションインストールディレクトリの外部から読み込まれたコンテンツをクロススクリプトすることはできません。ただし、AIR には非アプリケーションコンテンツのクロススクリプトを可能にするいくつかのテクニックがあります。

まず、フレームまたはインラインフレームを使用してアプリケーションコンテンツを異なるセキュリティサンドボックスにマップする方法があります。アプリケーションのサンドボックス領域から読み込まれたページは、リモートドメインから読み込まれたページのように動作します。例えば、アプリケーションコンテンツを **example.com** ドメインにマップすると、そのコンテンツでは example.com から読み込まれたページ間のクロススクリプトを実行できます。

このテクニックはアプリケーションコンテンツを異なるサンドボックスに配置するので、このコンテンツ内のコードは評価されたストリング内のコード実行に関する制限も受けなくなります。このサンドボックスのマッピングを使用すると、リモートコンテンツをクロススクリプトする必要がない場合でも制限を緩和できます。この方法でのコンテンツのマッピング

は、多くの JavaScript フレームワークの中の 1 つを操作する場合や、評価するストリングに依存する既存のコードを操作する場合に特に有用です。ただし、コンテンツがアプリケーションサンドボックスの外部で実行される場合には、信頼できないコンテンツが挿入されて実行される可能性があります。この付加的なリスクに対して、考慮し、防御する必要があります。

同時に、別のサンドボックスにマップされるアプリケーションコンテンツは AIR API へのアクセス権を失うので、サンドボックスのマッピングを使用してアプリケーションサンドボックスの外部で実行されるコードに AIR 機能を公開することはできません。

別のクロススクリプトの方法としては、非アプリケーションサンドボックス内のコンテンツと、アプリケーションサンドボックス内のその親ドキュメントの間にサンドボックスブリッジと呼ばれるインターフェイスを作成します。サンドボックスブリッジを使用すると、親コンテンツが定義したプロパティとメソッドに子コンテンツからアクセスしたり、子コンテンツで定義したプロパティとメソッドに親コンテンツからアクセスしたりできます。

最後の方法として、アプリケーションサンドボックス（および必要に応じて他のサンドボックス）からクロスドメインの XMLHttpRequest を実行できます。

詳しくは、12 ページの「HTML の frame エレメントと iframe エレメント」、72 ページの「Adobe AIR の HTML セキュリティ」および 6 ページの「XMLHttpRequest オブジェクト」を参照してください。

アプリケーションコンテンツの非アプリケーションサンドボックスへの読み込み

Adobe AIR 1.0 およびそれ以降

アプリケーションインストールディレクトリ以外から読み込まれたコンテンツのクロススクリプトをアプリケーションコンテンツで安全に実行できるようにするには、frame エレメントまたは iframe エレメントを使用して、アプリケーションコンテンツを外部コンテンツとして同じセキュリティサンドボックスに読み込みます。リモートコンテンツのクロススクリプトは必要ないが、アプリケーションサンドボックス外のアプリケーションのページを読み込む必要がある場合は、元のドメインとして http://localhost/ または問題のない他の値を指定して、同じテクニックを使用できます。

AIR では、frame エレメントに sandboxRoot および documentRoot という新しい属性を追加しています。これにより、フレームに読み込まれるアプリケーションファイルを非アプリケーションサンドボックスにマップするかどうかを指定できます。sandboxRoot URL の下のパスに解決されるファイルは、代わりに documentRoot ディレクトリから読み込まれます。セキュリティのため、この方法で読み込まれるアプリケーションコンテンツは、実際に sandboxRoot URL から読み込まれているかのように扱われます。

sandboxRoot プロパティは、フレームコンテンツを配置するサンドボックスおよびドメインの決定に使用する URL を指定します。file:、http: または https: URL スキームを使用する必要があります。相対的な URL を指定する場合、コンテンツはアプリケーションサンドボックス内に留まります。

documentRoot プロパティでは、フレームコンテンツの読み込み元のディレクトリを指定します。file:、app: または app-storage: URL スキームを使用する必要があります。

次の例では、アプリケーションの sandbox サブディレクトリにインストールされたコンテンツをマップし、リモートサンドボックスおよび www.example.com ドメインで実行します。

```
<iframe
    src="http://www.example.com/local/ui.html"
    sandboxRoot="http://www.example.com/local/"
    documentRoot="app:/sandbox/">
</iframe>
```

ui.html ページでは、次のスクリプトタグを使用してローカル sandbox フォルダーから javascript ファイルを読み込むことができます。

```
<script src="http://www.example.com/local/ui.js"></script>
```

また、次のようなスクリプトタグを使用してリモートサーバーのディレクトリからコンテンツを読み込むこともできます。

```
<script src="http://www.example.com/remote/remote.js"></script>
```

sandboxRoot URL は、リモートサーバー上の同じ URL にあるコンテンツをマスクします。上の例では、AIR によって要求がローカルアプリケーションディレクトリに再マップされるので、www.example.com/local/（またはそのサブディレクトリ）にあるリモートコンテンツにはアクセスできません。要求は、ページナビゲーションからの派生か、XMLHttpRequest からの派生か、コンテンツを読み込む他の手段からの派生かによって、再マップされます。

サンドボックスブリッジインターフェイスの設定

Adobe AIR 1.0 およびそれ以降

アプリケーションサンドボックス内のコンテンツが非アプリケーションサンドボックス内のコンテンツによって定義されたプロパティやメソッドにアクセスする必要がある場合、または非アプリケーションコンテンツがアプリケーションサンドボックス内のコンテンツによって定義されたプロパティとメソッドにアクセスする必要がある場合に、サンドボックスブリッジを使用できます。子ドキュメントの window オブジェクトの childSandboxBridge プロパティと parentSandboxBridge プロパティを使用してブリッジを作成します。

子サンドボックスブリッジの確立

Adobe AIR 1.0 およびそれ以降

childSandboxBridge プロパティを使用すると、子ドキュメントで親ドキュメント内のコンテンツにインターフェイスを公開できます。インターフェイスを公開するには、childSandbox プロパティを子ドキュメント内の関数またはオブジェクトに設定します。設定すると、親ドキュメント内のコンテンツからオブジェクトまたは関数にアクセスできます。次の例では、子ドキュメントで実行中のスクリプトで、関数とプロパティを 1 つずつ含むオブジェクトをその親に公開する方法を示します。

```
var interface = {};
interface.calculatePrice = function(){
    return ".45 cents";
}
interface.storeID = "abc"
window.childSandboxBridge = interface;
```

この子コンテンツが「子」の ID を割り当てられたインラインフレームに読み込まれていれば、次のようにして、フレームの childSandboxBridge プロパティを読み取ることによって親コンテンツからインターフェイスにアクセスできます。

```
var childInterface = document.getElementById("child").contentWindow.childSandboxBridge;
air.trace(childInterface.calculatePrice()); //traces ".45 cents"
air.trace(childInterface.storeID)); //traces "abc"
```

親サンドボックスブリッジの確立

Adobe AIR 1.0 およびそれ以降

parentSandboxBridge プロパティを使用すると、親ドキュメントで子ドキュメント内のコンテンツにインターフェイスを公開できます。インターフェイスを公開するには、親ドキュメントで子ドキュメントの parentSandbox プロパティを親ドキュメント内に定義された関数またはオブジェクトに設定します。設定すると、子ドキュメント内のコンテンツからオブジェクトまたは関数にアクセスできます。次の例では、親フレームで実行中のスクリプトで、関数を含むオブジェクトを子ドキュメントに公開する方法を示します。

```
var interface = {};
interface.save = function(text){
    var saveFile = air.File("app-storage:/save.txt");
    //write text to file
}
document.getElementById("child").contentWindow.parentSandboxBridge = interface;
```

このインターフェイスを使用すると、子フレーム内のコンテンツで、テキストを save.txt という名前のファイルに保存できますが、ファイルシステムに他の方法ではアクセスできません。子コンテンツは、save 関数を次のように呼び出します。

```
var textToSave = "A string.";
window.parentSandboxBridge.save(textToSave);
```

アプリケーションコンテンツは、他のサンドボックスにできる限り範囲の狭いインターフェイスを公開する必要があります。非アプリケーションコンテンツは、偶発的に、または悪意によってコードが挿入される可能性があるので、本来、信頼できないものです。親サンドボックスブリッジによって公開するインターフェイスの誤用を防ぐ適切な安全対策を実施する必要があります。

ページ読み込み時の親サンドボックスブリッジへのアクセス

Adobe AIR 1.0 およびそれ以降

子ドキュメント内のスクリプトから親サンドボックスブリッジにアクセスするには、スクリプトを実行する前にブリッジを設定する必要があります。新しいページの DOM が作成された後で、スクリプトが解析されるか DOM エレメントが追加される前に、ウィンドウ、フレームおよびインラインフレームのオブジェクトは、dominitialize イベントを送出します。dominitialize イベントを使用して、子ドキュメント内のすべてのスクリプトがアクセスできるページ構成シーケンスで、前もってブリッジを確立できます。

次の例は、子フレームから送出された dominitialize イベントに対応して親サンドボックスブリッジを作成する方法を示しています。

```
<html>
<head>
<script>
var bridgeInterface = {};
bridgeInterface.testProperty = "Bridge engaged";
function engageBridge(){
    document.getElementById("sandbox").contentWindow.parentSandboxBridge = bridgeInterface;
}
</script>
</head>
<body>
<iframe id="sandbox"
            src="http://www.example.com/air/child.html"
            documentRoot="app:/"
            sandboxRoot="http://www.example.com/air/"
            ondominitialize="engageBridge()"/>
</body>
</html>
```

次の child.html ドキュメントは、子コンテンツから親サンドボックスブリッジにアクセスする方法を示しています。

```
<html>
    <head>
        <script>
            document.write(window.parentSandboxBridge.testProperty);
        </script>
    </head>
    <body></body>
</html>
```

フレームではなく、子ウィンドウ上で dominitialize イベントをリッスンするには、次のようにして、window.open() 関数によって作成された新しい子ウィンドウオブジェクトにリスナーを追加する必要があります。

```
var childWindow = window.open();
childWindow.addEventListener("dominitialize", engageBridge());
childWindow.document.location = "http://www.example.com/air/child.html";
```

この場合、アプリケーションコンテンツを非アプリケーションサンドボックスにマップする方法はありません。このテクニックは、child.html がアプリケーションディレクトリの外部から読み込まれる場合にのみ有用です。ウィンドウ内のアプリケーションコンテンツを非アプリケーションサンドボックスにマップすることはできますが、フレームを使用して子ドキュメントを読み込んで目的のサンドボックスにマップする中間ページを最初に読み込む必要があります。

HTMLLoader クラスの createRootWindow() 関数を使用してウィンドウを作成する場合、新しいウィンドウは createRootWindow() の呼び出し元のドキュメントの子ではありません。したがって、呼び出し元ウィンドウから、新しいウィンドウに読み込まれる非アプリケーションコンテンツへのサンドボックスブリッジを作成することはできません。その代わり、フレームを使用して子ドキュメントを読み込む新しいウィンドウ内の中間ページを読み込む必要があります。その後、新しいウィンドウの親ドキュメントから、フレームに読み込まれた子ドキュメントへのブリッジを確立できます。

第 3 章: AIR における HTML 関連イベントの処理

Adobe AIR 1.0 およびそれ以降

イベント処理システムは、ユーザー入力やシステムイベントにプログラムが応答するための便利な仕組みです。Adobe® AIR® のイベントモデルは、便利なだけでなく、標準に準拠しています。業界標準のイベント処理アーキテクチャであるドキュメントオブジェクトモデル（DOM）Level 3 Events 仕様に基づいたイベントモデルにより、強力で直感的に使用できるイベント処理ツールが提供されます。

HTMLLoader イベント

Adobe AIR 1.0 およびそれ以降

HTMLLoader オブジェクトは、次の Adobe® ActionScript®3.0 イベントを送出します。

イベント	説明
htmlDOMInitialize	HTML ドキュメントが作成されたとき（ただし、スクリプトが解析される前、または DOM ノードがページに追加される前）に送出されます。
complete	読み込み操作に対する応答として HTML DOM が作成されたとき（HTML ページの onload イベントの直後）に送出されます。
htmlBoundsChanged	contentWidth および contentHeight プロパティの一方または両方が変更されたときに送出されます。
locationChange	HTMLLoader の location プロパティが変更されたときに送出されます。
locationChanging	ユーザーによるナビゲーション、JavaScript の呼び出しまたはリダイレクトにより、HTMLLoader の location が変更される前に送出されます。locationChanging イベントは、load()、loadString()、reload()、historyGo()、historyForward() または historyBack() メソッドの呼び出し時には送出されません。 送出されたイベントオブジェクトの preventDefault() メソッドを呼び出すと、ナビゲーションがキャンセルされます。 システムブラウザーでリンクが開いている場合、HTMLLoader は location を変更しないので、locationChanging イベントは送出されません。
scroll	HTML エンジンがスクロール位置を変更するたびに送出されます。scroll イベントは、ページ内のアンカーリンク（# リンク）への移動、または window.scrollTo() メソッドの呼び出しによって発生します。テキスト入力欄またはテキスト領域にテキストを入力した場合にも、scroll イベントが発生する可能性があります。
uncaughtScriptException	HTMLLoader で JavaScript 例外が発生し、その例外が JavaScript コードでキャッチされない場合に送出されます。

AIR クラスのイベント処理と HTML DOM の他のイベント処理との相違

Adobe AIR 1.0 およびそれ以降

HTML DOM では、イベントを処理する方法がいくつかあります。

- on イベントハンドラーを HTML エレメントの開始タグで定義します。例を次に示します。

```
<div id="myDiv" onclick="myHandler()">
```

- コールバック関数プロパティ。例を次に示します。

```
document.getElementById("myDiv").onclick
```

- addEventListener() メソッドを使用して登録するイベントリスナー。例を次に示します。

```
document.getElementById("myDiv").addEventLister("click", clickHandler)
```

ただし、ランタイムオブジェクトは DOM では使用されないので、イベントリスナーを追加するには、AIR オブジェクトの addEventListener() メソッドを呼び出すことが唯一の方法です。

JavaScript の場合と同様に、AIR オブジェクトから送出されるイベントは、デフォルト動作と関連付けることができます（デフォルト動作とは、ある種のイベントに対する通常の結果として AIR で実行されるアクションのことです）。

ランタイムオブジェクトが送出するイベントオブジェクトは、Event クラスまたはそのサブクラスのインスタンスです。イベントオブジェクトには、特定のイベントに関する情報が格納されているのに加え、イベントオブジェクトを操作する際に役立つメソッドが備わっています。例えば AIR は、ファイルの非同期読み取り中に I/O エラーイベントを検出すると、イベントオブジェクト（IOErrorEvent クラスのインスタンス）を作成し、その特定の I/O エラーイベントを表します。

イベントハンドラーコードを作成する場合、そのコードは次のような同じ基本構造に基づいています。

```
function eventResponse(eventObject)
{
    // Actions performed in response to the event go here.
}

eventTarget.addEventListener(EventType.EVENT_NAME, eventResponse);
```

このコードは 2 つのことを行います。まず、ハンドラー関数を定義します。これにより、イベントに対する応答として実行されるアクションを指定します。次に、ソースオブジェクトの addEventListener() メソッドを呼び出します。これは実際には、指定したイベントに対して関数を登録し、イベントが発生したときにハンドラーアクションが実行されるようにするための処理です。イベントが実際に発生すると、イベントターゲットは、イベントリスナーに登録されているすべての関数とメソッドのリストを確認します。次にイベントターゲットは、リストの関数とメソッドを順に呼び出し、イベントオブジェクトをパラメーターとして渡します。

デフォルト動作

Adobe AIR 1.0 およびそれ以降

イベントに応答するコードは、通常は開発者が記述します。しかし、イベントに対して何らかの決まった動作が非常によく行われる場合については、一般的な処理が AIR によって自動的に実行されるようになっています（開発者が特にこの動作をキャンセルするためのコードを記述した場合を除く）。AIR によって自動的に実行されるそうした動作を、デフォルト動作と呼びます。

例えば、ユーザーがアプリケーションのウィンドウにある閉じるボックスをクリックする場合には、ウィンドウが閉じることが期待されているため、この動作は AIR にビルトインされています。このデフォルト動作が不要な場合は、イベント処理システムを使用することでキャンセルできます。ユーザーがウィンドウの閉じるボックスをクリックすると、そのウィンドウを表す NativeWindow オブジェクトが closing イベントを送出します。ランタイムがウィンドウを閉じないようにするには、送出したイベントオブジェクトの preventDefault() メソッドを呼び出す必要があります。

デフォルト動作の中には、キャンセルできないものもあります。例えばこのランタイムは、FileStream オブジェクトがデータをファイルに書き込む際に、OutputProgressEvent オブジェクトを生成します。この場合、ファイルの内容が新しいデータで更新されることがデフォルト動作となります。このデフォルト動作はキャンセルできません。

多くの種類のイベントオブジェクトには、デフォルト動作が関連付けられていません。例えば Sound オブジェクトは、ID3 情報を提供するために十分なデータが MP3 ファイルから読み取られると、id3 イベントを送出しますが、デフォルト動作は関連付けられていません。API ドキュメントの Event クラスとそのサブクラスに関する項目には、各種イベントの一覧と、関連付けられたデフォルト動作がある場合はその内容、およびデフォルト動作をキャンセルできるかどうかが説明されています。

注意：デフォルト動作は、ランタイムによって直接送出されるイベントオブジェクトにのみ関連付けられており、JavaScript で作成したプログラムによって送出されるイベントオブジェクトに関連付けられたデフォルト動作は存在しません。例えば、EventDispatcher クラスのメソッドを使用してイベントオブジェクトを送出できますが、イベントを送出してもデフォルト動作はトリガーされません。

イベントフロー

Adobe AIR 1.0 およびそれ以降

AIR で実行される SWF ファイルコンテンツは、ビジュアルコンテンツの表示に ActionScript 3.0 表示リストアーキテクチャを使用します。ActionScript 3.0 表示リストは、コンテンツの親子関係を設定すると共に、親と子の表示オブジェクトの間で伝達される、SWF ファイルコンテンツのイベント（例えばマウスクリックイベント）の親子関係を設定します。HTML DOM には、DOM エレメント間でのみやり取りされる、専用の独立したイベントフローがあります。AIR 向けに HTML ベースのアプリケーションを作成する場合､開発者は ActionScript 3.0 表示リストではなく HTML DOM を主に使用するので、AIR リファレンスドキュメントに記載されているイベントフェーズに関する情報は、通常は無視してかまいません。

Adobe AIR イベントオブジェクト

Adobe AIR 1.0 およびそれ以降

イベント処理システムにおいて、イベントオブジェクトには主として 2 つの用途があります。1 つは、個々のイベントに関する情報を各種プロパティに格納して、具体的なイベントを表現することです。もう 1 つは、各種メソッドを使用してイベントオブジェクトを操作し、イベント処理システムの動作を変化させることです。

AIR API では、各 AIR API クラスから送出されるすべてのイベントオブジェクトの基本クラスとなる Event クラスが定義されています。Event クラスは、すべてのイベントオブジェクトに共通する基本的なプロパティとメソッドを備えています。

Event オブジェクトを使用する場合は、最初に Event クラスのプロパティとメソッド、および Event クラスのサブクラスの機能について理解することが重要です。

Event クラスのプロパティについて

Adobe AIR 1.0 およびそれ以降

Event クラスでは、イベントに関する重要な情報を提供する、いくつかの読み取り専用プロパティおよび定数が定義されています。特に重要なものを次に示します。

- Event.type は、イベントオブジェクトが表すイベントのタイプを示します。
- Event.cancelable は、そのイベントに関連付けられているデフォルト動作（存在する場合）がキャンセル可能かどうかを報告するブール値です。
- イベントフロー情報はその他のプロパティに格納され、AIR で実行される SWF コンテンツで ActionScript 3.0 を使用している場合にのみ使用されます。

イベントオブジェクトのタイプ

Adobe AIR 1.0 およびそれ以降

すべてのイベントオブジェクトには、それぞれイベントタイプが設定されます。イベントタイプはストリング値として Event.type プロパティに格納されます。イベントオブジェクトのタイプを知ることができると、コードを作成する際に、タイプに応じてオブジェクトの処理方法を区別できて便利です。例えば次のコードでは、myFileStream によって送出される complete イベントに応答する fileReadHandler() リスナー関数を登録します。

```
myFileStream.addEventListener(Event.COMPLETE, fileReadHandler);
```

AIR の Event クラスでは、ランタイムオブジェクトによって送出されるイベントのタイプを表す COMPLETE、CLOSING および ID3 などの多数のクラス定数が定義されています。これらの定数については、『HTML 開発者用 Adobe AIR API リファレンスガイド』の Event クラスのページで説明されています。

イベント定数を使用すると、特定のイベントタイプを簡単に参照できます。ストリング値の代わりに定数を使用することで、入力ミスをよりすばやく見つけることができます。コード内の定数名にスペルミスがある場合、JavaScript パーサによってその誤りが検出されます。定数名ではなく、イベントストリングにスペルミスがあると、イベントハンドラーは、決して送出されることのないタイプのイベントについて登録されることになります。そのため、イベントリスナーを追加する場合は、次のコードを使用することをお勧めします。

```
myFileStream.addEventListener(Event.COMPLETE, htmlRenderHandler);
```

次のコードはお勧めできません。

```
myFileStream.addEventListener("complete", htmlRenderHandler);
```

デフォルト動作に関する情報

Adobe AIR 1.0 およびそれ以降

コードで cancelable プロパティを調べることにより、特定のイベントオブジェクトに対するデフォルト動作をキャンセルできるかどうかを知ることができます。cancelable プロパティの値は Boolean 型で、デフォルト動作がキャンセル可能かどうかを示します。デフォルト動作をキャンセルできるイベントは少数ですが、キャンセルできる場合は、関連付けられた動作を preventDefault() メソッドで無効化できます。詳しくは、44 ページの「イベントのデフォルト動作のキャンセル」を参照してください。

Event クラスのメソッドについて

Adobe AIR 1.0 およびそれ以降

Event クラスのメソッドは、次の 3 種類に大別されます。

- ユーティリティメソッド：イベントオブジェクトのコピーを作成するか、またはストリングに変換します。
- イベントフローメソッド：イベントフローからイベントオブジェクトを削除します（主としてランタイムの SWF コンテンツで ActionScript 3.0 を使用する場合に使用します。42 ページの「イベントフロー」を参照してください）。
- デフォルト動作メソッド：デフォルト動作をキャンセルするか、またはキャンセル済みかどうかを確認します。

Event クラスのユーティリティメソッド

Adobe AIR 1.0 およびそれ以降

Event クラスには、2 種類のユーティリティメソッドがあります。clone() メソッドは、イベントオブジェクトのコピーを作成します。toString() メソッドは、イベントオブジェクトの各種プロパティとそれらの値を表すストリング表現を生成します。

イベントのデフォルト動作のキャンセル

Adobe AIR 1.0 およびそれ以降

デフォルト動作のキャンセルに関連するメソッドには、preventDefault() メソッドと isDefaultPrevented() メソッドの 2 つがあります。preventDefault() メソッドを呼び出すと、イベントに関連付けられたデフォルト動作をキャンセルできます。あるイベントオブジェクトについて preventDefault() が既に呼び出されているかどうかを確認するには、isDefaultPrevented() メソッドを使用します。

preventDefault() メソッドは、イベントのデフォルト動作がキャンセル可能な場合にのみ機能します。あるイベントに対する動作がキャンセル可能かどうかを確認するには、API ドキュメントを参照するか、またはイベントオブジェクトの cancelable プロパティを調べます。

デフォルト動作をキャンセルしても、イベントフローによるイベントオブジェクト処理の進行には影響しません。イベントフローからイベントオブジェクトを削除するには、Event クラスのイベントフローメソッドを使用します。

Event クラスのサブクラス

Adobe AIR 1.0 およびそれ以降

多くのイベントは、Event クラスに定義されている共通のプロパティセットを使用すれば十分に表現できます。しかし、それ以外のイベントを表現するためには、Event クラスでは提供されていないプロパティが必要になります。そのようなイベントのために、AIR API には Event クラスのサブクラスがいくつか定義されています。

各サブクラスには、そのイベントのカテゴリに特有のプロパティおよびイベントタイプが追加されています。例えば、マウス入力に関連するイベントには、イベントが発生した時点でのマウスの位置を示すプロパティがあります。同様に、InvokeEvent クラスを使用すると、呼び出すファイルのファイルパスと、コマンドライン呼び出しでパラメーターとして渡される引数を格納するプロパティを追加できます。

多くの Event サブクラスでは、そのサブクラスに関連付けられているイベントタイプを表す追加の定数が定義されています。例えば、FileListEvent クラスでは、directoryListing および selectMultiple イベントタイプを表す定数が定義されています。

JavaScript を使用したランタイムイベントの処理

Adobe AIR 1.0 およびそれ以降

ランタイムクラスでは、addEventListener() メソッドを使用することで、イベントハンドラーを追加できます。あるイベントに対するハンドラー関数を追加するには、そのイベントを送出したオブジェクトの addEventListener() メソッドを呼び出し、イベントタイプと処理関数を指定します。例えば、ユーザーがウィンドウのタイトルバーにある閉じるボタンをクリックしたときに送出される closing イベントをリッスンするには、次のステートメントを使用します。

```
window.nativeWindow.addEventListener(air.NativeWindow.CLOSING, handleWindowClosing);
```

addEventListener() メソッドの type パラメーターはストリングですが、AIR API では、すべてのランタイムイベントタイプに対して定数が定義されています。これらの定数を使用すると、ストリング version を使用するよりも、type パラメーターに入力したスペルの誤りを発見しやすくなります。

イベントハンドラー関数の作成

Adobe AIR 1.0 およびそれ以降

次のコードでは、メインウィンドウの位置に関する情報を表示する簡単な HTML ファイルを作成します。ハンドラー関数は moveHandler() という名前で、メインウィンドウの move イベント（NativeWindowBoundsEvent クラスによって定義されています）をリッスンします。

```
<html>
    <script src="AIRAliases.js" />
    <script>
        function init() {
            writeValues();
            window.nativeWindow.addEventListener(air.NativeWindowBoundsEvent.MOVE,
                                                 moveHandler);
        }
        function writeValues() {
            document.getElementById("xText").value = window.nativeWindow.x;
            document.getElementById("yText").value = window.nativeWindow.y;
        }
        function moveHandler(event) {
            air.trace(event.type); // move
            writeValues();
        }
    </script>
    <body onload="init()" />
        <table>
            <tr>
                <td>Window X:</td>
                <td><textarea id="xText"></textarea></td>
            </tr>
            <tr>
                <td>Window Y:</td>
                <td><textarea id="yText"></textarea></td>
            </tr>
        </table>
    </body>
</html>
```

ユーザーがウィンドウを移動すると、textarea エレメントにウィンドウの更新された X および Y 位置が表示されます。

イベントオブジェクトが引数として moveHandler() メソッドに渡されています。この event パラメーターにより、ハンドラー関数はイベントオブジェクトを確認できます。この例では、イベントオブジェクトの type プロパティを使用して、イベントが move イベントであることを報告しています。

注意：listener パラメーターを指定する際には、括弧を使用しないでください。例えば、次に示す addEventListener() メソッドの呼び出しでは、moveHandler() 関数が括弧なしで指定されています。addEventListener(Event.MOVE, moveHandler)

addEventListener() メソッドには、他にも 3 つのパラメーター（useCapture、priority、および useWeakReference）があります。詳しくは、『**HTML 開発者用 Adobe AIR API リファレンスガイド**』を参照してください。

イベントリスナーの削除

Adobe AIR 1.0 およびそれ以降

不要になったイベントリスナーを削除するには、removeEventListener() メソッドを使用します。不要になったリスナーは削除することをお勧めします。必須パラメーターには eventName および listener があります。これらは、addEventListener() メソッドの必須パラメーターと同じです。

移動する HTML ページでのイベントリスナーの削除

Adobe AIR 1.0 およびそれ以降

HTML コンテンツを移動する場合、または HTML コンテンツを格納しているウィンドウが閉じられたためにそのコンテンツが破棄される場合、読み込みを解除されたページ上のオブジェクトを参照するイベントリスナーは、自動的には削除されません。オブジェクトが既に読み込みを解除されたハンドラーにイベントを送出すると、「The application attempted to reference a JavaScript object in an HTML page that is no longer loaded.」というエラーメッセージが表示されます。

このエラーを回避するには、HTML ページが読み込みを解除される前に、ページ内の JavaScript イベントリスナーを削除します。（HTMLLoader オブジェクト内での）ページナビゲーションの場合には、イベントリスナーを window オブジェクトの unload イベントの処理中に削除します。

例えば、次の JavaScript コードでは、uncaughtScriptException イベントをリッスンするイベントリスナーを削除します。

```
window.onunload = cleanup;
window.htmlLoader.addEventListener('uncaughtScriptException', uncaughtScriptException);
function cleanup()
{
    window.htmlLoader.removeEventListener('uncaughtScriptException',
                            uncaughtScriptExceptionHandler);
}
```

HTML コンテンツを格納したウィンドウを閉じる際にエラーが発生しないようにするには、NativeWindow オブジェクト（window.nativeWindow）の closing イベントに対する応答として、cleanup 関数を呼び出します。例えば、次の JavaScript コードでは、uncaughtScriptException イベントをリッスンするイベントリスナーを削除します。

```
 window.nativeWindow.addEventListener(air.Event.CLOSING, cleanup);
function cleanup()
{
    window.htmlLoader.removeEventListener('uncaughtScriptException',
                            uncaughtScriptExceptionHandler);
}
```

また、イベントの処理が 1 回のみの場合、実行後すぐにイベントリスナーを削除することで、このエラーが発生しないようにできます。例えば、次の JavaScript コードでは、HTMLLoader クラスの createRootWindow() メソッドを呼び出し、complete イベントのイベントリスナーを追加することで、html ウィンドウを作成します。complete イベントハンドラーは、呼び出されると、removeEventListener() 関数を使用して自身のイベントリスナーを削除します。

```
 var html = runtime.flash.html.HTMLLoader.createRootWindow(true);
html.addEventListener('complete', htmlCompleteListener);
function htmlCompleteListener()
{
    html.removeEventListener(complete, arguments.callee)
    // handler code..
}
html.load(new runtime.flash.net.URLRequest("second.html"));
```

また、不要なイベントリスナーを削除することで、システムのガベージコレクタは、これらのリスナーに関連付けられていたメモリを再生できます。

既存のイベントリスナーの確認

Adobe AIR 1.0 およびそれ以降

hasEventListener() メソッドを使用することで、あるオブジェクトに関してイベントリスナーが存在しているかどうかを確認することができます。

エラーイベントのリスナーを登録しない場合

Adobe AIR 1.0 およびそれ以降

例外（イベントではなく）は、ランタイムクラスでのエラー処理に関する主要なメカニズムです。しかし例外処理は、ファイルの読み込みなどの非同期操作に対しては機能しません。非同期操作の実行中にエラーが発生すると、ランタイムによりエラーイベントオブジェクトが送出されます。エラーイベントについてリスナーを作成しない場合は、AIR Debug Launcher により、エラーに関する情報を含むダイアログボックスが表示されます。

ほとんどのエラーイベントは ErrorEvent クラスに基づいていて、説明的なエラーメッセージを格納するために使用される text という名前のプロパティを備えています。例外は StatusEvent クラスで、text プロパティの代わりに level プロパティを備えています。level プロパティの値が error の場合、StatusEvent はエラーイベントと見なされます。

エラーイベントが発生しても、アプリケーションの実行は停止されません。エラーイベントは、AIR Debug Launcher のダイアログボックスとしてのみ表示されます。ランタイムで実行されているインストール済み AIR アプリケーションでは一切表示されません。

第 4 章：AIR HTML コンテナのスクリプト作成

Adobe AIR 1.0 およびそれ以降

HTMLLoader クラスは、Adobe® AIR® の HTML コンテンツのコンテナとして機能します。このクラスには、HTML コンテンツの動作や外観を制御するための多数のプロパティとメソッドが用意されています。また、HTML コンテンツの読み込みや操作、履歴管理などのタスクに使用するプロパティとメソッドについても、このクラスで定義します。

HTMLHost クラスは、HTMLLoader の一連のデフォルトの動作を定義します。HTMLLoader オブジェクトを作成しても、HTMLHost の実装は提供されません。そのため、HTML コンテンツによって、ウィンドウ位置やウィンドウタイトルの変更などのデフォルト動作がトリガーされても、何も起こりません。HTMLHost クラスを拡張することで、アプリケーションに必要な動作を定義できます。

HTMLHost のデフォルト実装は、AIR で作成された HTML ウィンドウに対して提供されます。HTMLHost のデフォルト実装を別の HTMLLoader オブジェクトに割り当てるには、そのオブジェクトの htmlHost プロパティを、defaultBehavior パラメーターを true に設定して作成した新しい HTMLHost オブジェクトを使用して設定します。

HTMLHost クラスを拡張するには、ActionScript を使用する必要があります。HTML ベースのアプリケーションでは、HTMLHost クラスの実装を含むコンパイル済みの SWF ファイルを読み込むことができます。window.htmlLoader プロパティを使用してホストクラスの実装を割り当てます。

```
<script src="HTMLHostLibrary.swf" type="application/x-shockwave-flash"></script>
<script>
    window.htmlLoader.htmlHost = new window.runtime.HTMLHostImplementation();
</script>
```

HTMLLoader オブジェクトの表示プロパティ

Adobe AIR 1.0 およびそれ以降

HTMLLoader オブジェクトは、Adobe® Flash® Player の Sprite クラスの表示プロパティを継承します。例えば、サイズを変更する、移動する、隠す、背景色を変更するなどの操作を行うことができます。それ以外にも、フィルター、マスク、拡大／縮小、回転などの高度なエフェクトを適用できます。エフェクトを適用する場合は、見やすさへの影響を考慮してください。適用するエフェクトによっては、HTML ページに読み込まれる SWF コンテンツおよび PDF コンテンツを表示できなくなります。

HTML ウィンドウには、HTML コンテンツをレンダリングする HTMLLoader オブジェクトが含まれます。このオブジェクトはウィンドウの領域の制約を受けるので、サイズ、位置、回転または拡大／縮小率を変更しても、意図した結果が得られない場合があります。

基本的な表示プロパティ

Adobe AIR 1.0 およびそれ以降

HTMLLoader の基本的な表示プロパティを使用すると、親表示オブジェクト内でのコントロールの位置指定、サイズの設定、コントロールの表示と非表示の切り替えを行うことができます。HTML ウィンドウの HTMLLoader オブジェクトについては、これらのプロパティを変更しないでください。

基本的な表示プロパティを次に示します。

プロパティ	注記
x、y	親コンテナ内でのオブジェクトの位置を指定します。
width、height	表示領域のサイズを変更します。
visible	オブジェクトとオブジェクトに含まれるコンテンツを表示するかどうかを指定します。

HTML ウィンドウ外では、HTMLLoader オブジェクトの width プロパティと height プロパティのデフォルト値は 0 です。読み込んだ HTML コンテンツを表示するには、まず width と height を設定する必要があります。HTML コンテンツは、HTMLLoader のサイズで描画され、コンテンツの HTML プロパティと CSS プロパティに従ってレイアウトされます。HTMLLoader のサイズを変更すると、コンテンツが再配置されます。

コンテンツを新しい HTMLLoader オブジェクトに読み込む場合（width は 0 に設定されたまま）、HTMLLoader の表示の width と height を contentWidth プロパティと contentHeight プロパティで設定することもできます。このテクニックは、HTML と CSS の配置ルールに従ってレイアウトした場合に、最小幅が十分に確保されるページに対しては有効です。ただし、HTMLLoader によって適正な幅が指定されていないと、ページが細長いレイアウトに配置される場合があります。

注意：他のタイプの表示オブジェクトについても一般的に言えることですが、HTMLLoader オブジェクトの width と height を変更しても、scaleX と scaleY の値は変化しません。

HTMLLoader コンテンツの透明度

Adobe AIR 1.0 およびそれ以降

HTMLLoader オブジェクトの paintsDefaultBackground プロパティは、HTMLLoader オブジェクトで不透明な背景を描画するかどうかを決定するもので、デフォルトで true になっています。paintsDefaultBackground を false にすると、背景が透明になります。表示オブジェクトコンテナまたは HTMLLoader オブジェクトの下に位置するその他の表示オブジェクトは、HTML コンテンツの前景エレメントの背後に表示されます。

body エレメントまたは HTML ドキュメントのその他のエレメントで背景色を指定（例えば style="background-color:gray" を使用して指定）すると、HTML の該当部分の背景は不透明になり、指定した背景色でレンダリングされます。HTMLLoader オブジェクトの opaqueBackground プロパティを設定し、paintsDefaultBackground を false にした場合は、opaqueBackground に設定した色が表示されます。

注意：透明な PNG 形式のグラフィックを使用して、HTML ドキュメント内のエレメントにアルファブレンドの背景を指定できます。HTML エレメントの opacity スタイルの設定はサポートされていません。

HTMLLoader コンテンツの拡大／縮小

Adobe AIR 1.0 およびそれ以降

HTMLLoader オブジェクトの拡大／縮小では、拡大／縮小率が 1.0 を超えないようにしてください。HTMLLoader コンテンツのテキストは特定の解像度でレンダリングされるので、HTMLLoader オブジェクトを拡大するとピクセルが目立つようになります。

HTML ページに SWF コンテンツまたは PDF コンテンツを読み込む場合の考慮事項

Adobe AIR 1.0 およびそれ以降

次の場合、HTMLLoader オブジェクトに読み込まれる SWF コンテンツおよび PDF コンテンツは表示されません。

- HTMLLoader オブジェクトを 1.0 以外の倍率に拡大／縮小した場合。

- HTMLLoader オブジェクトの alpha プロパティを 1.0 以外の値に設定した場合。
- HTMLLoader コンテンツを回転した場合。

問題のあるプロパティ設定を削除し、アクティブなフィルターを削除すると、コンテンツが再表示されます。

また、ランタイムでは、透明ウィンドウに PDF コンテンツを表示できません。オブジェクトまたは埋め込みタグの wmode パラメーターが opaque または transparent に設定されている場合、ランタイムでは HTML ページに埋め込まれた SWF コンテンツのみが表示されます。wmode のデフォルト値は window なので、wmode パラメーターを明示的に設定しない限り、SWF コンテンツは透明ウィンドウに表示されません。

注意：AIR 1.5.2 より前のバージョンでは、どの wmode 値が使用されていようと、HTML に埋め込まれた SWF が表示されることはありません。

HTMLLoader でこれらのタイプのメディアを読み込む方法について詳しくは、30 ページの「HTML への SWF コンテンツの埋め込み」および 264 ページの「AIR での PDF コンテンツの追加」を参照してください。

高度な表示プロパティ

Adobe AIR 1.0 およびそれ以降

HTMLLoader クラスは、特殊エフェクトに使用できるメソッドを複数継承しています。一般に、これらのエフェクトは、HTMLLoader の表示で使用する場合は制限がありますが、トランジションやその他の一時的なエフェクトに役立ちます。例えば、ユーザー入力を収集するダイアログウィンドウを表示する場合に、ユーザーがダイアログを閉じるまでメインウィンドウの表示をぼかすことができます。同様に、ウィンドウを閉じるときに表示をフェードアウトさせることもできます。

高度な表示プロパティを次に示します。

プロパティ	制限
alpha	HTML コンテンツが見えにくくなる場合があります。
filters	HTML ウィンドウでは、外部エフェクトはウィンドウの端でクリップされます。
graphics	graphics コマンドを使用して描画されるシェイプは、デフォルトの背景を含め、HTML コンテンツの下に表示されます。描画されたシェイプが表示されるようにするには、paintsDefaultBackground プロパティを false にする必要があります。
opaqueBackground	デフォルトの背景の色は変更されません。このカラーレイヤーが表示されるようにするには、paintsDefaultBackground プロパティを false にする必要があります。
rotation	矩形の HTMLLoader 領域の角が、ウィンドウの端でクリップされることがあります。HTML コンテンツに読み込まれた SWF コンテンツおよび PDF コンテンツは表示されません。
scaleX、scaleY	拡大／縮小率が 1 を超えると、レンダリングされた表示部分のピクセルが目立つようになることがあります。HTML コンテンツに読み込まれた SWF コンテンツおよび PDF コンテンツは表示されません。
transform	HTML コンテンツが見えにくくなる場合があります。HTML 表示がウィンドウの端でクリップされることがあります。回転、拡大／縮小または傾斜を伴う変形の場合、HTML コンテンツに読み込まれた SWF コンテンツおよび PDF コンテンツは表示されません。

次の例では、filters 配列を設定して HTML 表示全体をぼかす方法を示します。

```
var blur = new window.runtime.flash.filters.BlurFilter();
var filters = [blur];
window.htmlLoader.filters = filters;
```

注意：Sprite や BlurFilter などの表示オブジェクトクラスは、HTML ベースのアプリケーションでは一般には使用されません。これらのクラスは、『HTML 開発者用 Adobe AIR API リファレンスガイド』には記載されていません。これらのクラスのドキュメントについては、『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。

HTML 履歴リストへのアクセス

Adobe AIR 1.0 およびそれ以降

HTMLLoader オブジェクトに新しいページが読み込まれるとき、ランタイムでは、そのオブジェクトの履歴リストを維持します。履歴リストは、HTML ページの window.history オブジェクトに対応します。HTMLLoader クラスに含まれている次のプロパティとメソッドを使用すると、HTML 履歴リストを操作することができます。

クラスメンバー	説明
historyLength	後方と前方のエントリを含む、履歴リスト全体の長さです。
historyPosition	履歴リストでの現在の位置です。この位置より前の履歴アイテムは「戻る」ナビゲーションを表し、この位置より後の履歴アイテムは「進む」ナビゲーションを表します。
getHistoryAt()	履歴リスト内の指定した位置にある履歴エントリに対応する URLRequest オブジェクトを返します。
historyBack()	可能であれば、履歴リスト内を後方に進みます。
historyForward()	可能であれば、履歴リスト内を前方に進みます。
historyGo()	ブラウザーの履歴内を指定のステップ数だけ移動します。正の場合は前方に、負の場合は後方に移動します。0 の場合はページが再読み込みされます。終端を超える位置を指定すると、リストの最後に移動します。

履歴リスト内の項目は、HTMLHistoryItem 型のオブジェクトとして格納されます。HTMLHistoryItem クラスには、次のプロパティがあります。

プロパティ	説明
isPost	HTML ページに POST データが含まれている場合は true に設定されます。
originalUrl	リダイレクトされる前の HTML ページの元の URL です。
title	HTML ページのタイトルです。
url	HTML ページの URL です。

HTML コンテンツの読み込み時に使用するユーザーエージェントの設定

Adobe AIR 1.0 およびそれ以降

HTMLLoader クラスの userAgent プロパティを使用すると、HTMLLoader で使用するユーザーエージェントストリングを設定することができます。HTMLLoader オブジェクトの userAgent プロパティは、load() メソッドを呼び出す前に設定してください。HTMLLoader インスタンスに対してこのプロパティを設定した場合、load() メソッドに渡される URLRequest の userAgent プロパティは使用されません。

アプリケーションドメインですべての HTMLLoader オブジェクトが使用するデフォルトのユーザーエージェントストリングを設定するには、URLRequestDefaults.userAgent プロパティを設定します。静的な URLRequestDefaults プロパティは、HTMLLoader オブジェクトの load() メソッドで使用される URLRequests だけでなく、すべての URLRequest オブジェクトのデフォルトとして適用されます。HTMLLoader の userAgent プロパティの設定は、デフォルトの URLRequestDefaults.userAgent 設定をオーバーライドします。

HTMLLoader オブジェクトの userAgent プロパティと URLRequestDefaults.userAgent のどちらにもユーザーエージェント値を設定していない場合は、デフォルトの AIR ユーザーエージェント値が使用されます。このデフォルト値は、次の 2 つの例に示すように、ランタイムのオペレーティングシステム（Mac OS や Windows など）、ランタイムの言語およびランタイムのバージョンによって異なります。

- "Mozilla/5.0 (Mac OS; U; PPC Mac OS X; en) AppleWebKit/420+ (KHTML, like Gecko) AdobeAIR/1.0"
- "Mozilla/5.0 (Windows; U; en) AppleWebKit/420+ (KHTML, like Gecko) AdobeAIR/1.0"

HTML コンテンツで使用する文字エンコードの設定

Adobe AIR 1.0 およびそれ以降

HTML ページでは、次のように meta タグを含めることによって、使用する文字エンコードを指定できます。

```
meta http-equiv="content-type" content="text/html" charset="ISO-8859-1";
```

HTMLLoader オブジェクトの textEncodingOverride プロパティを設定して、特定の文字エンコードが使用されるようにページ設定をオーバーライドします。

```
window.htmlLoader.textEncodingOverride = "ISO-8859-1";
```

HTML ページで設定が指定されていない場合に使用する HTMLLoader コンテンツの文字エンコードは、HTMLLoader オブジェクトの textEncodingFallback プロパティで指定します。

```
window.htmlLoader.textEncodingFallback = "ISO-8859-1";
```

textEncodingOverride プロパティは、HTML ページでの設定をオーバーライドします。また、textEncodingOverride プロパティと HTML ページでの設定は、textEncodingFallback プロパティをオーバーライドします。

textEncodingOverride プロパティまたは textEncodingFallback プロパティを設定してから、HTML コンテンツを読み込んでください。

HTML コンテンツのブラウザー形式のユーザーインターフェイスの定義

Adobe AIR 1.0 およびそれ以降

JavaScript には、HTML コンテンツを表示するウィンドウを制御するための API がいくつか用意されています。AIR では、これらの API はカスタム HTMLHost クラスを実装してオーバーライドできます。

重要：HTMLHost クラスのカスタム実装を作成するには、ActionScript を使用する必要があります。HTML ページでカスタム実装を含むコンパイル済みの ActionScript（SWF）ファイルを読み込んで使用できます。HTML への ActionScript ライブラリの読み込みについて詳しくは、33 ページの「HTML ページ内での ActionScript ライブラリの使用」を参照してください。

HTMLHost クラスの拡張について

Adobe AIR 1.0 およびそれ以降

AIR HTMLHost クラスは、次の JavaScript プロパティとメソッドを制御します。

- window.status
- window.document.title
- window.location
- window.blur()
- window.close()
- window.focus()
- window.moveBy()
- window.moveTo()
- window.open()
- window.resizeBy()
- window.resizeTo()

new HTMLLoader() を使用して HTMLLoader オブジェクトを作成した場合、上記の JavaScript プロパティまたはメソッドは有効になりません。HTMLHost クラスは、これらの JavaScript API のデフォルトのブラウザー形式の実装を提供します。HTMLHost クラスを拡張して、動作をカスタマイズすることもできます。デフォルトの動作をサポートする HTMLHost オブジェクトを作成するには、HTMLHost コンストラクターの defaultBehaviors パラメーターを true に設定します。

```
var defaultHost = new HTMLHost(true);
```

AIR で、HTMLLoader クラスの createRootWindow() メソッドを使用して HTML ウィンドウを作成する場合は、デフォルトの動作をサポートする HTMLHost インスタンスが自動的に割り当てられます。ホストオブジェクトの動作を変更するには、別の HTMLHost 実装を HTMLLoader の htmlHost プロパティに割り当てます。また、このプロパティに null を割り当てて、機能全体を無効にすることもできます。

注意：AIR では、HTML ベースの AIR アプリケーション用に作成された初期ウィンドウと、JavaScript の window.open() メソッドのデフォルト実装によって作成されたすべてのウィンドウに、デフォルトの HTMLHost オブジェクトが割り当てられます。

例：HTMLHost クラスの拡張

Adobe AIR 1.0 およびそれ以降

次の例では、HTMLHost クラスを拡張して、HTMLLoader オブジェクトがユーザーインターフェイスに及ぼす影響をカスタマイズする方法を示します。

Flex の例：

1. HTMLHost クラスを拡張するクラス（サブクラス）を作成します。
2. 新しいクラスのメソッドをオーバーライドして、ユーザーインターフェイス関連の設定の変更を処理します。例えば、次の CustomHost クラスは、window.open() の呼び出しと window.document.title の変更の動作を定義します。window.open() の呼び出しによって新しいウィンドウで HTML ページを開き、window.document.title の変更（HTML ページの <title> エレメントの設定を含む）によってそのウィンドウのタイトルを設定します。

```
package
{
    import flash.html.*;
    import flash.display.StageScaleMode;
    import flash.display.NativeWindow;
    import flash.display.NativeWindowInitOptions;

    public class CustomHost extends HTMLHost
    {
        import flash.html.*;
        override public function
            createWindow(windowCreateOptions:HTMLWindowCreateOptions):HTMLLoader
        {
            var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
            var bounds:Rectangle = new Rectangle(windowCreateOptions.x,
                                                 windowCreateOptions.y,
                                                 windowCreateOptions.width,
                                                 windowCreateOptions.height);
            var htmlControl:HTMLLoader = HTMLLoader.createRootWindow(true, initOptions,
                                                windowCreateOptions.scrollBarsVisible, bounds);
            htmlControl.htmlHost = new HTMLHostImplementation();
            if(windowCreateOptions.fullscreen){
                htmlControl.stage.displayState =
                    StageDisplayState.FULL_SCREEN_INTERACTIVE;
            }
            return htmlControl;
        }
        override public function updateTitle(title:String):void
        {
            htmlLoader.stage.nativeWindow.title = title;
        }
    }
}
```

3 HTMLLoader を含むコードで（HTMLHost の新しいサブクラスのコードではなく）、新しいクラスのオブジェクトを作成します。その新しいオブジェクトを HTMLLoader の htmlHost プロパティに割り当てます。次の Flex コードでは、前述の手順で定義した CustomHost クラスを使用します。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    applicationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.html.HTMLLoader;
            import CustomHost;
            private function init():void
            {
                var html:HTMLLoader = new HTMLLoader();
                html.width = container.width;
                html.height = container.height;
                var urlReq:URLRequest = new URLRequest("Test.html");
                html.htmlHost = new CustomHost();
                html.load(urlReq);
                container.addChild(html);
            }
        ]]>
    </mx:Script>
    <mx:UIComponent id="container" width="100%" height="100%"/>
</mx:WindowedApplication>
```

ここで示したコードをテストするには、次のコンテンツを含む HTML ファイルをアプリケーションディレクトリに含めます。

```
<html>
    <head>
        <title>Test</title>
    </head>
    <script>
        function openWindow()
        {
            window.runtime.trace("in");
            document.title = "foo"
            window.open('Test.html');
            window.runtime.trace("out");
        }
    </script>
    <body>
        <a href="#" onclick="openWindow()">window.open('Test.html')</a>
    </body>
</html>
```

Flash Professional の例：

1. AIR の Flash ファイルを作成します。ドキュメントクラスを CustomHostExample に設定し、CustomHostExample.fla としてファイルを保存します。
2. HTMLHost クラスを拡張するクラス（サブクラス）を含む CustomHost.as という ActionScript ファイルを作成します。このクラスは、新しいクラスの特定のメソッドをオーバーライドして、ユーザーインターフェイス関連の設定の変更を処理します。例えば、次の CustomHost クラスは、window.open() の呼び出しと window.document.title の変更の動作を定義します。window.open() メソッドの呼び出しによって新しいウィンドウで HTML ページを開き、window.document.title プロパティの変更（HTML ページの <title> エレメントの設定を含む）によってそのウィンドウのタイトルを設定します。

```
package
{
    import flash.display.StageScaleMode;
    import flash.display.NativeWindow;
    import flash.display.NativeWindowInitOptions;
    import flash.events.Event;
    import flash.events.NativeWindowBoundsEvent;
    import flash.geom.Rectangle;
    import flash.html.HTMLLoader;
    import flash.html.HTMLHost;
    import flash.html.HTMLWindowCreateOptions;
    import flash.text.TextField;

    public class CustomHost extends HTMLHost
    {
        public var statusField:TextField;

        public function CustomHost(defaultBehaviors:Boolean=true)
        {
            super(defaultBehaviors);
        }
        override public function windowClose():void
        {
            htmlLoader.stage.nativeWindow.close();
        }
        override public function createWindow(
                                windowCreateOptions:HTMLWindowCreateOptions ):HTMLLoader
        {
            var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
```

```
            var bounds:Rectangle = new Rectangle(windowCreateOptions.x,
                                                 windowCreateOptions.y,
                                                 windowCreateOptions.width,
                                                 windowCreateOptions.height);
            var htmlControl:HTMLLoader = HTMLLoader.createRootWindow(true, initOptions,
                                                windowCreateOptions.scrollBarsVisible, bounds);
            htmlControl.htmlHost = new HTMLHostImplementation();
            if(windowCreateOptions.fullscreen){
                htmlControl.stage.displayState =
                    StageDisplayState.FULL_SCREEN_INTERACTIVE;
            }
            return htmlControl;
        }
        override public function updateLocation(locationURL:String):void
        {
            trace(locationURL);
        }
        override public function set windowRect(value:Rectangle):void
        {
            htmlLoader.stage.nativeWindow.bounds = value;
        }
        override public function updateStatus(status:String):void
        {
            statusField.text = status;
            trace(status);
        }
        override public function updateTitle(title:String):void
        {
            htmlLoader.stage.nativeWindow.title = title + "- Example Application";
        }
        override public function windowBlur():void
        {
            htmlLoader.alpha = 0.5;
        }
        override public function windowFocus():void
        {
            htmlLoader.alpha = 1;
        }
    }
}
```

3 アプリケーションのドキュメントクラスを格納する CustomHostExample.as という別の ActionScript ファイルを作成します。このクラスは、HTMLLoader オブジェクトを作成し、そのホストプロパティを前述の手順で定義した CustomHost クラスのインスタンスに設定します。

```
package
{
    import flash.display.Sprite;
    import flash.html.HTMLLoader;
    import flash.net.URLRequest;
    import flash.text.TextField;

    public class CustomHostExample extends Sprite
    {
        function CustomHostExample():void
        {
            var html:HTMLLoader = new HTMLLoader();
            html.width = 550;
            html.height = 380;
            var host:CustomHost = new CustomHost();
            html.htmlHost = host;

            var urlReq:URLRequest = new URLRequest("Test.html");
            html.load(urlReq);

            addChild(html);

            var statusTxt:TextField = new TextField();
            statusTxt.y = 380;
            statusTxt.height = 20;
            statusTxt.width = 550;
            statusTxt.background = true;
            statusTxt.backgroundColor = 0xEEEEEEEE;
            addChild(statusTxt);

            host.statusField = statusTxt;
        }
    }
}
```

ここで示したコードをテストするには、次のコンテンツを含む HTML ファイルをアプリケーションディレクトリに含めます。

```
<html>
     <head>
     <title>Test</title>
     <script>
     function openWindow()
     {
     document.title = "Test"
     window.open('Test.html');
     }
     </script>
     </head>
     <body bgColor="#EEEEEE">
     <a href="#" onclick="window.open('Test.html')">window.open('Test.html')</a>
     <br/><a href="#" onclick="window.document.location='http://www.adobe.com'">
     window.document.location = 'http://www.adobe.com'</a>
     <br/><a href="#" onclick="window.moveBy(6, 12)">moveBy(6, 12)</a>
     <br/><a href="#" onclick="window.close()">window.close()</a>
     <br/><a href="#" onclick="window.blur()">window.blur()</a>
     <br/><a href="#" onclick="window.focus()">window.focus()</a>
     <br/><a href="#" onclick="window.status = new Date().toString()">window.status=new
Date().toString()</a>
     </body>
</html>
```

1 HTMLHostImplementation.as などの ActionScript ファイルを作成します。

2 このファイルで、HTMLHost クラスを拡張するクラスを定義します。

3 新しいクラスのメソッドをオーバーライドして、ユーザーインターフェイス関連の設定の変更を処理します。例えば、次の CustomHost クラスは、window.open() の呼び出しと window.document.title の変更の動作を定義します。window.open() の呼び出しによって新しいウィンドウで HTML ページを開き、window.document.title の変更（HTML ページの <title> エレメントの設定を含む）によってそのウィンドウのタイトルを設定します。

```
package {
    import flash.html.HTMLHost;
    import flash.html.HTMLLoader;
    import flash.html.HTMLWindowCreateOptions;
    import flash.geom.Rectangle;
    import flash.display.NativeWindowInitOptions;
    import flash.display.StageDisplayState;

    public class HTMLHostImplementation extends HTMLHost{
        public function HTMLHostImplementation(defaultBehaviors:Boolean = true):void{
            super(defaultBehaviors);
        }

        override public function updateTitle(title:String):void{
            htmlLoader.stage.nativeWindow.title = title + " - New Host";
        }

        override public function createWindow(windowCreateOptions:HTMLWindowCreateOptions):HTMLLoader{
            var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
            var bounds:Rectangle = new Rectangle(windowCreateOptions.x,
                                                 windowCreateOptions.y,
                                                 windowCreateOptions.width,
                                                 windowCreateOptions.height);

            var htmlControl:HTMLLoader = HTMLLoader.createRootWindow(true, initOptions,
                                         windowCreateOptions.scrollBarsVisible, bounds);

            htmlControl.htmlHost = new HTMLHostImplementation();

            if(windowCreateOptions.fullscreen){
                htmlControl.stage.displayState =
                    StageDisplayState.FULL_SCREEN_INTERACTIVE;
            }

            return htmlControl;
        }
    }
}
```

4 acompc コンポーネントコンパイラーを使用して、このクラスを SWF ファイルにコンパイルします。

```
acompc -source-path . -include-classes HTMLHostImplementation -output Host.zip
```

注意：acompc コンパイラーは Flex SDK に付属しています（AIR SDK は SWF ファイルのコンパイルを通常必要としない HTML 開発者を対象としているため、AIR SDK には付属していません）。acompc の使用方法については、「Using compc, the component compiler」で説明されています。

5 Host.zip ファイルを開き、その中の Library.swf ファイルを展開します。

6 Library.swf の名前を HTMLHostLibrary.swf に変更します。この SWF ファイルは、HTML ページに読み込むライブラリです。

7 <script> タグを使用して、このライブラリを HTML ページに読み込みます。

```
<script src="HTMLHostLibrary.swf" type="application/x-shockwave-flash"></script>
```

8 HTMLHost 実装の新しいインスタンスをページの HTMLLoader オブジェクトに割り当てます。

```
    window.htmlLoader.htmlHost = new window.runtime.HTMLHostImplementation();
```

次の HTML ページは、HTMLHost 実装を読み込んで使用する方法を示しています。ボタンをクリックして新しいフルスクリーンウィンドウを開くことで、updateTitle() と createWindow() の実装をテストできます。

```
<html>
    <head>
    <title>HTMLHost Example</title>
    <script src="HTMLHostLibrary.swf" type="application/x-shockwave-flash"></script>
    <script language="javascript">
        window.htmlLoader.htmlHost = new window.runtime.HTMLHostImplementation();

        function test(){
            window.open('child.html', 'Child', 'fullscreen');
        }
    </script>
    </head>
    <body>
        <button onClick="test()">Create Window</button>
    </body>
</html>
```

この例を実行するには、child.html という HTML ファイルをアプリケーションディレクトリに含めます。

window.location プロパティの変更の処理

Adobe AIR 1.0 およびそれ以降

locationChange() メソッドをオーバーライドして、HTML ページの URL の変更を処理します。locationChange() メソッドは、ページの JavaScript で window.location の値を変更するときに呼び出されます。次の例では、要求された URL を読み込んでいます。

```
override public function updateLocation(locationURL:String):void
{
    htmlLoader.load(new URLRequest(locationURL));
}
```

注意：HTMLHost オブジェクトの htmlLoader プロパティを使用して、現在の HTMLLoader オブジェクトを参照できます。

window.moveBy()、window.moveTo()、window.resizeTo()、window.resizeBy() の JavaScript 呼び出しの処理

Adobe AIR 1.0 およびそれ以降

set windowRect() メソッドをオーバーライドして、HTML コンテンツの境界の変更を処理します。set windowRect() メソッドは、ページの JavaScript から window.moveBy()、window.moveTo()、window.resizeTo() または window.resizeBy() を呼び出すときに呼び出されます。次の例では、デスクトップウィンドウの境界を更新します。

```
override public function set windowRect(value:Rectangle):void
{
    htmlLoader.stage.nativeWindow.bounds = value;
}
```

window.open() の JavaScript 呼び出しの処理

Adobe AIR 1.0 およびそれ以降

createWindow() メソッドをオーバーライドして、window.open() の JavaScript 呼び出しを処理します。createWindow() メソッドの実装は、新しい HTMLLoader オブジェクトを作成して返します。通常、HTMLLoader は新しいウィンドウに表示しますが、ウィンドウの作成は必須ではありません。

次の例では、createWindow() 関数を HTMLLoader.createRootWindow() で実装し、ウィンドウと HTMLLoader オブジェクトの両方を作成する方法を示します。NativeWindow オブジェクトを別個に作成し、HTMLLoader をウィンドウステージに追加することもできます。

```
override public function createWindow(windowCreateOptions:HTMLWindowCreateOptions):HTMLLoader{
    var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
    var bounds:Rectangle = new Rectangle(windowCreateOptions.x, windowCreateOptions.y,
                                         windowCreateOptions.width, windowCreateOptions.height);
    var htmlControl:HTMLLoader = HTMLLoader.createRootWindow(true, initOptions,
                                     windowCreateOptions.scrollBarsVisible, bounds);
    htmlControl.htmlHost = new HTMLHostImplementation();
    if(windowCreateOptions.fullscreen){
        htmlControl.stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
    }
    return htmlControl;
}
```

注意：この例では、window.open() で作成した新しいウィンドウにカスタム HTMLHost 実装を割り当てます。必要に応じて、別の実装を使用したり、新しいウィンドウの htmlHost プロパティを null に設定したりすることもできます。

createWindow() メソッドにパラメーターとして渡されるオブジェクトは HTMLWindowCreateOptions オブジェクトです。HTMLWindowCreateOptions クラスには、features パラメーターストリング（window.open() の呼び出しで使用）の設定値を知らせるプロパティがあります。

HTMLWindowCreateOptions プロパティ	window.open() の JavaScript 呼び出しにおける features ストリングの対応する設定
fullscreen	fullscreen
height	height
locationBarVisible	location
menuBarVisible	menubar
resizeable	resizable
scrollBarsVisible	scrollbars
statusBarVisible	status
toolBarVisible	toolbar
width	width
x	left または screenX
y	top または screenY

HTMLLoader クラスは、features ストリングで指定できる機能の一部は実装していません。スクロールバー、ロケーションバー、メニューバー、ステータスバーおよびツールバーは、必要に応じてアプリケーションで用意する必要があります。

JavaScript の window.open() メソッドのその他の引数は、システムによって処理されます。createWindow() の実装では、HTMLLoader オブジェクトにコンテンツを読み込んだり、ウィンドウタイトルを設定したりしないでください。

window.close() の JavaScript 呼び出しの処理

Adobe AIR 1.0 およびそれ以降

windowClose() をオーバーライドして、window.close() メソッドの JavaScript 呼び出しを処理します。次の例では、window.close() メソッドが呼び出されたときにデスクトップウィンドウを閉じます。

```
override public function windowClose():void
{
    htmlLoader.stage.nativeWindow.close();
}
```

window.close() の JavaScript 呼び出しでは、必ずしもコンテナウィンドウを閉じる必要はありません。例えば、次のコードのように、ウィンドウ（他のコンテンツを含んでいるウィンドウ）を開いたまま、表示リストから HTMLLoader を削除することもできます。

```
override public function windowClose():void
{
    htmlLoader.parent.removeChild(htmlLoader);
}
```

window.status プロパティの変更の処理

Adobe AIR 1.0 およびそれ以降

updateStatus() メソッドをオーバーライドして、JavaScript による window.status の値の変更を処理します。次の例では、ステータス値をトレースします。

```
override public function updateStatus(status:String):void
{
    trace(status);
}
```

要求されたステータスが、updateStatus() メソッドにストリングとして渡されます。

HTMLLoader オブジェクトにステータスバーはありません。

window.document.title プロパティの変更の処理

Adobe AIR 1.0 およびそれ以降

updateTitle() メソッドをオーバーライドして、JavaScript による window.document.title の値の変更を処理します。次の例では、ウィンドウタイトルを変更し、「Sample」というストリングをタイトルに追加します。

```
override public function updateTitle(title:String):void
{
    htmlLoader.stage.nativeWindow.title = title + " - Sample";
}
```

document.title が HTML ページで設定されると、要求されたタイトルが updateTitle() メソッドにストリングとして渡されます。

document.title の変更では、必ずしも HTMLLoader オブジェクトを含むウィンドウのタイトルを変更する必要はありません。例えば、テキストフィールドなど、別のインターフェイスエレメントを変更することもできます。

window.blur() と window.focus() の JavaScript 呼び出しの処理

Adobe AIR 1.0 およびそれ以降

windowBlur() メソッドと windowFocus() メソッドをオーバーライドして、window.blur() と window.focus() の JavaScript 呼び出しを処理します。この例を次に示します。

```
 override public function windowBlur():void
{
    htmlLoader.alpha = 0.5;
}
override public function windowFocus():void
{
    htmlLoader.alpha = 1.0;
    NativeApplication.nativeApplication.activate(htmlLoader.stage.nativeWindow);
}
```

注意：AIR には、ウィンドウまたはアプリケーションを非アクティブ化するための API はありません。

スクロールする HTML コンテンツを含むウィンドウの作成

Adobe AIR 1.0 およびそれ以降

HTMLLoader クラスには HTMLLoader.createRootWindow() という静的メソッドがあり、このメソッドを使用すると、HTMLLoader オブジェクトを含む新しいウィンドウ（NativeWindow オブジェクトによって表されるウィンドウ）を開き、そのウィンドウのユーザーインターフェイス設定を定義することができます。このメソッドは、ユーザーインターフェイスの定義に使用できる 4 つのパラメーターを受け取ります。

パラメーター	説明
visible	初期状態でウィンドウを表示する（true）か、非表示にする（false）かを指定する Boolean 値です。
windowInitOptions	NativeWindowInitOptions オブジェクトです。NativeWindowInitOptions クラスは、NativeWindow オブジェクトの初期化オプションを定義します。例えば、ウィンドウの最小化、最大化またはサイズ変更が可能かどうか、ウィンドウでシステムクロムまたはカスタムクロムを使用するか、ウィンドウを透明にするかどうか（システムクロムを使用しないウィンドウの場合）や、ウィンドウのタイプなどを指定するオプションがあります。
scrollBarsVisible	スクロールバーを表示する（true）か、表示しない（false）かを指定します。
bounds	新しいウィンドウの位置とサイズを定義する Rectangle オブジェクトです。

例えば、次のコードでは、HTMLLoader.createRootWindow() メソッドを使用して、スクロールバーを使用する HTMLLoader コンテンツを含むウィンドウを作成します。

```
var initOptions = new air.NativeWindowInitOptions();
var bounds = new air.Rectangle(10, 10, 600, 400);
var html2 = air.HTMLLoader.createRootWindow(true, initOptions, true, bounds);
var urlReq2 = new air.URLRequest("http://www.example.com");
html2.load(urlReq2);
html2.stage.nativeWindow.activate();
```

注意：JavaScript で createRootWindow() を直接呼び出すことによって作成されたウィンドウは、そのウィンドウを開いた HTML ウィンドウから独立したままになります。例えば、JavaScript ウィンドウの opener プロパティと parent プロパティは null になります。ただし、createRootWindow() を、HTMLHost の createWindow() メソッドをオーバーライドして createRootWindow() を呼び出すことによって間接的に呼び出した場合には、opener と parent は、そのウィンドウを開いた HTML ウィンドウを参照します。

第 5 章：ベクターの操作

Adobe AIR 1.5 およびそれ以降

Vector インスタンスは、型指定配列です。つまり、Vector インスタンスのすべての要素は必ず同一のデータ型を含みます。NativeProcess や NetworkInfo などの一部の AIR API はプロパティまたはメソッドのデータ型として Vector を使用します。

Adobe AIR で実行している JavaScript コードでは、Vector クラスは air.Vector（AIRAliases.js ファイル内）として参照されます。

ベクターの基礎

Adobe AIR 1.5 およびそれ以降

Vector 変数を宣言するか、または Vector オブジェクトをインスタンス化するときは、ベクターが含むことのできるオブジェクトのデータ型を明示的に指定します。指定したデータ型は、ベクターのベース型と呼ばれます。実行時に、ベクターの値を設定または取得するすべてのコードが検証されます。追加または取得されるオブジェクトのデータ型がベクターのベース型と一致しない場合は、エラーが発生します。

データ型の制限に加えて、Vector クラスには Array クラスと区別される次のようなその他の制限があります。

- ベクターは高密度の配列です。Array オブジェクトは、位置 1 ～ 6 に値を含まない場合でも、インデックス 0 および 7 に値を含む場合があります。ただし、ベクターはインデックスに値（つまり、null）を含む必要があります。
- オプションでベクターを固定長にすることができます。固定長の場合、ベクターが含む要素数を変更することはできません。
- Vector の要素へのアクセスは範囲チェックが行われます。最後の要素（length - 1）より長いインデックスから値を読み取ることはできません。現在の最後のインデックスを越えて値を設定することはできません（つまり、既存のインデックスまたはインデックス [length] でのみ値を設定することができます）。

このような制限の結果として、Vector インスタンスには、エレメントがすべて単一クラスのインスタンスとなる Array インスタンスにはない、次の 3 つの主要な利点があります。

- パフォーマンス：Array インスタンスを使用するよりも、Vector インスタンスを使用する方が、配列要素のアクセスと繰り返しが高速になります。
- 型の安全性：このようなエラーの例としては、ベクターに不正なデータ型の値を割り当てたり、ベクターから値を読み取るときに誤ったデータ型を要求したりすることが挙げられます。実行時には、Vector オブジェクトにデータを追加するときや、このオブジェクトからデータを読み取るときにデータ型が検証されます。
- 信頼性：実行時に範囲を検証（または固定長を検証）することにより、配列よりも信頼性が大幅に向上します。

これらの追加の制限および利点以外では、Vector クラスは Array クラスに非常によく似ています。Vector オブジェクトのプロパティとメソッドは、配列のプロパティとメソッドと似ており、通常は同じです。多くの場合、すべての要素が同じデータ型を持つ配列を使用するときは、Vector インスタンスが推奨されます。

重要な概念と用語

次の参照リストに、配列およびベクター処理ルーチンのプログラミングで使用される重要な用語を示します。

配列アクセス（[]）演算子 配列要素を一意に識別するインデックスまたはキーを囲む一組の角括弧。このシンタックスはベクター変数名の後に使用され、ベクター全体ではなくベクターの 1 つの要素を指定します。

ベース型 Vector インスタンスを格納できるオブジェクトのデータ型。

エレメント ベクターの 1 つのアイテム。

索引 インデックス配列内の単一の要素を識別するための「番号アドレス」。

T ベース型が何であるかに関係なく、Vector インスタンスのベース型を表すこのマニュアルで使用する標準の表記規則。T 表記規則は、型パラメーターの説明に示すように、クラス名を表すために使用されます (「T」は「データ型」の「型」を表します)。

型パラメーター Vector クラス名で使用され、ベクターのベース型を指定するシンタックス（Vector クラスが格納するオブジェクトのデータ型)。シンタックスは、ピリオド（.）とその後に続く山括弧（<>）で囲まれたデータ型名で構成されます。これらをまとめると、Vector.<T> のようになります。本マニュアルでは、型パラメーターで指定されるクラスは、通常、T として表されます。

Vector すべての要素が同じデータ型のインスタンスである配列の型。

ベクターの作成

AIR 1.5 以降

air.Vector["<T>"]() コンストラクターを呼び出して Vector インスタンスを作成します。このコンストラクターを呼び出すときに、Vector 変数のベース型を指定します。ベクターのベース型は、型パラメーターシンタックスを使用して指定します。コードでは、Vector という語の直後に型パラメーターが置かれます。型パラメーターは左角括弧、山括弧（<>）で囲まれた基本クラス名の文字列、右角括弧の順に構成されます。このシンタックスの例は次のとおりです。

```
var v = new air.Vector["<String>"]();
```

この例では、変数 v は String オブジェクトのベクターとして宣言されています。つまり、これは、String インスタンスのみを保持できるインデックス配列を表します。

引数を指定しないで air.Vector["<T>"]() コンストラクターを使用すると、空の Vector インスタンスが作成されます。ベクターの length プロパティを検証することによって、Vector が空であることをテストできます。例えば、次のコードは、Vector["<T>"]() コンストラクターを引数なしで呼び出しています。

```
var names = new air.Vector["<String>"]();
air.trace(names.length); // output: 0
```

ベクターが最初に必要とする要素の数を事前に把握している場合、ベクター内の要素数を事前に定義できます。特定の数の要素を含むベクターを作成するには、1 番目のパラメーター（length パラメーター）として要素数を渡します。Vector 要素は空にできないので、要素にはベース型のインスタンスが入力されます。ベース型が null 値を許可する参照型である場合、すべての要素は null を含みます。その他の場合、要素はすべてクラスのデフォルト値を含みます。例えば、Number 変数は null にできません。その結果、次のコードでは、3 つの要素を含みそれぞれの要素のデフォルトの Numbe の値が NaN である ages というベクターが作成されます。

```
var ages = new air.Vector["<Number>"](3);
air.trace(ages); // output: NaN, NaN, NaN
```

Vector["<T>"]() コンストラクターを使用すると、第 2 パラメーター（fixed パラメーター）に true を渡すことによって固定長のベクターを作成することもできます。この場合、指定の数の要素を含むベクターが作成され、要素数は変更できません。ただし、固定長のベクターの要素の値を変更することはできます。

AIR ランタイムオブジェクト（window.runtime オブジェクトで定義されたクラス）のベクターを作成する場合、Vector コンストラクターを呼び出すときに ActionScript 3.0 のクラスの完全修飾名を参照します。例えば、次に示すコードは File オブジェクトのベクターを作成します。

```
var files = new air.Vector["flash.filesystem.File"](3);
```

ベクターへの要素の挿入

Adobe AIR 1.5 およびそれ以降

ベクターに要素を追加する最も基本的な方法は、配列アクセス（[]）演算子を使用することです。

```
songTitles[5] = "Happy Birthday";
```

Vector にまだそのインデックスの要素がない場合は、インデックスが作成され、そこに値が格納されます。

Vector オブジェクトでは、既存のインデックスまたは次に使用可能なインデックスにのみ値を割り当てることができます。次に使用可能なインデックスは、Vector オブジェクトの length プロパティに相当します。Vector オブジェクトに新しい要素を追加する最も安全な方法は、次のようなコードを使用する方法です。

```
myVector[myVector.length] = valueToAdd;
```

配列と同様に、Vector クラスの push()、unshift()、splice() の 3 つのメソッドを使用して、ベクターに要素を挿入することができます。

注意：Vector オブジェクトの fixed プロパティが true である場合は、ベクターの要素数の合計を変更することはできません。push() メソッドまたは他の方法で固定長のベクターに新しい要素を追加しようとすると、エラーが発生します。

値の取得とベクター要素の削除

Adobe AIR 1.5 およびそれ以降

ベクターから要素の値を取得する最も単純な方法は、配列アクセス（[]）演算子を使用する方法です。ベクター要素の値を取得するには、代入ステートメントの右側に Vector オブジェクトの名前とインデックス番号を使用します。

```
var myFavoriteSong = songTitles[3];
```

要素を含まないインデックスを使用して、ベクターから値の取得を試みることは可能です。その場合、ベクターは RangeError の例外をスローします。

要素を削除する場合は、Array および Vector クラスにある pop()、shift()、および splice() の 3 つのメソッドを使用できます。

```
var vegetables = new air.Vector["<String>"];
vegetables.push("spinach");
vegetables.push("green pepper");
vegetables.push("cilantro");
vegetables.push("onion");
var spliced = vegetables.splice(2, 2);
air.trace(spliced); // output: spinach,green pepper
```

length プロパティを使用してベクターを切り詰めることができます。

ベクターの length プロパティをベクターの現在の長さ未満に設定すると、ベクターは切り詰められます。length から 1 を引いた新しい値よりも大きいインデックス番号に格納されているすべての要素は削除されます。

注意：Vector オブジェクトの fixed プロパティが true である場合は、ベクターの要素数の合計を変更することはできません。ここで説明したテクニックを使用して固定長のベクターから要素を削除したり、固定長のベクターを切り詰めたりしようとすると、エラーが発生します。

Vector オブジェクトのプロパティとメソッド

Adobe AIR 1.5 およびそれ以降

Array オブジェクトの同一のメソッドとプロパティの多くは Vector オブジェクトに使用できます。例えば、reverse() メソッドを呼び出してベクターの要素順を変更することができます。また、sort() メソッドを呼び出してベクターの要素を並べ替えることができます。ただし、Vector クラスには、sortOn() メソッドはありません。

サポートされているプロパティとメソッドについて詳しくは、『HTML 開発者用 Adobe AIR API リファレンスガイド』の Vector クラスのドキュメントを参照してください。

ベクターを必要とする AIR API の使用例

Adobe AIR 1.5 およびそれ以降

一部の Adobe AIR ランタイムクラスは、プロパティまたはメソッドの戻り値としてベクターを使用します。例えば、NetworkInfo クラスの findInterfaces() メソッドは、NetworkInterface オブジェクトの配列を返します。arguments プロパティの NativeProcessStartupInfo クラスは文字列のベクターです。

Vector オブジェクトを返す AIR API へのアクセス

Adobe AIR 2.0 およびそれ以降

NetworkInfo クラスの findInterfaces() メソッドは、NetworkInterface オブジェクトの配列を返します。例えば、次のコードはコンピューターのネットワークインターフェイスの一覧です。

```
var netInfo = air.NetworkInfo;
var interfaces = netInfo.findInterfaces();
for (i = 0; i < interfaces.length; i++)
{
    air.trace(interfaces[i].name];
    air.trace(" hardware address: ", interface.hardwareAddress);
}
```

配列での繰り返し処理と同様に、NetworkInfo オブジェクトのベクターで繰り返し処理を実行します。for ループと角括弧を使用して、ベクターのインデックス要素にアクセスします。

NetworkInterface オブジェクトの interfaces プロパティは InterfaceAddress オブジェクトのベクターです。次のコードは上記の例を拡張したものです。各ネットワークインターフェイスのインターフェイスアドレスを列挙する関数が追加されています。

```
var netInfo = air.NetworkInfo;
var interfaces = netInfo.findInterfaces();
for (i = 0; i < interfaces.length; i++)
{
    air.trace(interfaces[i].name];
    air.trace(" hardware address: ", interface.hardwareAddress);
    air.trace(" addresses: ", traceAddresses(i);
}

function traceAddresses(i)
{
    returnString = new String();
    for (j = 0; j < interfaces[i].addresses.length; j++)
        returnString += interfaces[i],addresses[j].address + " ";
    }
}
```

ベクターがある AIR API の設定

Adobe AIR 2.0 およびそれ以降

arguments プロパティの NativeProcessStartupInfo クラスは文字列のベクターです。このプロパティを設定するには、air.Vector() コンストラクターを使用して文字列のベクターを作成します。push() メソッドを使用して、ベクターに文字列を追加することができます。

```
var arguments = new air.Vector["<String>"]();

arguments.push("test");
arguments.push("44");

var startupInfo = new air.NativeProcessStartupInfo();
startupInfo.arguments = arguments;
startupInfo.executable = File.applicationDirectory.resolvePath("myApplication.exe");

process = new air.NativeProcess();
process.start(startupInfo);
```

ネイティブプロセス API の使用について詳しくは、『ネットワーキングと通信』の「ネイティブプロセスとの通信」を参照してください。

第 6 章：AIR のセキュリティ

Adobe AIR 1.0 およびそれ以降

AIR のセキュリティの基礎

Adobe AIR 1.0 およびそれ以降

AIR アプリケーションは、ネイティブアプリケーションと同じセキュリティ制限で実行されます。一般的に、AIR アプリケーションでは、ネイティブアプリケーションと同様に、ファイルの読み取りや書き込み、アプリケーションの起動、画面への描画、ネットワークとの通信など、オペレーティングシステム機能に広範囲にアクセスできます。ネイティブアプリケーションに適用されるオペレーティングシステムの制限（ユーザー固有の権限など）は、AIR アプリケーションにも同等に適用されます。

Adobe® AIR® のセキュリティモデルは Adobe® Flash® Player のセキュリティモデルの進化形ですが、セキュリティコントラクトは、ブラウザーのコンテンツに適用されるセキュリティコントラクトとは異なります。このコントラクトは、開発者に、ブラウザーベースのアプリケーションには十分すぎるほど自由にリッチエクスペリエンスを実現できる豊富な機能を提供します。

AIR アプリケーションは、コンパイル済みバイトコード（SWF コンテンツ）またはインタープリットされるスクリプト（JavaScript、HTML）のいずれかを使用して作成されているので、ランタイムでメモリ管理できます。これにより、メモリ管理に関連する脆弱性（バッファーオーバーフローやメモリの破損など）によって AIR アプリケーションが影響を受ける可能性を最小限に抑えることができます。これらは、ネイティブコードで作成されたデスクトップアプリケーションに影響を与える最も一般的な脆弱性の一部です。

インストールとアップデート

Adobe AIR 1.0 およびそれ以降

AIR アプリケーションは、拡張子が air の AIR インストーラーファイルまたはネイティブインストーラーを使用して配布されます。ネイティブインストーラーでは、ネイティブプラットフォームのファイル形式と拡張子を使用します。例えば、Windows のネイティブインストーラーの形式は EXE ファイルです。Android のネイティブインストーラーの形式は APK ファイルです。

Adobe AIR がインストールされている環境で AIR インストーラーファイルを開く場合、AIR ランタイムによってインストールプロセスが管理されます。ネイティブインストーラーを使用する場合、オペレーティングシステムによってインストールプロセスが管理されます。

注意：開発者は、バージョン、アプリケーション名、発行者ソースを指定できますが、アプリケーションのインストールの最初のワークフロー自体は変更できません。ランタイムによって管理される安全で合理化された一貫性のあるインストール手順をすべての AIR アプリケーションで共有できるので、この制限はユーザーにとってメリットになります。アプリケーションのカスタマイズが必要な場合は、アプリケーションを初めて実行するときに指定できます。

ランタイムのインストール先

Adobe AIR 1.0 およびそれ以降

SWF ファイルで最初に Flash Player ブラウザープラグインをインストールする必要があるのと同様に、AIR アプリケーションでも最初にランタイムをユーザーのコンピューターにインストールする必要があります。

ランタイムはデスクトップコンピューター上の次の場所にインストールされます。

- Mac OS：/Library/Frameworks/
- Windows：C:¥Program Files¥Common Files¥Adobe AIR
- Linux：/opt/Adobe AIR/

Mac OS でアプリケーションのアップデートバージョンをインストールするには、アプリケーションディレクトリにインストールするための適切なシステム権限が必要です。Windows および Linux では、管理権限が必要です。

注意：iOS では、各 AIR ランタイムは個別にインストールされません。それぞれの AIR アプリケーションは 1 つのアプリケーションに統合されています。

ランタイムは 2 つの方法でインストールできます。1 つはシームレスインストール機能を使用する（Web ブラウザーから直接インストールする）方法で、もう 1 つは手動でインストールする方法です。

シームレスインストール（ランタイムおよびアプリーション）

Adobe AIR 1.0 およびそれ以降

シームレスインストール機能を使用すると、開発者は、Adobe AIR をインストールしていないユーザーに合理化されたインストールエクスペリエンスを提供できます。シームレスインストール方法では、開発者はインストール対象のアプリケーションを表す SWF ファイルを作成します。ユーザーが SWF ファイルをクリックしてアプリケーションをインストールすると、SWF ファイルはランタイムの検出を試みます。ランタイムが検出されない場合、ランタイムはインストールされて、開発者のアプリケーションのインストールプロセスですぐに有効化されます。

手動によるインストール

Adobe AIR 1.0 およびそれ以降

ユーザーが AIR ファイルを開く前にランタイムを手動でダウンロードしてインストールすることもできます。この場合、開発者は様々な方法（電子メールや Web サイト上の HTML リンクなど）で AIR ファイルを配布できます。AIR ファイルを開くと、ランタイムはアプリケーションのインストール処理を開始します。

アプリケーションのインストールフロー

Adobe AIR 1.0 およびそれ以降

AIR のセキュリティモデルでは、ユーザーが AIR アプリケーションをインストールするかどうかを決定できます。AIR インストールエクスペリエンスでは、ユーザーがこの信用判断をより簡単に行うことができるように、ネイティブアプリケーションのインストールテクノロジにいくつかの改良が行われています。

- ランタイムは、AIR アプリケーションが Web ブラウザー内のリンクからインストールされる場合でも、すべてのオペレーティングシステムで一貫したインストールエクスペリエンスを提供します。ネイティブアプリケーションのインストールエクスペリエンスは、セキュリティ情報がすべて提供される場合、セキュリティ情報を提供するブラウザーまたは他のアプリケーションに依存します。

- AIR アプリケーションのインストールエクスペリエンスでは、アプリケーションのソースと、アプリケーションで使用可能な権限（ユーザーがインストールの処理を許可されている場合）に関する情報を識別します。
- ランタイムは、AIR アプリケーションのインストールプロセスを管理します。AIR アプリケーションは、ランタイムが使用するインストールプロセスを操作できません。

一般的に、ユーザーは信頼していないソース、または検証できないソースからのデスクトップアプリケーションはインストールしません。他のインストール可能なアプリケーションと同様に、ネイティブアプリケーションのセキュリティに関する立証責任は、AIR アプリケーションにも同等に当てはまります。

アプリケーションのインストール先

Adobe AIR 1.0 およびそれ以降

インストールディレクトリは、次の 2 つのオプションのいずれかを使用して設定できます。

1. ユーザーがインストール中にインストール先をカスタマイズします。アプリケーションは、ユーザーが指定した場所にインストールされます。
2. ユーザーがインストール先を変更しない場合、アプリケーションはランタイムが指定した次のデフォルトパスにインストールされます。
   - Mac OS：~/Applications/
   - Windows XP 以前：C:¥¥rogram Files¥
   - Windows Vista：~¥¥
   - Linux：/opt/

   開発者がアプリケーション記述ファイルで installFolder 設定を指定している場合は、このディレクトリのサブパスにアプリケーションがインストールされます。

AIR ファイルシステム

Adobe AIR 1.0 およびそれ以降

AIR アプリケーションのインストールプロセスでは、開発者が AIR インストーラーファイルに含めたすべてのファイルがユーザーのローカルコンピューターにコピーされます。インストールされたアプリケーションの構成は次のとおりです。

- Windows：AIR インストーラーファイルに含まれるすべてのファイルを格納するディレクトリ。AIR アプリケーションのインストール時にはランタイムによって exe ファイルも作成されます。
- Linux：AIR インストーラーファイルに含まれるすべてのファイルを格納するディレクトリ。AIR アプリケーションのインストール時にはランタイムによって bin ファイルも作成されます。
- Mac OS：AIR インストーラーファイルのコンテンツがすべて含まれる app ファイル。このファイルは、Finder の「Show Package Contents」オプションを使用して調べることができます。ランタイムはこの app ファイルを AIR アプリケーションのインストール中に作成します。

AIR アプリケーションは次の方法で実行されます。

- Windows：インストールフォルダーで .exe ファイル、またはこのファイルに対応するショートカット（スタートメニューまたはデスクトップのショートカットなど）を実行します。
- Linux：インストールフォルダーの .bin ファイルを起動、アプリケーションメニューのアプリケーションを選択、またはエイリアスやデスクトップショートカットからアプリケーションを実行します。
- Mac OS：.app ファイル、または .app ファイルを参照するエイリアスを実行します。

アプリケーションファイルシステムには、アプリケーションの機能に関連するサブディレクトリも含まれます。例えば、暗号化されたローカル記憶領域に書き込まれた情報は、アプリケーションのアプリケーション ID に基づく名前が付いたディレクトリ内のサブディレクトリに保存されます。

AIR アプリケーション記憶領域

Adobe AIR 1.0 およびそれ以降

AIR アプリケーションは、ユーザーのハードディスク上のどの場所にも書き込むことができる権限を持っています。ただし、開発者は、作成したアプリケーションに関連するローカル記憶領域に app-storage:/ パスを使用することをお勧めします。アプリケーションから app-storage:/ に書き込まれたファイルは、ユーザーのオペレーティングシステムの標準の保存場所に保存されます。

- Mac OS：アプリケーションの記憶領域ディレクトリは <appData>/<appId>/Local Store/ です。ここで <appData> はユーザーの「基本設定フォルダー」であり、通常は /Users/<user>/Library/Preferences になります。
- Windows：アプリケーションの記憶領域ディレクトリは <appData>¥>¥¥ です。ここで、<appData> は、ユーザーの CSIDL_APPDATA 「特殊フォルダー」で、通常は C:¥Documents and Settings¥<user>¥Application Data になります。
- Linux：<appData>/<appID>/Local Store/ です。ここで、<appData> は、/home/<user>/.appdata です。

アプリケーション記憶領域ディレクトリには、air.File.applicationStorageDirectory プロパティを介してアクセスできます。このディレクトリのコンテンツには、File クラスの resolvePath() メソッドを使用してアクセスできます。詳しくは、143 ページの「ファイルシステムの操作」を参照してください。

Adobe AIR のアップデート

Adobe AIR 1.0 およびそれ以降

ランタイムのアップデートバージョンを必要とする AIR アプリケーションをユーザーがインストールすると、ランタイムによって必要なランタイムアップデートが自動的にインストールされます。

ランタイムをアップデートする作業は、当該コンピューターの管理権限を持つユーザーが実行する必要があります。

AIR アプリケーションのアップデート

Adobe AIR 1.0 およびそれ以降

ソフトウェアアップデートの開発と展開は、ネイティブコードアプリケーションの中でも最も大きなセキュリティ問題の 1 つです。AIR API には、この問題を改善するメカニズムが用意されています。Updater.update() メソッドを起動時に呼び出して、リモートの場所に AIR ファイルがあるかどうか確認することができます。アップデートが必要な場合、AIR ファイルがダウンロード、インストールされ、アプリケーションが再起動します。開発者はこのクラスを、新しい機能を提供するためだけでなく、潜在的なセキュリティの脆弱性に対処するためにも使用できます。

Updater クラスは、AIR ファイルとして配布されたアプリケーションの更新にのみ使用できます。ネイティブアプリケーションとして配布されたアプリケーションでは、ネイティブオペレーティングシステムの更新機能があれば、それを使用する必要があります。

注意：開発者はアプリケーション記述ファイルの versionNumber プロパティを設定してアプリケーションのバージョンを指定できます。

AIR アプリケーションのアンインストール

Adobe AIR 1.0 およびそれ以降

AIR アプリケーションを削除すると、アプリケーションディレクトリ内のすべてのファイルが削除されます。ただし、アプリケーションがアプリケーションディレクトリ以外のディレクトリに書き込んだファイルは削除されません。AIR アプリケーションを削除すると、AIR アプリケーションがアプリケーションディレクトリ以外のディレクトリに行った変更は元に戻りません。

管理者用の Windows レジストリ設定

Adobe AIR 1.0 およびそれ以降

Windows では、管理者は、AIR アプリケーションのインストールおよびランタイムのアップデートを行わないように（または許可するように）マシンを設定できます。これらの設定は、Windows レジストリの HKLM¥Software¥Policies¥Adobe¥AIR というキーの下にあります。次の設定があります。

レジストリ設定	説明
AppInstallDisabled	AIR アプリケーションのインストールとアンインストールを許可することを指定します。「許可」する場合は 0 に設定し、「許可しない」場合は 1 に設定します。
UntrustedAppInstallDisabled	信頼できない AIR アプリケーション（信頼できる証明書を含まないアプリケーション）のインストールを許可することを指定します。「許可」する場合は 0 に設定し、「許可しない」場合は 1 に設定します。
UpdateDisabled	ランタイムのアップデートを、バックグラウンドタスクまたは明示的なインストールの一部として許可することを指定します。「許可」する場合は 0 に設定し、「許可しない」場合は 1 に設定します。

Adobe AIR の HTML セキュリティ

Adobe AIR 1.0 およびそれ以降

このトピックでは、AIR HTML セキュリティアーキテクチャについて、およびインラインフレーム、フレーム、サンドボックスブリッジを使用して HTML ベースのアプリケーションを設定し、HTML コンテンツを SWF ベースのアプリケーションに安全に統合する方法について説明します。

ランタイムは、HTML および JavaScript の潜在的なセキュリティの脆弱性を克服するためのルールを適用し、メカニズムを提供します。同じルールは、アプリケーションを最初に JavaScript で記述する場合や HTML および JavaScript コンテンツを SWF ベースのアプリケーション内に読み込む場合にも適用されます。アプリケーションサンドボックスと非アプリケーションセキュリティサンドボックスのコンテンツは権限が異なります。コンテンツをインラインフレームまたはフレームに読み込むと、ランタイムによって安全なサンドボックスブリッジメカニズムが提供されて、フレームまたはインラインフレーム内のコンテンツはアプリケーションセキュリティサンドボックス内のコンテンツと安全にやり取りできるようになります。

AIR SDK には、HTML コンテンツのレンダリングに使用する 3 つのクラスが用意されています。

HTMLLoader クラスは、JavaScript コードと AIR API との密接な統合を提供します。

StageWebView クラスは、HTML レンダリングクラスです。このクラスとホスト AIR アプリケーションとの統合は非常に限定されています。StageWebView クラスによって読み込まれたコンテンツは、アプリケーションセキュリティサンドボックスに配置されません。また、ホスト AIR アプリケーションでのデータへのアクセスや関数の呼び出しもできません。デスクトッププラットフォームの場合、StageWebView クラスは、Webkit をベースとしたビルトイン AIR HTML エンジンを使用します。このエンジンは HTMLLoader クラスでも使用されます。モバイルプラットフォームの場合、StageWebView

クラスは、オペレーティングシステムが提供する HTML コントロールを使用します。したがって、モバイルプラットフォームの場合、StageWebView クラスには、システムの Web ブラウザーと同じセキュリティ上の考慮事項と脆弱性があります。

TextField クラスは、HTML テキストのストリングを表示できます。JavaScript を実行することはできませんが、リンクおよび外部から読み込まれたイメージをテキストに含めることはできます。

詳しくは、23 ページの「セキュリティ関連の JavaScript エラーの回避」を参照してください。

HTML ベースのアプリケーションの設定の概要

Adobe AIR 1.0 およびそれ以降

フレームとインラインフレームは、AIR で HTML コンテンツを編成するための便利な構造を提供します。フレームを使用すると、データの永続化の維持とリモートコンテンツの安全な使用の両方が実現できます。

AIR 内の HTML では標準のページベースの編成が維持されるので、HTML コンテンツの最上位のフレームが別のページに「移動」すると HTML 環境は完全に更新されます。フレームとインラインフレームを使用すると、ブラウザーで実行されている Web アプリケーションと同じように、AIR でデータの永続性を維持できます。最上位のフレームでメインのアプリケーションオブジェクトを定義すると、フレームが新しいページに移動できない限り、それらのオブジェクトは永続化されます。アプリケーションの一時的な部分を読み込んで表示するには、子のフレームまたはインラインフレームを使用します（フレーム以外にも使用できるデータ永続化の維持方法は多数あります。例えば、Cookie、ローカル共有オブジェクト、ローカルファイルストレージ、暗号化されたファイルストア、ローカルデータベースストレージがあります）。

AIR では、HTML の実行可能なコードとデータとの間に通常のあいまいな境界が維持されるため、HTML 環境のトップフレームのコンテンツはアプリケーションサンドボックスに配置されます。ページの load イベント後は、テキストのストリングを実行可能なオブジェクトに変換する eval() などの操作は制限されます。この制限は、アプリケーションがリモートコンテンツを読み込まない場合でも適用されます。そのような制限された操作を HTML コンテンツで実行できるようにするには、フレームまたはインラインフレームを使用して、コンテンツをアプリケーションとは別のサンドボックス内に配置する必要があります（eval() 関数に依存する JavaScript アプリケーションフレームワークを使用する場合も、サンドボックス化した子フレーム内でコンテンツを実行する必要が生じることがあります）。アプリケーションサンドボックスでの JavaScript に関する制限について詳しくは、75 ページの「異なるサンドボックス内のコンテンツに対するコードの制限」を参照してください。

AIR 内の HTML ではリモートの安全でない可能性のあるコンテンツを読み込む機能が維持されるので、AIR では別のドメイン内のコンテンツとやり取りを回避するための同一生成元ポリシーを適用します。アプリケーションコンテンツと別のドメイン内のコンテンツ間のやり取りを許可するためには、親フレームと子フレーム間のインターフェイスとして機能するブリッジを設定します。

親子サンドボックス関係の設定

Adobe AIR 1.0 およびそれ以降

AIR では、HTML の frame および iframe エレメントに sandboxRoot 属性と documentRoot 属性を追加します。これらの属性を使用すると、アプリケーションコンテンツを別のドメインからのコンテンツとして扱うことができます。

属性	説明
sandboxRoot	フレームコンテンツを配置するサンドボックスとドメインを決定するために使用する URL。file:、http: または https: URL スキームを使用する必要があります。
documentRoot	フレームコンテンツの読み込み元の URL。file:、app: または app-storage: URL スキームを使用する必要があります。

次の例では、アプリケーションのサンドボックスサブディレクトリにインストールされているコンテンツをマップして、リモートサンドボックスおよび www.example.com ドメインで実行します。

```
<iframe
    src="ui.html"
    sandboxRoot="http://www.example.com/local/"
    documentRoot="app:/sandbox/">
</iframe>
```

異なるサンドボックスまたはドメインでの親フレームと子フレーム間のブリッジの設定

Adobe AIR 1.0 およびそれ以降

AIR では、childSandboxBridge および parentSandboxBridge プロパティを、任意の子フレームの window オブジェクトに追加します。これらのプロパティを使用すると、親フレームと子フレームの間のインターフェイスとして機能するブリッジを定義できます。各ブリッジは一方向に移動します。

childSandboxBridge — childSandboxBridge プロパティを使用すると、子フレームで親フレーム内のコンテンツにインターフェイスを公開できます。インターフェイスを公開するには、childSandbox プロパティを子フレーム内の関数またはオブジェクトに設定します。これで、親フレーム内のコンテンツからオブジェクトまたは関数にアクセスできます。次の例では、子フレームで実行中のスクリプトで、関数とプロパティを 1 つずつ含むオブジェクトをその親に公開する方法を示します。

```
var interface = {};
interface.calculatePrice = function(){
    return .45 + 1.20;
}
interface.storeID = "abc"
window.childSandboxBridge = interface;
```

id として「child」が割り当てられたインラインフレーム内に子コンテンツがある場合、フレームの childSandboxBridge プロパティを読み取って親コンテンツからインターフェイスにアクセスできます。

```
var childInterface = document.getElementById("child").childSandboxBridge;
air.trace(childInterface.calculatePrice()); //traces "1.65"
air.trace(childInterface.storeID)); //traces "abc"
```

parentSandboxBridge — parentSandboxBridge プロパティを使用すると、親フレームは子フレーム内のコンテンツにインターフェイスを公開できます。インターフェイスを公開するには、子フレームの parentSandbox プロパティを、親フレーム内の関数またはオブジェクトに設定します。これで、子フレーム内のコンテンツからオブジェクトまたは関数にアクセスできます。次の例では、親フレームで実行中のスクリプトで、save 関数を含むオブジェクトを子に公開する方法を示します。

```
var interface = {};
interface.save = function(text){
    var saveFile = air.File("app-storage:/save.txt");
    //write text to file
}
document.getElementById("child").parentSandboxBridge = interface;
```

このインターフェイスを使用すると、子フレーム内のコンテンツで、テキストを save.txt という名前のファイルに保存できます。ただし、コンテンツはファイルシステムに対する他のアクセス権を持つことはできません。一般的に、アプリケーションコンテンツは、他のサンドボックスにできる限り範囲の狭いインターフェイスを公開する必要があります。子コンテンツは、save 関数を次のように呼び出します。

```
var textToSave = "A string.";
window.parentSandboxBridge.save(textToSave);
```

子コンテンツが parentSandboxBridge オブジェクトのプロパティを設定しようとすると、ランタイムによって SecurityError 例外がスローされます。親コンテンツが childSandboxBridge オブジェクトのプロパティを設定しようとすると、ランタイムによって SecurityError 例外がスローされます。

異なるサンドボックス内のコンテンツに対するコードの制限

Adobe AIR 1.0 およびそれ以降

このトピックの概要（72 ページの「Adobe AIR の HTML セキュリティ」）で説明したように、ランタイムは、HTML および JavaScript の潜在的なセキュリティの脆弱性を克服するためのルールを適用し、メカニズムを提供します。ここでは、それらの制限について説明します。コードでこれらの制限された API を呼び出そうとすると、ランタイムによって、「Adobe AIR runtime security violation for JavaScript code in the application security sandbox」というメッセージのエラーがスローされます。

詳しくは、23 ページの「セキュリティ関連の JavaScript エラーの回避」を参照してください。

JavaScript eval() 関数および同様の方法の使用に関する制限

Adobe AIR 1.0 およびそれ以降

アプリケーションセキュリティサンドボックス内の HTML コンテンツの場合は、コードが読み込まれた後（body エレメントの onload イベントが送出され、onload ハンドラー関数が実行を完了した後）にストリングを実行可能コードに動的に変換できる API の使用について制限があります。これは、アプリケーションで、アプリケーション以外のソース（潜在的に安全でないネットワークドメインなど）から誤ってコードを挿入（および実行）されないようにするための制限です。

例えば、アプリケーションがリモートソースのストリングデータを使用して DOM エレメントの innerHTML プロパティに書き込みを行うと、ストリングには安全ではない操作を実行する可能性のある実行可能（JavaScript）コードが含まれている場合があります。ただし、コンテンツが読み込まれている間は、DOM にリモートストリングが挿入される可能性はありません。

1 つの制限は、JavaScript eval() 関数の使用に関するものです。アプリケーションサンドボックス内のコードが読み込まれ、onload イベントハンドラーの処理が完了した後は、eval() 関数を制限された方法でのみ使用できます。eval() 関数（アプリケーションセキュリティサンドボックスからコードが読み込まれた後）の使用については、次のルールが適用されます。

- リテラルを含む式は許可されます。次に、例を示します。

```
eval("null");
eval("3 + .14");
eval("'foo'");
```

- 次のようなオブジェクトリテラルは許可されます。

```
{ prop1: val1, prop2: val2 }
```

- 次のようなオブジェクトリテラル setter/getter は、禁止されます。

```
{ get prop1() { ... }, set prop1(v) { ... } }
```

- 次のような配列リテラルは許可されます。

```
[ val1, val2, val3 ]
```

- 次のようなプロパティの読み込みを含む式は、禁止されます。

```
a.b.c
```

- 関数の呼び出しは、禁止されます。
- 関数の定義は、禁止されます。
- プロパティの設定は、禁止されます。
- 関数リテラルは、禁止されます。

ただし、コードの読み込み中、onload イベントの前、および onload イベントハンドラー関数の実行中は、これらの制限はアプリケーションセキュリティサンドボックス内のコンテンツに適用されません。

例えば、コードが読み込まれた後に、次のコードを実行するとランタイムによって例外がスローされます。

```
eval("alert(44)");
eval("myFunction(44)");
eval("NativeApplication.applicationID");
```

eval() 関数を呼び出すときに作成されるような、動的に生成されたコードがアプリケーションサンドボックス内で許可されると、セキュリティリスクが発生します。例えば、アプリケーションがネットワークドメインから読み込まれたストリングを誤って実行し、そのストリングに悪意のあるコードが含まれている可能性があります。これは例えば、ユーザーのコンピューターのファイルを削除または変更するコードである可能性があります。または、信頼されないネットワークドメインにローカルファイルのコンテンツを送り返すコードである場合もあります。

動的なコードを生成する方法は次のとおりです。

- eval() 関数を呼び出します。
- innerHTML プロパティまたは DOM 関数を使用して、アプリケーションディレクトリ外のスクリプトを読み込む script タグを挿入します。
- innerHTML プロパティまたは DOM 関数を使用して、（src 属性を使用してスクリプトを読み込むのではなく）インラインコードを含む script タグを挿入します。
- src 属性を script タグに設定し、アプリケーションディレクトリ外にある JavaScript ファイルを読み込みます。
- javascript URL スキームを使用します（href="javascript:alert('Test')" など）。
- setInterval() または setTimout() 関数を使用します。この場合、（非同期で実行する関数を定義する）最初の関数が関数名ではなく（評価される）ストリングになります（setTimeout('x = 4', 1000) など)。
- document.write() または document.writeln() を呼び出します。

アプリケーションセキュリティサンドボックス内のコードは、コンテンツを読み込んでいるときにのみこれらのメソッドを使用できます。

これらの制限で回避されないのは、JSON オブジェクトリテラルを含む eval() の使用です。これによってアプリケーションコンテンツは、JSON JavaScript ライブラリで使用できます。ただし、オーバーロードされた JSON コード（イベントハンドラーを含む）の使用は制限されます。

その他の Ajax フレームワークおよび JavaScript コードライブラリの場合は、フレームワークまたはライブラリ内のコードが動的に生成されたコードに関するこれらの制限内で機能しているかどうかを確認します。制限に従っていない場合は、フレームワークまたはライブラリを使用するコンテンツを非アプリケーションセキュリティサンドボックスに含めます。詳しくは、AIR における JavaScript の制限および 81 ページの「アプリケーションコンテンツと非アプリケーションコンテンツ間のスクリプト作成」を参照してください。アドビ システムズ社では、アプリケーションセキュリティサンドボックスをサポートする Ajax フレームワークのリストを http://www.adobe.com/products/jp/air/develop/ajax/features/ で管理しています。

アプリケーションセキュリティサンドボックスのコンテンツとは異なり、非アプリケーションセキュリティサンドボックスの JavaScript コンテンツは、eval() 関数を呼び出して、動的に生成されたコードをいつでも実行できます。

AIR API へのアクセスに関する制限（非アプリケーションサンドボックス）

Adobe AIR 1.0 およびそれ以降

非アプリケーションサンドボックス内の JavaScript コードは、window.runtime オブジェクトにアクセスできないので、AIR API を実行できません。非アプリケーションセキュリティサンドボックス内のコンテンツが次のコードを呼び出すと、アプリケーションは TypeError 例外をスローします。

```
try {
    window.runtime.flash.system.NativeApplication.nativeApplication.exit();
}
catch (e)
{
    alert(e);
}
```

例外のタイプが TypeError（未定義の値）であるのは、非アプリケーションサンドボックス内のコンテンツでは window.runtime オブジェクトが認識されず、未定義の値として示されるからです。

スクリプトブリッジを使用してランタイム機能を非アプリケーションサンドボックス内のコンテンツに公開できます。詳しくは、81 ページの「アプリケーションコンテンツと非アプリケーションコンテンツ間のスクリプト作成」を参照してください。

XMLHttpRequest 呼び出しの使用に関する制限

Adobe AIR 1.0 およびそれ以降

アプリケーションセキュリティサンドボックス内の HTML コンテンツでは、HTML コンテンツが読み込まれている間や onLoad イベントの発生時に同期 XMLHttpRequest メソッドを使用してアプリケーションサンドボックス外からデータを読み込むことはできません。

デフォルトでは、非アプリケーションセキュリティサンドボックス内の HTML コンテンツで JavaScript XMLHttpRequest オブジェクトを使用して、リクエストを呼び出すドメイン以外のドメインからデータを読み込むことはできません。frame または iframe タグには、allowcrosscomainxhr 属性を含めることができます。この属性を null 値以外の値に設定すると、frame または iframe 内のコンテンツで Javascript XMLHttpRequest オブジェクトを使用して、リクエストを呼び出しているコードのドメイン以外のドメインからデータを読み込むことができます。

```
<iframe id="UI"
    src="http://example.com/ui.html"
    sandboxRoot="http://example.com/"
    allowcrossDomainxhr="true"
    documentRoot="app:/">
</iframe>
```

詳しくは、78 ページの「異なるドメインのコンテンツ間のスクリプト作成」を参照してください。

CSS、frame、iframe および img エレメントの読み込みに関する制限（非アプリケーションサンドボックス内のコンテンツ）

Adobe AIR 1.0 およびそれ以降

リモート（ネットワーク）セキュリティサンドボックス内の HTML コンテンツは、CSS、frame、iframe および img エレメントをリモートサンドボックス（ネットワーク URL）からのみ読み込みます。

local-with-filesystem、local-with-networking または local-trusted サンドボックス内の HTML コンテンツは、CSS、frame、iframe および img コンテンツを（アプリケーションサンドボックスまたはリモートサンドボックスからではなく）ローカルサンドボックスからのみ読み込みます。

JavaScript window.open() メソッドの呼び出しに関する制限

Adobe AIR 1.0 およびそれ以降

JavaScript window.open() メソッドの呼び出しによって作成されたウィンドウに非アプリケーションセキュリティサンドボックスのコンテンツが表示される場合、ウィンドウのタイトルはメイン（起動している）ウィンドウのタイトルで始まり、その後にコロンが続きます。コードを使用してウィンドウのタイトルの部分を画面から移動することはできません。

非アプリケーションセキュリティサンドボックス内のコンテンツは、ユーザーがマウスやキーボードを操作すると発生するイベントに応じてのみ JavaScript window.open() メソッドを正しく呼び出すことができます。これにより、非アプリケーションコンテンツで、詐欺に使用される（フィッシング攻撃など）可能性のあるウィンドウが作成されることはなくなります。また、マウスイベントやキーボードイベントのイベントハンドラーでは、window.open() メソッドを、遅延の後に（setTimeout() 関数を呼び出す場合など）実行されるように設定することはできません。

リモート（ネットワーク）サンドボックス内のコンテンツでは、リモートネットワークサンドボックス内のコンテンツを開く目的でのみ window.open() メソッドを使用できます。window.open() メソッドを使用して、アプリケーションサンドボックスやローカルサンドボックスのコンテンツを開くことはできません。

local-with-filesystem、local-with-networking または local-trusted サンドボックス内のコンテンツ（セキュリティサンドボックスを参照）は、ローカルサンドボックスのコンテンツを開く目的でのみ window.open() メソッドを使用できます。window.open() を使用してアプリケーションサンドボックスやリモートサンドボックスのコンテンツを開くことはできません。

制限付きコードを呼び出した場合のエラー

Adobe AIR 1.0 およびそれ以降

これらのセキュリティ制限によりサンドボックスで使用が制限されているコードを呼び出すと、ランタイムは JavaScript エラー「Adobe AIR runtime security violation for JavaScript code in the application security sandbox」を送出します。

詳しくは、23 ページの「セキュリティ関連の JavaScript エラーの回避」を参照してください。

ストリングから HTML コンテンツを読み込む場合のサンドボックスによる保護

Adobe AIR 1.0 およびそれ以降

HTMLLoader クラスの loadString() メソッドを使用すると、実行時に HTML コンテンツを作成できます。ただし、安全でないインターネットソースから読み込んだデータを HTML コンテンツとして使用すると、データが壊れる場合があります。このため、デフォルトでは、loadString() メソッドを使用して作成した HTML はアプリケーションサンドボックスに配置されず、AIR API にアクセスできません。ただし、HTMLLoader オブジェクトの placeLoadStringContentInApplicationSandbox プロパティを true に設定すると、loadString() メソッドを使用して作成した HTML をアプリケーションサンドボックスに配置できます。詳しくは、ストリングからの HTML コンテンツの読み込みを参照してください。

異なるドメインのコンテンツ間のスクリプト作成

Adobe AIR 1.0 およびそれ以降

AIR アプリケーションは、インストール時に特殊な権限を付与されます。アプリケーションに含まれていないリモートファイルやローカルファイルなどの他のコンテンツに同じ権限情報が流出しないようにすることが重要です。

AIR サンドボックスブリッジについて

Adobe AIR 1.0 およびそれ以降

通常、他のドメインのコンテンツで、他のドメインのスクリプトを呼び出すことはできません。

メインの AIR アプリケーションが、リモートドメインのコンテンツに対してメインの AIR アプリケーション内のスクリプトへの制御付きアクセス権を持っていることを要求する場合や、その逆の場合もあります。これを実現するために、ランタイムには、2 つのサンドボックス間のゲートウェイとして機能する、サンドボックスブリッジメカニズムがあります。サンドボックスブリッジを使用すると、リモートサンドボックスとアプリケーションセキュリティサンドボックス間で明示的な操作を実行できます。

サンドボックスブリッジは、読み込まれる側のスクリプトと読み込む側のスクリプトの両方がアクセスできる 2 つのオブジェクトを公開します。

- parentSandboxBridge オブジェクトを使用すると、読み込む側のコンテンツは読み込まれる側のコンテンツのスクリプトにプロパティと関数を公開できます。
- childSandboxBridge オブジェクトを使用すると、読み込まれる側のコンテンツが読み込む側のコンテンツのスクリプトにプロパティと関数を公開できます。

サンドボックスブリッジによって公開されるオブジェクトは、参照ではなく値によって受け渡されます。すべてのデータは直列化されます。つまり、ブリッジの片側で公開されたオブジェクトはもう一方の側で設定できず、公開されたそのオブジェクトはすべて型指定されません。また、公開できるのは単純なオブジェクトと関数だけで、複雑なオブジェクトは公開できません。

子コンテンツが parentSandboxBridge オブジェクトのプロパティを設定しようとすると、ランタイムによって SecurityError 例外がスローされます。同様に、親コンテンツが childSandboxBridge オブジェクトのプロパティを設定しようとすると、ランタイムによって SecurityError 例外がスローされます。

サンドボックスブリッジの例（HTML）

Adobe AIR 1.0 およびそれ以降

HTML コンテンツで、parentSandboxBridge プロパティと childSandboxBridge プロパティが子ドキュメントの JavaScript の window オブジェクトに追加されます。HTML コンテンツでのブリッジ関数の設定方法の例については、37 ページの「サンドボックスブリッジインターフェイスの設定」を参照してください。

API 公開の制限

Adobe AIR 1.0 およびそれ以降

サンドボックスブリッジを公開する場合は、高度なレベルの API を公開してそれらが悪用されない程度まで制限することが重要です。ブリッジ実装を呼び出しているコンテンツは（コードの挿入などにより）改ざんされる可能性があることに注意してください。例えば、readFile(path) メソッド（任意のファイルのコンテンツを読み込むメソッド）をブリッジを通じて公開すると、悪用されやすくなります。この場合はパスを取得せずに特定のファイルを読み込む readApplicationSetting() API を公開する方が安全です。アプリケーションの一部が改ざんされた後でもアプリケーションで実行できる、より意味のある方法によって、損害の広がりを抑えることができます。

関連項目

35 ページの「異なるセキュリティサンドボックス内のコンテンツのクロススクリプト」

ディスクへの書き込み

Adobe AIR 1.0 およびそれ以降

Web ブラウザーで実行中のアプリケーションは、ユーザーのローカルファイルシステムとのやり取りのみが制限されます。Web ブラウザーは、Web コンテンツを読み込んでもユーザーのコンピューターが改ざんされないことを保証するセキュリティポリシーを実装します。例えば、ブラウザーで Flash Player を使って実行されている SWF ファイルは、ユーザーのコンピューターに既に存在するファイルと直接やり取りできません。共有オブジェクトと Cookie はユーザーの環境設定およびその他のデータを管理する目的でユーザーのコンピューターに書き込むことができますが、これはファイルシステム操作の制限になります。AIR アプリケーションはネイティブにインストールされるため、ローカルファイルシステムに対する読み取りと書き込みの機能を含む、異なるセキュリティコントラクトを持ちます。

この自由度には開発者の大きな責任が伴います。アプリケーションの予想外の不安定さは、アプリケーションの機能だけでなく、ユーザーのコンピューターの完全性も危険にさらします。このような理由から、開発者は、81 ページの「開発者のためのセキュリティのベストプラクティス」を参照する必要があります。

AIR 開発者は、いくつかの URL スキーム規則を使用してローカルファイルシステムにアクセスしたり、ファイルをそのシステムに書き込むことができます。

URL スキーム	説明
app:/	アプリケーションディレクトリのエイリアス。このパスからアクセスされたファイルはアプリケーションサンドボックスに割り当てられ、ランタイムによって完全な権限が与えられます。
app-storage:/	ランタイムによって標準化されたローカル記憶領域ディレクトリのエイリアス。このパスからアクセスされたファイルは非アプリケーションサンドボックスに割り当てられます。
file:///	ユーザーのハードディスクのルートを表すエイリアス。このパスからアクセスされたファイルは、アプリケーションディレクトリ内に存在している場合はアプリケーションサンドボックスに割り当てられ、それ以外の場合は非アプリケーションサンドボックスに割り当てられます。

注意：AIR アプリケーションは、app: URL スキームを使用してコンテンツを変更できません。また、アプリケーションディレクトリは、管理者の設定により、読み取り専用になる場合があります。

ユーザーのコンピューターに対する管理者の制限がない限り、AIR アプリケーションには、ユーザーのハードドライブ上のどの場所にも書き込むことができる権限が付与されます。開発者には、app-storage:/ パスを、アプリケーションに関連するローカル記憶領域に使用することをお勧めします。アプリケーションから app-storage:/ へ書き込まれたファイルは、標準の場所に挿入されます。

- Mac OS：アプリケーションの記憶領域ディレクトリは <appData>/<appId>/Local Store/ です。ここで <appData> はユーザーの基本設定フォルダーです。これは通常 /Users/<user>/Library/Preferences になります。
- Windows：アプリケーションの記憶領域ディレクトリは <appData>¥<appId>¥Local Store¥ です。ここで <appData> はユーザーの CSIDL_APPDATA 特殊フォルダーです。通常は、C:¥Documents and Settings¥<userName>¥Application Data になります。
- Linux：<appData>/<appID>/Local Store/ です。ここで、<appData> は、/home/<user>/.appdata です。

アプリケーションがユーザーのファイルシステム内の既存のファイルとやり取りするように設計されている場合は、必ず 81 ページの「開発者のためのセキュリティのベストプラクティス」を参照してください。

信頼されないコンテンツの安全な使用

Adobe AIR 1.0 およびそれ以降

アプリケーションサンドボックスに割り当てられていないコンテンツは、ランタイムのセキュリティ基準を満たす場合のみアプリケーションに追加のスクリプト機能を提供できます。ここでは、非アプリケーションコンテンツでの AIR セキュリティコントラクトについて説明します。

アプリケーションコンテンツと非アプリケーションコンテンツ間のスクリプト作成

Adobe AIR 1.0 およびそれ以降

アプリケーションコンテンツと非アプリケーションコンテンツとの間をスクリプト化する AIR アプリケーションではさらに複雑なセキュリティの調整が行われます。アプリケーションサンドボックス内に含まれていないファイルは、サンドボックスブリッジを使用したアプリケーションサンドボックス内のファイルのプロパティとメソッドへのアクセスのみが許可されます。サンドボックスブリッジはアプリケーションコンテンツと非アプリケーションコンテンツ間のゲートウェイとして機能し、2 つのファイル間の明示的な操作を可能にします。サンドボックスブリッジを正しく使用すると、セキュリティの追加のレイヤーが提供され、非アプリケーションコンテンツが、アプリケーションコンテンツの一部となるオブジェクト参照にアクセスするのを制限します。

サンドボックスブリッジのメリットについて、例を使って説明します。AIR ミュージックストアアプリケーションは、独自の SWF ファイルを作成する必要のある広告主に API を提供し、ストアアプリケーションはこのファイルを使用して通信できます。ストアは、ストアが提供するアーティストと CD を探すためのメソッドを広告主に提供する必要がありますが、セキュリティ上の理由でサードパーティの SWF ファイルからメソッドとプロパティを分離する必要もあります。

サンドボックスブリッジはこの機能を提供できます。デフォルトでは、ランタイムで外部から AIR アプリケーションに読み込まれたコンテンツには、メインアプリケーションのどのメソッドまたはプロパティへのアクセス権もありません。カスタムのサンドボックスブリッジ実装を使用すると、開発者はこれらのメソッドやプロパティを公開せずにリモートコンテンツにサービスを提供できます。サンドボックスブリッジは信頼されるコンテンツと信頼されないコンテンツ間の通路と見なすことができ、オブジェクト参照を公開せずに読み込む側のコンテンツと読み込まれる側のコンテンツ間の通信を可能にします。

サンドボックスブリッジを安全に使用する方法について詳しくは、78 ページの「異なるドメインのコンテンツ間のスクリプト作成」を参照してください。

開発者のためのセキュリティのベストプラクティス

Adobe AIR 1.0 およびそれ以降

AIR アプリケーションが Web テクノロジを使用して構築されていても、開発者はそれらの AIR アプリケーションがブラウザーのセキュリティサンドボックス内で機能していないことに気づくことが重要です。つまり、故意にまたは誤ってローカルシステムに損害を与える可能性がある AIR アプリケーションを構築する場合があります。AIR はリスクを最小化しようと試みますが、脆弱性が取り込まれる可能性はまだあります。ここでは、重要なセキュリティ上の潜在的なリスクについて説明します。

ファイルをアプリケーションセキュリティサンドボックスにインポートした場合のリスク

Adobe AIR 1.0 およびそれ以降

アプリケーションディレクトリに存在するファイルはアプリケーションサンドボックスに割り当てられ、ランタイムの完全な権限が付与されます。ローカルファイルシステムに書き込みを行うアプリケーションは、app-storage:/ に書き込むことをお勧めします。このディレクトリはユーザーのコンピューター上のアプリケーションファイルとは別の場所にあるため、ファイルはアプリケーションサンドボックスに割り当てられず、セキュリティのリスクは軽減されます。開発者は次の点に注意する必要があります。

- ファイルは必要な場合にのみ AIR ファイル（インストール済みアプリケーション内）に含めます。
- スクリプトファイルは、動作が完全に把握され、信頼される場合にのみ AIR ファイル（インストール済みアプリケーション内）に含めます。
- アプリケーションディレクトリに書き込みを行ったり、アプリケーションディレクトリ内のコンテンツを変更したりしないでください。ランタイムでは、アプリケーションが app:/ URL スキームを使用してファイルやディレクトリに書き込みを行ったり、それらを変更したりすると、SecurityError 例外をスローしてその操作を回避します。
- ネットワークソースのデータを AIR API のメソッドのパラメーターとして使用しないでください。これによりコードが実行される可能性があります。これには、Loader.loadBytes() メソッドや JavaScript eval() 関数の使用が含まれます。

外部ソースを使用してパスを判断する場合のリスク

Adobe AIR 1.0 およびそれ以降

AIR アプリケーションは、外部のデータまたはコンテンツを使用すると、改ざんされる可能性があります。このため、ネットワークまたはファイルシステムのデータを使用する場合は特に注意する必要があります。信頼の責任は最終的には開発者と、開発者が作成するネットワーク接続にありますが、外部データの読み込みは本来リスクを伴うものであるので、機密データの操作への入力には使用しないでください。開発者は次の操作を行わないことをお勧めします。

- ネットワークソースのデータを使用したファイル名の決定
- ネットワークソースのデータを使用した、アプリケーションでの個人情報の送信に使用される URL の作成

保護されていない資格情報を使用、保存または送信する場合のリスク

Adobe AIR 1.0 およびそれ以降

ユーザー資格情報をユーザーのローカルファイルシステムに保存することは本来リスクを伴い、これらの資格情報が改ざんされる可能性があります。開発者は次の点に注意する必要があります。

- 資格情報をローカルに保存する必要がある場合は、ローカルファイルシステムに書き込むときに資格情報を暗号化します。ランタイムでは、EncryptedLocalStore クラスを使って各インストール済みアプリケーションに固有の暗号化された記憶領域を提供します。詳しくは、250 ページの「暗号化されたローカルストア」を参照してください。
- ネットワークソースが信頼され、送信に HTTPS: または Transport Layer Security（TLS）プロトコルが使用されているのでなければ、暗号化されていないユーザー資格情報をそのソースに送信しないでください。
- 資格情報の作成時にデフォルトのパスワードを指定しないでください。パスワードはユーザー自身が作成できるようにしてください。デフォルトパスワードを指定すると、それが攻撃者に知られた場合、デフォルトを変更しないで使用しているユーザーの資格情報が無防備な状態に置かれることになります。

ダウングレード攻撃のリスク

Adobe AIR 1.0 およびそれ以降

アプリケーションのインストール中に、ランタイムはアプリケーションのバージョンが現在インストールされていないことを確認します。アプリケーションが既にインストールされている場合、ランタイムはバージョンストリングをインストールされているバージョンと比較します。このストリングが異なる場合、ユーザーはインストールのアップグレードを選択できます。ランタイムは、新しくインストールされたバージョンが古いバージョンよりも新しいことを保証せず、それが異なるバージョンであるということだけを保証します。攻撃者は古いバージョンをユーザーに配布して、セキュリティの弱点をくぐり抜ける可能性があります。このため、開発者はアプリケーションの実行時にバージョンの確認を行うことをお勧めします。ネットワークで要求されたアップデートがあるかどうかをアプリケーションに確認させるのも良い方法です。その場合、攻撃者がユーザーに古いバージョンを実行させていても、その古いバージョンは更新する必要があることを認識します。また、アプリケーションで明確なバージョン管理スキームを使用すると、ユーザーがダウングレードバージョンをインストールする可能性が低くなります。

コード署名

Adobe AIR 1.0 およびそれ以降

すべての AIR インストーラーファイルには、コード署名を実行する必要があります。コード署名は、ソフトウェアの指定された作成元が正確であるかどうかを確認する暗号化プロセスです。AIR アプリケーションの署名には、外部の認証機関（CA）が発行した証明書または独自に作成した自己署名入り証明書を使用できます。有名な CA の商用証明書を使用することを強くお勧めします。これらの証明書を使用すると、偽造ではない正しいアプリケーションをインストールしていることをユーザーに保証できます。ただし、SDK から adt を使用するか、証明書の作成に adt を使用する Flash、Flash Builder または別のアプリケーションを使用して、自己署名入り証明書を作成できます。自己署名入り証明書では、インストールするアプリケーションが本物であることを保証できません。この証明書は、アプリケーションを公開する前のテスト用としてのみご使用ください。

第 7 章：AIR ネイティブウィンドウの操作

Adobe AIR 1.0 およびそれ以降

デスクトップウィンドウを作成および管理するには、Adobe® AIR® ネイティブウィンドウ API のクラスを使用します。

AIR のネイティブウィンドウの基礎

Adobe AIR 1.0 およびそれ以降

AIR でのネイティブウィンドウの操作に関する簡単な説明およびコード例については、Adobe Developer Connection で次のクイックスタートの記事を参照してください。

- ウィンドウの外観のカスタマイズ

AIR には、Flash®、Flex™ および HTML のプログラミング技法を使用してオペレーティングシステムのネイティブウィンドウを作成するための、簡単に使用できるクロスプラットフォームのウィンドウ API が用意されています。

AIR では、アプリケーションの外観を柔軟に開発できます。ユーザーが作成するウィンドウは、標準のデスクトップアプリケーションのようになります。Macintosh で実行する場合は Apple スタイルに一致し、Windows で実行する場合は Microsoft の規則に準拠し、Linux で実行する場合はウィンドウマネージャーに適合します。いずれの場合も、プラットフォーム固有のコード行を含める必要はありません。または、Flex フレームワークで提供されているスキンとして使用できて拡張性のあるクロムを使用して、アプリケーションの実行環境に左右されない独自のスタイルを作成することもできます。さらには、デスクトップに対する透明度とアルファブレンドを完全にサポートするベクトルとビットマップのアートワークを使用して、独自のウィンドウクロムを描画することもできます。矩形のウィンドウに飽きているなら、丸いウィンドウを作成してみてください。

AIR のウィンドウ

Adobe AIR 1.0 およびそれ以降

AIR では、ウィンドウを操作するための 3 つの API がサポートされています。

- ActionScript 指向の NativeWindow クラスは、最下位レベルのウィンドウ API を提供します。NativeWindows は、ActionScript および Flash Professional オーサリングによるアプリケーションで使用します。アプリケーションで使用する目的に特化したウィンドウを作成する場合は、NativeWindow クラスの拡張を検討してください。
- HTML 環境では、ブラウザーベースの Web アプリケーションの場合と同じように JavaScript Window クラスを使用できます。JavaScript Window メソッドの呼び出しは、基になるネイティブウィンドウオブジェクトに転送されます。
- Flex framework mx:WindowedApplication クラスおよび mx:Window クラスは、NativeWindow クラスに Flex の「ラッパー」を提供します。Flex を使用して作成した AIR アプリケーションを Flex アプリケーションの初期ウィンドウとして使用する必要がある場合、WindowedApplication コンポーネントは Application コンポーネントを置き換えます。

ActionScript ウィンドウ

NativeWindow クラスでウィンドウを作成するときは、Flash Player のステージと表示リストを直接使用します。NativeWindow に表示オブジェクトを追加するには、ウィンドウステージの表示リストまたはステージの別の表示オブジェクトコンテナにオブジェクトを追加します。

HTML ウィンドウ
HTML ウィンドウを作成するときは、HTML、CSS および JavaScript を使用してコンテンツを表示します。表示オブジェクトを HTML ウィンドウに追加するには、そのコンテンツを HTML DOM に追加します。HTML ウィンドウは、NativeWindow の特殊なカテゴリです。AIR ホストでは、基になる NativeWindow インスタンスへのアクセスを提供する nativeWindow プロパティが、HTML ウィンドウで定義されています。このプロパティを使用して、ここで説明する NativeWindow のプロパティ、メソッドおよびイベントにアクセスできます。

注意：また、JavaScript の Window オブジェクトには、moveTo() や close() など、コンテナウィンドウのスクリプトを作成するためのメソッドもあります。オーバーラップするメソッドを使用できる場合は、どれでも便利なメソッドを使用してかまいません。

Flex フレームワークウィンドウ
Flex Framework では、固有のウィンドウコンポーネントが定義されています。これらのコンポーネント（mx:WindowedApplication と mx:Window）はフレームワークの外部では使用できず、したがって HTML ベースの AIR アプリケーションでは使用できません。

初期アプリケーションウィンドウ
アプリケーションの最初のウィンドウは、AIR によって自動的に作成されます。AIR は、アプリケーション記述ファイルの initialWindow エレメントで指定されているパラメーターを使用して、ウィンドウのプロパティとコンテンツを設定します。

ルートコンテンツが SWF ファイルの場合、AIR は NativeWindow インスタンスを作成して、SWF ファイルを読み込み、それをウィンドウステージに追加します。ルートコンテンツが HTML ファイルの場合、AIR は HTML ウィンドウを作成して HTML を読み込みます。

ネイティブウィンドウクラス

Adobe AIR 1.0 およびそれ以降

ネイティブウィンドウ API には、次のクラスが含まれます。

パッケージ	クラス
flash.display	• NativeWindow • NativeWindowInitOptions • NativeWindowDisplayState • NativeWindowResize • NativeWindowSystemChrome • NativeWindowType
flash.events	• NativeWindowBoundsEvent • NativeWindowDisplayStateEvent

ネイティブウィンドウのイベントフロー

Adobe AIR 1.0 およびそれ以降

ネイティブウィンドウは、イベントを送出して、重要な変更が発生しようとしていること、または既に発生したことを、関係するコンポーネントに通知します。多くのウィンドウ関連イベントは、ペアで送出されます。最初のイベントは、変更が発生しようとしていることを警告します。2 番目のイベントは、変更が行われたことを通知します。警告イベントはキャンセルできますが、通知イベントはキャンセルできません。次のシーケンスは、ユーザーがウィンドウの最大化ボタンをクリックすると発生するイベントのフローを示したものです。

1. NativeWindow オブジェクトが、displayStateChanging イベントを送出します。
2. 登録済みのどのリスナーもイベントをキャンセルしない場合、ウィンドウは最大化します。
3. NativeWindow オブジェクトが、displayStateChange イベントを送出します。

   さらに、NativeWindow オブジェクトはウィンドウのサイズと位置に関連する変更についてのイベントも送出します。これらの関連する変更については、警告イベントは送出されません。次のような関連イベントがあります。

   a 最大化操作によりウィンドウの左上隅の位置が移動した場合、move イベントが送出されます。

   b 最大化操作によりウィンドウのサイズが変更された場合、resize イベントが送出されます。

   NativeWindow オブジェクトは、ウィンドウを最小化する、元のサイズに戻す、閉じる、移動する、サイズ変更するときも、同様のシーケンスでイベントを送出します。

   警告イベントは、ウィンドウクロムまたはその他のオペレーティングシステムによって制御されるメカニズムを通して変更が開始されるときにのみ、送出されます。ウィンドウメソッドを呼び出してウィンドウのサイズ、位置、表示状態を変更すると、ウィンドウは変更を通知するイベントのみを送出します。必要な場合は、ウィンドウの dispatchEvent() メソッドを使用して警告イベントを送出した後、警告イベントがキャンセルされたかどうかを確認してから、変更を継続できます。

   ウィンドウ API のクラス、メソッド、プロパティおよびイベントについて詳しくは、『HTML 開発者用 Adobe AIR API リファレンスガイド』を参照してください。

ネイティブウィンドウのスタイルと動作を制御するプロパティ

Flash Player 9 以降、Adobe AIR 1.0 以降

ウィンドウの基本的な外観と動作は、次のプロパティが制御します。

- type
- systemChrome
- transparent
- owner

ウィンドウを作成するときは、ウィンドウコンストラクターに渡す NativeWindowInitOptions オブジェクトで、これらのプロパティを設定します。AIR は、初期アプリケーションウィンドウのプロパティを、アプリケーション記述子から読み取ります（ただし、type プロパティはアプリケーション記述子では設定できず、常に normal に設定されます）。ウィンドウの作成後に、プロパティを変更することはできません。

これらのプロパティの設定の中には、同時に指定できないものがあります。例えば、transparent が true のとき、または type が lightweight のときには、systemChrome を standard に設定することはできません。

ウィンドウタイプ

Adobe AIR 1.0 およびそれ以降

AIR のウィンドウタイプでは、ネイティブオペレーティングシステムの chrome 属性と visibility 属性を結合して、ウィンドウの 3 つの機能タイプを作成します。コードでタイプ名を参照するには、NativeWindowType クラスで定義されている定数を使用します。AIR には次のウィンドウタイプがあります。

型	説明
normal	一般的なウィンドウです。normal ウィンドウは、フルサイズスタイルのクロムを使用し、Windows のタスクバーおよび Mac OS X のウィンドウメニューに表示されます。
utility	ツールパレットです。utility ウィンドウはスリムなバージョンのシステムクロムを使用し、Windows のタスクバーおよび Mac OS X のウィンドウメニューには表示されません。
lightweight	lightweight ウィンドウはクロムを持たず、Windows のタスクバーおよび Mac OS-X のウィンドウメニューには表示されません。また、lightweight ウィンドウには、Windows のシステムメニュー（Alt + Space）がありません。lightweight ウィンドウは、通知バブルや、短時間だけ表示される領域を開くコンボボックスなどのコントロールに適しています。lightweight の type を使用するときは、systemChrome を none に設定する必要があります。

ウィンドウクロム

Adobe AIR 1.0 およびそれ以降

ウィンドウクロムは、ユーザーがデスクトップ環境内のウィンドウを操作できるようにする一連のコントロールです。クロムエレメントには、タイトルバー、タイトルバーボタン、ボーダー、サイズ変更グリップなどがあります。

システムクロム

systemChrome プロパティには、standard または none を設定できます。ユーザーのオペレーティングシステムによって作成およびスタイル設定される標準コントロールのセットをウィンドウで使用するには、standard システムクロムを選択します。ウィンドウに独自のクロムを使用するには、none を選択します。コードでシステムクロムの設定を参照するには、NativeWindowSystemChrome クラスで定義されている定数を使用します。

システムクロムは、システムによって管理されます。アプリケーションでは、コントロール自体に直接アクセスすることはできませんが、コントロールが使用されるときに送出されるイベントには対応できます。ウィンドウに標準のクロムを使用する場合は、transparent プロパティを false に設定し、type プロパティを normal または utility に設定する必要があります。

カスタムクロム

システムクロムを使用しないでウィンドウを作成する場合は、独自のクロムコントロールを追加して、ユーザーとウィンドウの間の対話を処理する必要があります。また、透明で非矩形のウィンドウを作成してもかまいません。

ウィンドウの透明度

Adobe AIR 1.0 およびそれ以降

ウィンドウとデスクトップまたは他のウィンドウとのアルファブレンドを可能にするには、transparent プロパティを true に設定します。transparent プロパティは、ウィンドウを作成する前に設定する必要があり、変更することはできません。

透明なウィンドウには、デフォルトの背景はありません。アプリケーションによって描画されたオブジェクトを含まないウィンドウ領域は表示されません。表示されたオブジェクトのアルファ設定が 1 未満の場合は、同じウィンドウ内の他の表示オブジェクト、他のウィンドウ、デスクトップなど、オブジェクトの下にあるすべてのものが透けて見えます。

ボーダーが不規則な形のウィンドウ、または「フェードアウト」するウィンドウや不可視のウィンドウを作成するには、透明なウィンドウが便利です。ただし、大きいアルファブレンド領域のレンダリングには時間がかかる場合があるので、この効果は控えめに使用する必要があります。

重要：Linux の場合、マウスイベントは完全に透明なピクセルを通過できません。完全に透明で面積が広いウィンドウを作成すると、見えないウィンドウによってユーザーの操作がブロックされ、デスクトップ上にある他のウィンドウやアイテムがアクセス不能になるため、望ましくありません。Mac OS X および Windows の場合、マウスイベントは完全に透明なピクセルを通過します。

透明度は、システムクロムを使用するウィンドウでは使用できません。さらに、HTML 内の SWF コンテンツと PDF コンテンツは、透明ウィンドウに表示されない場合があります。詳しくは、49 ページの「HTML ページに SWF コンテンツまたは PDF コンテンツを読み込む場合の考慮事項」を参照してください。

ウィンドウの透明度が使用可能かどうかは、静的な NativeWindow.supportsTransparency プロパティによって報告されます。透明度がサポートされていない場合、アプリケーションの背景は黒になります。この場合、アプリケーションの透明領域は不透明の黒として表示されます。このプロパティがテストで false になった場合のために、フォールバックを用意することをお勧めします。例えば、警告ダイアログをユーザーに表示するか、矩形の不透明なユーザーインターフェイスを表示することができます。

Windows および Macintosh オペレーティングシステムでは、常に透明度がサポートされます。Linux オペレーティングシステムでのサポートには、合成ウィンドウマネージャーが必要ですが、合成ウィンドウマネージャーがアクティブな場合でも、ユーザーの表示オプションまたはハードウェア構成により、透明度は使用できません。

HTML アプリケーションウィンドウでの透明度

Adobe AIR 1.0 およびそれ以降

デフォルトでは、HTML ウィンドウおよび HTMLLoader オブジェクトに表示される HTML コンテンツの背景は、コンテナウィンドウが透明であっても不透明です。HTML コンテンツに表示されるデフォルトの背景をオフにするには、paintsDefaultBackground プロパティを false に設定します。次の例では、HTMLLoader を作成し、デフォルトの背景をオフにしています。

```
var htmlView:HTMLLoader = new HTMLLoader();
htmlView.paintsDefaultBackground = false;
```

この例では、JavaScript を使用して HTML ウィンドウのデフォルトの背景をオフにしています。

```
window.htmlLoader.paintsDefaultBackground = false;
```

HTML ドキュメント内のエレメントに背景色を設定すると、そのエレメントの背景は透明ではなくなります。透明度（または不透明度）の値の部分的な設定は、サポートされていません。ただし、透明な PNG 形式のグラフィックをページまたはページエレメントの背景として使用することで、同じような視覚効果を得ることができます。

ウィンドウの所有権

1 つのウィンドウは、1 つ以上の他のウィンドウを所有できます。所有されているウィンドウは、常にマスターウィンドウの前面に表示され、マスターウィンドウと共に最小化と復元が実行され、マスターウィンドウが閉じられると一緒に閉じられます。ウィンドウの所有権を他のウィンドウに転送したり、削除したりすることはできません。複数のウィンドウが同じ 1 つのウィンドウを所有することはできませんが、1 つのウィンドウが複数のウィンドウを所有することはできます。

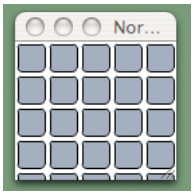
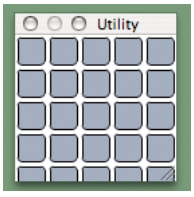
ウィンドウの所有機能を使用することで、ツールパレットやダイアログに使用されるウィンドウの管理が簡単になります。例えば、ドキュメントウィンドウに関連する保存ダイアログを表示する場合、ドキュメントウィンドウを保存ダイアログの所有者として設定しておくことによって、自動的にドキュメントウィンドウの前面にダイアログを表示し続けることができます。

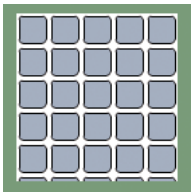
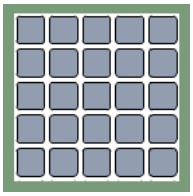
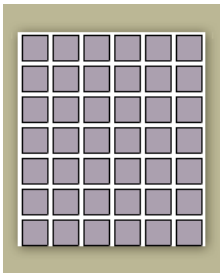
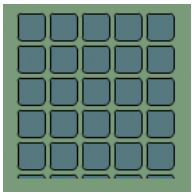
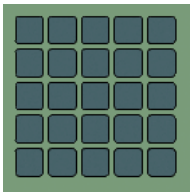
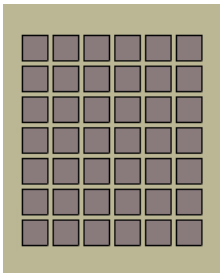
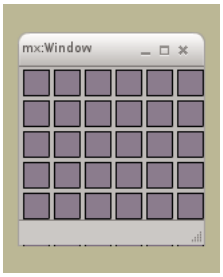
- NativeWindow.owner
- Christian Cantrell：Owned windows in AIR 2.6

表示ウィンドウカタログ

Adobe AIR 1.0 およびそれ以降

次の表に、異なるウィンドウプロパティ設定の組み合わせが Mac OS X、Windows および Linux オペレーティングシステムでどのような視覚効果になるかを示します。

ウィンドウ設定	Mac OS X	Microsoft Windows	Linux*
Type：normal SystemChrome：standard Transparent：false			
Type：utility SystemChrome：standard Transparent：false			

ウィンドウ設定	Mac OS X	Microsoft Windows	Linux*
Type：Any SystemChrome ：none Transparent：false			
Type：Any SystemChrome ：none Transparent：true			
mxWindowedApplication または mx:Window Type：Any SystemChrome ：none Transparent：true			

*Ubuntu と Compiz ウィンドウマネージャー

注意：Mac OS X ツールバー、Mac OS X プロキシアイコン、Windows タイトルバーアイコン、代替システムクロムの各システムクロムエレメントは、AIR ではサポートされていません。

ウィンドウの作成

Adobe AIR 1.0 およびそれ以降

アプリケーションの最初のウィンドウは AIR によって自動的に作成されますが、必要に応じて任意の追加ウィンドウを作成できます。ネイティブウィンドウを作成するには、NativeWindow コンストラクターメソッドを使用します。

HTML ウィンドウを作成するには、HTMLLoader の createRootWindow() メソッドを使用するか、または HTML ドキュメントから JavaScript の window.open() メソッドを呼び出します。作成されたウィンドウは、表示リストに HTMLLoader オブジェクトを含む NativeWindow オブジェクトとなります。HTMLLoader オブジェクトは、ウィンドウの HTML および JavaScript コンテンツを解釈して表示します。window.nativeWindow プロパティを使用して、基になる NativeWindow オブジェクトのプロパティに JavaScript からアクセスできます（このプロパティには、AIR アプリケーションサンドボックスで実行されているコードでのみアクセスできます）。

ウィンドウを初期化するときは（最初のアプリケーションウィンドウを含む）、ウィンドウを非表示の状態で作成し、コンテンツのロードまたはグラフィックの更新を実行した後で、ウィンドウを表示することを検討する必要があります。このシーケンスにより、目障りな視覚的変化がユーザーに表示されないようにすることができます。アプリケーションの初期ウィンドウが非表示の状態で作成されるようにするには、アプリケーション記述ファイルで <visible>false</visible> タグを指定します（この値はデフォルトで false になっているので、省略しても構いません）。新しい NativeWindows はデフォルトで非表示になります。HTMLLoader の createRootWindow() メソッドを使用して HTML ウィンドウを作成する場合は、visible 引数を false に設定します。ウィンドウを表示するには、NativeWindow の activate() メソッドを呼び出すか、visible プロパティを true に設定します。

ウィンドウの初期化プロパティの指定

Adobe AIR 1.0 およびそれ以降

デスクトップウィンドウの作成後に、ネイティブウィンドウの初期化プロパティを変更することはできません。このような不変のプロパティとそのデフォルト値は次のとおりです。

プロパティ	デフォルト値
systemChrome	standard
type	normal
transparent	false
owner	null
maximizable	true
minimizable	true
resizable	true

AIR によって作成される初期ウィンドウのプロパティは、アプリケーション記述ファイルで設定します。AIR アプリケーションのメインウィンドウのタイプは、常に **normal** です（記述ファイルでは、visible、width、height などの他のウィンドウプロパティを指定することができますが、これらのプロパティはいつでも変更できます）。

アプリケーションで作成する他のネイティブウィンドウおよび HTML ウィンドウのプロパティは、NativeWindowInitOptions クラスを使用して設定します。ウィンドウを作成するときは、ウィンドウプロパティを指定する NativeWindowInitOptions オブジェクトを、NativeWindow コンストラクター関数または HTMLLoader createRootWindow() メソッドに渡す必要があります。

次のコードでは、UTILITY ウィンドウの NativeWindowInitOptions オブジェクトを作成します。

```
var options = new air.NativeWindowInitOptions();
options.systemChrome = air.NativeWindowSystemChrome.STANDARD;
options.type = air.NativeWindowType.UTILITY
options.transparent = false;
options.resizable = false;
options.maximizable = false;
```

transparent が **true** のとき、または **type** が lightweight のときに、systemChrome を standard に設定することはできません。

注意： JavaScript の window.open() 関数で作成するウィンドウの初期化プロパティは設定できません。ただし、独自の HTMLHost クラスを実装することで、これらのウィンドウの作成方法をオーバーライドすることはできます。詳しくは、60 ページの「window.open() の JavaScript 呼び出しの処理」を参照してください。

初期アプリケーションウィンドウの作成

Adobe AIR 1.0 およびそれ以降

アプリケーションの初期ウィンドウには、標準の HTML ページを使用します。このページは、アプリケーションのインストールディレクトリから読み込まれて、アプリケーションサンドボックスに配置されます。ページは、アプリケーションの初期エントリポイントとして機能します。

アプリケーションを起動すると、AIR はウィンドウを作成し、HTML 環境を設定して、HTML ページを読み込みます。スクリプトを解析したり、エレメントを HTML DOM に追加したりする前に、AIR は、runtime、htmlLoader、nativeWindow の各プロパティを、JavaScript の Window オブジェクトに追加します。これらのプロパティを使用して、JavaScript からランタイムクラスにアクセスできます。nativeWindow プロパティを使用して、デスクトップウィンドウのプロパティやメソッドに直接アクセスできます。

次の例では、HTML で作成された AIR アプリケーションのメインページの基本的な構造を示します。このページは、JavaScript ウィンドウの load イベントを待機してから、ネイティブウィンドウを表示します。

```
<html>
    <head>
        <script language="javascript" type="text/javascript" src="AIRAliases.js"></script>
        <script language="javascript">
            window.onload=init;

            function init(){
                window.nativeWindow.activate();
            }
        </script>
    </head>
    <body></body>
</html>
```

注意：この例で参照されている AIRAliases.js ファイルは、よく使用されるビルトイン AIR クラスに対して便利なエイリアス変数を定義するスクリプトファイルです。このファイルは、AIR SDK の frameworks ディレクトリ内にあります。

NativeWindow の作成

Adobe AIR 1.0 およびそれ以降

NativeWindow を作成するには、NativeWindowInitOptions オブジェクトを NativeWindow コンストラクターに渡します。

```
var options = new air.NativeWindowInitOptions();
options.systemChrome = air.NativeWindowSystemChrome.STANDARD;
options.transparent = false;
var newWindow = new air.NativeWindow(options);
```

ウィンドウは、visible プロパティを true に設定すると、または activate() メソッドを呼び出すと表示されます。

ウィンドウが作成されたなら、ステージプロパティと Flash 表示リストの技法を使用して、ウィンドウのプロパティを初期化し、コンテンツをウィンドウに読み込むことができます。

ほとんどの場合は、新しいネイティブウィンドウのステージプロパティ scaleMode を noScale に設定する必要があります（StageScaleMode.NO_SCALE 定数を使用）。Flash スケールモードは、アプリケーション作成者がアプリケーション表示領域の縦横比を事前に知ることができない状況のために用意されています。スケールモードを使用することで、作成者は、コンテンツのクリッピング、拡大や縮小、または空き領域でのパディングから、最も影響の小さいものを選択できます。表示領域は AIR（ウィンドウフレーム）で制御するので、悪影響を与えることなく、ウィンドウのサイズをコンテンツに、またはコンテンツのサイズをウィンドウに合わせることができます。

HTML ウィンドウのスケールモードは、noScale に自動的に設定されます。

注意：現在のオペレーティングシステムで許される最大および最小のウィンドウサイズを判別するには、次の静的 NativeWindow プロパティを使用します。

```
var maxOSSize = air.NativeWindow.systemMaxSize;
var minOSSize = air.NativeWindow.systemMinSize;
```

HTML ウィンドウの作成

Adobe AIR 1.0 およびそれ以降

HTML ウィンドウを作成するには、JavaScript で Window.open() メソッドを呼び出すか、または AIR で HTMLLoader クラスの createRootWindow() メソッドを呼び出します。

セキュリティサンドボックス内の HTML コンテンツは、JavaScript の標準の Window.open() メソッドを使用できます。コンテンツがアプリケーションサンドボックスの外部で実行している場合は、マウスクリックやキー押下などのユーザー操作に対して呼び出すことができるのは、open() メソッドだけです。open() を呼び出すと、指定した URL にあるコンテンツを表示するために、システムクロムを使用するウィンドウが作成されます。次に、例を示します。

```
newWindow = window.open("xmpl.html", "logWindow", "height=600, width=400, top=10, left=10");
```

注意：ActionScript で HTMLHost クラスを拡張し、JavaScript の window.open() 関数で作成されるウィンドウをカスタマイズできます。53 ページの「HTMLHost クラスの拡張について」を参照してください。

アプリケーションセキュリティサンドボックス内のコンテンツは、ウィンドウ作成用のさらに強力なメソッドである HTMLLoader.createRootWindow() にアクセスできます。このメソッドを使用すると、新規ウィンドウ用のすべての作成オプションを指定できます。例えば、次の JavaScript コードでは、システムクロムを使用しない最小タイプウィンドウを、300x400 ピクセルのサイズで作成しています。

```
 var options = new air.NativeWindowInitOptions();
options.systemChrome = "none";
options.type = "lightweight";

var windowBounds = new air.Rectangle(200,250,300,400);
newHTMLLoader = air.HTMLLoader.createRootWindow(true, options, true, windowBounds);
newHTMLLoader.load(new air.URLRequest("xmpl.html"));
```

注意：新しいウィンドウによって読み込まれたコンテンツがアプリケーションセキュリティサンドボックスの外部にある場合は、ウィンドウオブジェクトには AIR プロパティの runtime、nativeWindow、または htmlLoader はありません。

透明ウィンドウを作成する場合は、そのウィンドウに読み込まれた HTML に埋め込まれている SWF コンテンツは、常に表示されるとは限りません。SWF ファイルを参照するために使用されるオブジェクトまたは埋め込みタグの wmode パラメーターを、opaque または transparent のどちらかに設定する必要があります。wmode のデフォルト値は window なので、デフォルトでは、SWF コンテンツは透明ウィンドウに表示されません。どの wmode 値が設定されていようと、PDF コンテンツを透明ウィンドウに表示することはできません（AIR 1.5.2 より前のバージョンでは、SWF コンテンツも透明ウィンドウに表示することはできませんでした）。

createRootWindow() メソッドで作成したウィンドウは、それを開いたウィンドウから独立したままになります。JavaScript Window オブジェクトの parent プロパティと opener プロパティは、null です。新しいウィンドウを開いたウィンドウは、createRootWindow() 関数から返る HTMLLoader の参照を使用して、新しいウィンドウの Window オブジェクトにアクセスできます。前記の例のコンテキストでは、ステートメント newHTMLLoader.window は作成されたウィンドウの JavaScript Window オブジェクトを参照します。

注意：createRootWindow() 関数は、JavaScript と ActionScript のどちらからでも呼び出すことができます。

mx:Window の作成

Adobe AIR 1.0 およびそれ以降

mx:Window を作成するには、mx:Window をルートタグとして使用する MXML ファイルを作成するか、または Window クラスのコンストラクターを直接呼び出します。

次の例では、Window コンストラクターを呼び出して、mx:Window を作成および表示しています。

```
 var newWindow:Window = new Window();
newWindow.systemChrome = NativeWindowSystemChrome.NONE;
newWindow.transparent = true;
newWindow.title = "New Window";
newWindow.width = 200;
newWindow.height = 200;
newWindow.open(true);
```

ウィンドウへのコンテンツの追加

Adobe AIR 1.0 およびそれ以降

AIR ウィンドウにコンテンツを追加する方法は、ウィンドウのタイプによって異なります。例えば、MXML と HTML では、ウィンドウの基本的なコンテンツを宣言的に定義できます。リソースをアプリケーションの SWF ファイルに埋め込むことも、別のアプリケーションファイルからリソースを読み込むこともできます。Flex、Flash、HTML のコンテンツはいずれも、同時進行で作成し、ウィンドウに動的に追加できます。

SWF コンテンツ、または JavaScript を含む HTML コンテンツを読み込むときは、AIR のセキュリティモデルを考慮する必要があります。アプリケーションセキュリティサンドボックス内のコンテンツ、つまりアプリケーションでインストールされて、app: URL スキームで読み込み可能なコンテンツは、すべての AIR API にアクセスできる完全な権限を持っています。このサンドボックスの外部から読み込まれるコンテンツは、AIR API にアクセスできません。アプリケーションサンドボックスの外部にある JavaScript コンテンツは、JavaScript Window オブジェクトの runtime、nativeWindow、または htmlLoader プロパティを使用できません。

安全なクロススクリプトを可能にするには、サンドボックスブリッジを使用して、アプリケーションコンテンツと非アプリケーションコンテンツの間に制限されたインターフェイスを提供できます。HTML コンテンツでは、アプリケーションのページを非アプリケーションサンドボックスにマップし、そのページのコードが外部コンテンツをクロススクリプトできるようにすることもできます。68 ページの「AIR のセキュリティ」を参照してください。

NativeWindow への HTML コンテンツの読み込み

HTML コンテンツを NativeWindow に読み込むには、HTMLLoader オブジェクトをウィンドウステージに追加してから HTML コンテンツを HTMLLoader に読み込むか、または HTMLLoader.createRootWindow() メソッドを使用して HTMLLoader オブジェクトを既に含んでいるウィンドウを作成します。次の例では、ネイティブウィンドウのステージ上の 300 × 500 ピクセルの表示領域に、HTML コンテンツを表示します。

```
 //newWindow is a NativeWindow instance
var htmlView:HTMLLoader = new HTMLLoader();
htmlView.width = 300;
htmlView.height = 500;

//set the stage so display objects are added to the top-left and not scaled
newWindow.stage.align = "TL";
newWindow.stage.scaleMode = "noScale";
newWindow.stage.addChild( htmlView );

//urlString is the URL of the HTML page to load
htmlView.load( new URLRequest(urlString) );
```

HTML ページを Flex アプリケーションに読み込むには、Flex HTML コンポーネントを使用します。

ウィンドウで透明度が使用されている場合（ウィンドウの transparent プロパティが true の場合）は、SWF ファイルを参照するために使用されるオブジェクトまたは埋め込みタグの wmode パラメーターが opaque または transparent のどちらかに設定されていない限り、HTML ファイル内の SWF コンテンツは表示されません。wmode 値のデフォルトは window なので、デフォルトでは、SWF コンテンツは透明ウィンドウには表示されません。どの wmode 値が使用されていようと、PDF コンテンツは透明ウィンドウには表示されません。

また、HTMLLoader コントロールが拡大／縮小または回転されている場合、または HTMLLoader の alpha プロパティが 1.0 以外の値に設定されている場合、SWF コンテンツと PDF コンテンツのどちらも表示されません。

HTML ウィンドウ上のオーバーレイとしての SWF コンテンツの追加

HTML ウィンドウは NativeWindow インスタンス内に含まれるので、表示リストでは HTML レイヤーの上または下のどちらにでも、Flash 表示オブジェクトを追加できます。

HTML レイヤーの上位に表示オブジェクトを追加するには、window.nativeWindow.stage プロパティの addChild() メソッドを使用します。addChild() メソッドは、ウィンドウ内の既存のすべてのコンテンツより上のレイヤーに、コンテンツを追加します。

HTML レイヤーより下位に表示オブジェクトを追加するには、window.nativeWindow.stage プロパティの addChildAt() メソッドを使用し、index パラメーターに値 0 を渡します。0 インデックスにオブジェクトを配置すると、HTML の表示を含む既存のコンテンツが 1 レイヤーだけ上に移動し、新しいコンテンツが最下位に挿入されます。HTML ページより下位のレイヤーにあるコンテンツを表示するには、HTMLlLoader オブジェクトの paintsDefaultBackground プロパティを false に設定する必要があります。また、ページのエレメントで背景色が設定されているものはすべて、透明になりません。例えば、ページの body エレメントに背景色を設定すると、ページのすべてのものが透明ではなくなります。

次の例では、Flash 表示オブジェクトをオーバーレイおよびアンダーレイとして HTML ページに追加する方法を示します。この例では、単純な形のオブジェクトを 2 つ作成し、1 つは HTML コンテンツより下位に、もう 1 つは上位に追加します。また、enterFrame イベントに基づいて図形の位置も更新します。

```
 <html>
<head>
<title>Bouncers</title>
<script src="AIRAliases.js" type="text/javascript"></script>
<script language="JavaScript" type="text/javascript">
air.Shape = window.runtime.flash.display.Shape;

function Bouncer(radius, color){
    this.radius = radius;
    this.color = color;

    //velocity
    this.vX = -1.3;
    this.vY = -1;

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

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


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

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

        //Keep the sprite within the window
        if( this.shape.x - this.radius < 0){
            this.vX = -this.vX;
        }
        if( this.shape.y - this.radius < 0){
            this.vY = -this.vY;
        }
        if( this.shape.x  + this.radius > window.nativeWindow.stage.stageWidth){
            this.vX = -this.vX;
        }
        if( this.shape.y  + this.radius > window.nativeWindow.stage.stageHeight){
            this.vY = -this.vY;
        }

    };
}

function init(){
    //turn off the default HTML background
    window.htmlLoader.paintsDefaultBackground = false;
    var bottom = new Bouncer(60,0xff2233);
    var top = new Bouncer(30,0x2441ff);

    //listen for the enterFrame event
    window.htmlLoader.addEventListener("enterFrame",function(evt){
        bottom.update();
        top.update();
    });

    //add the bouncing shapes to the window stage
    window.nativeWindow.stage.addChildAt(bottom.shape,0);
    window.nativeWindow.stage.addChild(top.shape);
}
</script>
<body onload="init();">
<h1>de Finibus Bonorum et Malorum</h1>
<p>Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium
doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis
et quasi architecto beatae vitae dicta sunt explicabo.</p>
<p style="background-color:#FFFF00; color:#660000;">This paragraph has a background color.</p>
<p>At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis
praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias
excepturi sint occaecati cupiditate non provident, similique sunt in culpa qui
officia deserunt mollitia animi, id est laborum et dolorum fuga.</p>
</body>
</html>
```

この例では、AIR において JavaScript と ActionScript の間にある境界を越える、若干高度な技法の初歩的な概要について説明しています。ActionScript の表示オブジェクトの使用に関する知識がない場合は、『**ActionScript 3.0** 開発ガイド』の表示のプログラミングを参照してください。

注意：JavaScript Window オブジェクトの runtime、nativeWindow、htmlLoader の各プロパティにアクセスするには、HTML ページをアプリケーションディレクトリから読み込む必要があります。これは、HTML ベースのアプリケーションのルートページには常に該当しますが、他のコンテンツについては該当しない場合があります。また、アプリケーションサンドボックス内であってもフレームに読み込まれたドキュメントはこれらのプロパティを受け取りませんが、親ドキュメントのプロパティにはアクセスできます。

例：ネイティブウィンドウの作成

Adobe AIR 1.0 およびそれ以降

次の例では、ネイティブウィンドウの作成方法を示します。

```
function createNativeWindow() {
    //create the init options
    var options = new air.NativeWindowInitOptions();
    options.transparent = false;
    options.systemChrome = air.NativeWindowSystemChrome.STANDARD;
    options.type = air.NativeWindowType.NORMAL;

    //create the window
    var newWindow = new air.NativeWindow(options);
    newWindow.title = "A title";
    newWindow.width = 600;
    newWindow.height = 400;

    //activate and show the new window
    newWindow.activate();
}
```

ウィンドウの管理

Adobe AIR 1.0 およびそれ以降

デスクトップウィンドウの外観、動作、ライフサイクルを管理するには、NativeWindow クラスのプロパティとメソッドを使用します。

注意：Flex フレームワークを使用する場合は、一般にフレームワーククラスを使用してウィンドウ動作を管理する方が適切です。NativeWindow のほとんどのプロパティおよびメソッドには、mx:WindowedApplication および mx:Window クラスを使用してアクセスできます。

NativeWindow インスタンスの取得

Adobe AIR 1.0 およびそれ以降

ウィンドウを操作するには、最初にウィンドウインスタンスを取得する必要があります。ウィンドウインスタンスは、次のいずれかの場所から取得できます。

- ウィンドウの作成に使用されるネイティブウィンドウコンストラクター：

  ```
  var nativeWin = new air.NativeWindow(initOptions);
  ```

- ウィンドウ内の表示オブジェクトの stage プロパティ：

  ```
  var nativeWin = window.htmlLoader.stage.nativeWindow;
  ```

- ウィンドウによって送出されるネイティブウィンドウイベントの target プロパティ：

  ```
  function onNativeWindowEvent(event)
  {
      var nativeWin = event.target;
  }
  ```

- ウィンドウに表示される HTML ページの nativeWindow プロパティ：

  ```
  var nativeWin = window.nativeWindow;
  ```

- NativeApplication オブジェクトの activeWindow プロパティと openedWindows プロパティ：

```
var win = NativeApplication.nativeApplication.activeWindow;
var firstWindow = NativeApplication.nativeApplication.openedWindows[0];
```

  NativeApplication.nativeApplication.activeWindow は、アプリケーションのアクティブなウィンドウを示しています（ただし、アクティブなウィンドウが AIR アプリケーションのウィンドウでない場合は、null を返します）。
  NativeApplication.nativeApplication.openedWindows 配列には、AIR アプリケーション内で閉じられていないすべてのウィンドウが格納されています。

Flex の mx:WindowedApplication オブジェクトと mx:Window オブジェクトは表示オブジェクトなので、次に示すように、stage プロパティを使用して、MXML ファイル内のアプリケーションウィンドウを簡単に参照できます。

```
 <?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" applicationComplete="init();">
    <mx:Script>
        <![CDATA[
            import flash.display.NativeWindow;

            public function init():void{
                var appWindow:NativeWindow = this.stage.nativeWindow;
                //set window properties
                appWindow.visible = true;
            }
        ]]>
    </mx:Script>
</WindowedApplication
```

注意：WindowedApplication コンポーネントまたは Window コンポーネントが Flex フレームワークによってウィンドウステージに追加されるまで、コンポーネントの stage プロパティは null です。この動作は、Flex Application コンポーネントの動作と同じですが、creationComplete などの WindowedApplication および Window コンポーネントの初期化サイクルの早い段階で発生するイベントに対するリスナーでは、ステージまたは NativeWindow インスタンスにアクセスできないことを意味します。applicationComplete イベントが送出された時点では、ステージおよび NativeWindow インスタンスにアクセスしても安全です。

ウィンドウのアクティブ化、表示、非表示

Adobe AIR 1.0 およびそれ以降

ウィンドウをアクティブ化するには、NativeWindow の activate() メソッドを呼び出します。ウィンドウをアクティブ化すると、ウィンドウが前面になり、キーボードとマウスのフォーカスが設定され、必要な場合は、ウィンドウを復元するか、または visible プロパティを true に設定することでウィンドウが表示されるようになります。あるウィンドウをアクティブ化しても、アプリケーション内での他のウィンドウの順序は変わりません。activate() メソッドを呼び出すと、ウィンドウは activate イベントを送出します。

非表示になっているウィンドウをアクティブ化しないで表示するには、visible プロパティを true に設定します。これによりウィンドウは前面になりますが、フォーカスがウィンドウに割り当てられることはありません。

ウィンドウを非表示にするには、visible プロパティを false に設定します。ウィンドウを非表示にすると、ウィンドウ、関連するタスクバーアイコン、および MacOS X ではウィンドウメニューのエントリが表示されなくなります。

ウィンドウの可視性を変更すると、このウィンドウが所有するすべてのウィンドウの可視性も同様に変更されます。例えば、ウィンドウを非表示にすると、そのウィンドウが所有しているウィンドウもすべて非表示になります。

注意：Mac OS X では、ドックのウィンドウ部分にアイコンがある最小化されたウィンドウを完全に非表示にすることはできません。最小化されたウィンドウの visible プロパティを false に設定しても、そのウィンドウのドックアイコンは表示されています。ユーザーがアイコンをクリックすると、ウィンドウは復元されて表示されるようになります。

ウィンドウの表示順序の変更

Adobe AIR 1.0 およびそれ以降

AIR では、ウィンドウの表示順序を複数の方法で直接変更できます。ウィンドウの表示順序の前面は背面に移動したり、ウィンドウを別のウィンドウの前や後に移動したりできます。同時に、ウィンドウをアクティブ化することでウィンドウの順序を変更することもできます。

alwaysInFront プロパティを true に設定することで、あるウィンドウを他のウィンドウより前面に保つことができます。このように設定されたウィンドウが複数ある場合、同じ設定のウィンドウ間で表示順序が並べ替えられます。ただし、それらのウィンドウは、alwaysInFront が false に設定されたウィンドウよりは常に前面に配置されます。

また、最上位グループのウィンドウは、AIR アプリケーションがアクティブではない場合でも、他のアプリケーションのウィンドウより前面に表示されます。この動作はユーザーにとっては迷惑である場合があるので、alwaysInFront を true に設定するのは、それが必要であって適切である場合にのみ行う必要があります。例えば、次のような場合には使用が正当化されます。

- ツールチップ、ポップアップリスト、カスタムメニュー、コンボボックスなどのコントロール用の一時的なポップアップウィンドウ。これらのウィンドウはフォーカスを失ったときに閉じる必要があるので、別のウィンドウの表示の邪魔になるのを避けることができます。
- 特に緊急を要するエラーメッセージと警告。ユーザーが直ちに応答しないと元に戻すことのできない変更が発生する可能性がある場合は、警告ウィンドウを最前面に移動してもかまいません。ただし、ほとんどのエラーと警告は、通常のウィンドウ表示順序で処理できます。
- 短時間だけ表示されるトーストスタイルのウィンドウ。

注意：AIR では、alwaysInFront プロパティの適切な使用は強制されません。ただし、ユーザーのワークフローを妨害するようなアプリケーションは、ごみ箱行きになることでしょう。

ウィンドウが他のウィンドウを所有している場合、所有下にあるウィンドウは常に所有者のウィンドウより前に配置されるように並べ替えられます。他のウィンドウを所有するウィンドウで orderToFront() を呼び出したり、alwaysInFront を true に設定した場合、所有下にあるウィンドウも所有者ウィンドウと一緒に並べ替えられ、他のウィンドウの前面に配置されます。ただし、所有下にあるウィンドウは、所有者ウィンドウの前面に引き続き表示されます。

所有下にあるウィンドウからウィンドウの並べ替えメソッドを呼び出した場合、所有者ウィンドウが同じであるウィンドウ間では通常通り動作します。ただし、所有下にあるウィンドウのグループ全体が、そのグループ以外のウィンドウと比較され、並べ替えられることがあります。例えば、所有下にあるウィンドウから orderToFront() を呼び出すと、そのウィンドウ自身、その所有者、所有者が同じであるその他すべてのウィンドウが、ウィンドウ表示順序の前面に移動します。

NativeWindow クラスには、ウィンドウ間の相対的な表示順序を設定する次のプロパティとメソッドが用意されています。

メンバー	説明
alwaysInFront プロパティ	ウィンドウを最上位ウィンドウグループに表示するかどうかを指定します。 通常は、false が最善の設定です。値を false から true に変更すると、そのウィンドウは他のウィンドウの前面に移動します（ただし、アクティブにはなりません）。値を true から false に変更すると、そのウィンドウは最上位グループに残っているウィンドウの背後に移動しますが、他のウィンドウよりは前面になっています。プロパティをウィンドウの現在値と同じ値に設定すると、ウィンドウの表示順序は変わりません。 alwaysInFront 設定は、他のウィンドウに所有されているウィンドウに対しては効果はありません。
orderToFront()	ウィンドウを前面に移動します。
orderInFrontOf()	ウィンドウを特定のウィンドウのすぐ前面に表示します。

メンバー	説明
orderToBack()	ウィンドウを他のウィンドウの背後に移動します。
orderBehind()	ウィンドウを特定のウィンドウのすぐ背後に表示します。
activate()	ウィンドウを前面に移動します（同時に、ウィンドウを表示状態にしてフォーカスを割り当てます）。

注意：ウィンドウが非表示状態（visible が false）または最小化状態の場合は、表示順序メソッドを呼び出しても順序は変更されません。

Linux オペレーティングシステムでは、別のウィンドウマネージャーにより、ウィンドウの表示順序に関する別の規則が適用されます。

- 一部のウィンドウマネージャーでは、normal ウィンドウの前面にユーティリティウィンドウが常に表示されます。
- 一部のウィンドウマネージャーでは、alwaysInFront が true に設定されているフルスクリーンウィンドウが、alwaysInFront が true に設定されている他のウィンドウの前面に常に表示されます。

ウィンドウを閉じる

Adobe AIR 1.0 およびそれ以降

ウィンドウを閉じるには、NativeWindow.close() メソッドを使用します。

ウィンドウを閉じると、ウィンドウのコンテンツの読み込みは解除されますが、他のオブジェクトがそのコンテンツを参照している場合は、コンテンツオブジェクトは破棄されません。NativeWindow.close() メソッドは非同期に実行するので、ウィンドウに含まれるアプリケーションは、閉じる処理が行われている間も実行を続けます。閉じる操作が完了すると、close メソッドは close イベントを送出します。NativeWindow オブジェクトは技術的にはまだ有効な状態ですが、閉じられたウィンドウのほとんどのプロパティとメソッドは、アクセスすると IllegalOperationError が生成されます。閉じられたウィンドウを再び開くことはできません。ウィンドウが閉じられているかどうかを確認するには、ウィンドウの closed プロパティを調べます。単にウィンドウを非表示にするには、NativeWindow.visible プロパティを false に設定します。

Nativeapplication.autoExit プロパティが true の場合は（デフォルト）、最後のウィンドウが閉じるとアプリケーションは終了します。

所有下にあるウィンドウは、所有者が閉じられたときに一緒に閉じます。所有下にあるウィンドウからは closing イベントが送出されないため、閉じる処理を取り消すことはできません。close イベントは送出されます。

ウィンドウ操作のキャンセルの許可

Adobe AIR 1.0 およびそれ以降

ウィンドウがシステムクロムを使用している場合は、適切なイベントをリッスンし、イベントのデフォルト動作をキャンセルすることで、ユーザーによるウィンドウの操作をキャンセルできます。例えば、ユーザーがシステムクロムの閉じるボタンをクリックすると、closing イベントが送出されます。登録されているリスナーがイベントの preventDefault() メソッドを呼び出すと、ウィンドウは閉じません。

ウィンドウがシステムクロムを使用していない場合は、目的の変更が行われる前に変更の通知イベントが自動的に送出されることはありません。したがって、ウィンドウを閉じたり、ウィンドウの状態を変更したり、ウィンドウ境界プロパティを設定したりするメソッドを呼び出した場合、変更はキャンセルできません。ウィンドウの変更が行われる前にアプリケーションのコンポーネントに通知するには、アプリケーションのロジックで、ウィンドウの dispatchEvent() メソッドを使用して、関連する通知イベントを送出します。

例えば、次のロジックは、ウィンドウの閉じるボタンに対するキャンセル可能なイベントハンドラーを実装しています。

```
function onCloseCommand(event){
    var closingEvent = new air.Event(air.Event.CLOSING,true,true);
    dispatchEvent(closingEvent);
    if(!closingEvent.isDefaultPrevented()){
        win.close();
    }
}
```

イベントの preventDefault() メソッドがリスナーによって呼び出されると、dispatchEvent() メソッドは false を返します。ただし、このメソッドは他の理由でも false を返す場合があるので、isDefaultPrevented() メソッドを明示的に使用して、変更をキャンセルする必要があるかどうかを判定するのがよい方法です。

ウィンドウの最大化、最小化、復元

Adobe AIR 1.0 およびそれ以降

ウィンドウを最大化するには、NativeWindow の maximize() メソッドを使用します。

```
window.nativeWindow.maximize();
```

ウィンドウを最小化するには、NativeWindow の minimize() メソッドを使用します。

```
window.nativeWindow.minimize();
```

ウィンドウを復元するには（つまり、最大化または最小化されていたウィンドウを元のサイズに戻すには）、NativeWindow の restore() メソッドを使用します。

```
window.nativeWindow.restore();
```

所有下にあるウィンドウは、所有者ウィンドウが最小化または復元されたときに、一緒に最小化または復元されます。所有者の最小化に伴い最小化される際には、所有下にあるウィンドウからはイベントは送出されません。

注意：AIR ウィンドウを最大化したときの動作は、Mac OS X での標準の動作とは異なります。アプリケーションで定義されている「標準」サイズとユーザーが最後に設定したサイズの間で切り替えが行われるのではなく、AIR ウィンドウは、アプリケーションまたはユーザーが最後に設定したサイズと、画面で使用可能な領域全体との間で切り替わります。

Linux オペレーティングシステムでは、別のウィンドウマネージャーにより、ウィンドウの表示状態の設定に関する別の規則が適用されます。

- 一部のウィンドウマネージャーでは、ユーティリティウィンドウを最大化できません。
- ウィンドウに最大サイズが設定されている場合、一部のウィンドウではウィンドウを最大化できません。他のウィンドウマネージャーでは、表示状態が最大化に設定されますが、ウィンドウのサイズは変更されません。いずれの場合も、表示状態変更イベントは送出されません。
- 一部のウィンドウマネージャーは、ウィンドウの maximizable 設定または minimizable 設定を受け取りません。

注意：Linux では、ウィンドウプロパティは非同期で変更されます。プログラムのある行で表示状態を変更し、次の行で値を読み取る場合、読み取られた値には古い値が反映されます。すべてのプラットフォームで、表示状態が変更されると、NativeWindow オブジェクトから displayStateChange イベントが送出されます。ウィンドウの新しい状態に基づいて何らかのアクションを実行する必要がある場合、常に displayStateChange イベントハンドラーで行ってください。104 ページの「ウィンドウイベントのリッスン」を参照してください。

例：ウィンドウの最小化、最大化、復元、終了

Adobe AIR 1.0 およびそれ以降

次の短い HTML ページでは、NativeWindow の maximize()、minimize()、restore() および close() メソッドの使用方法を示します。

```
<html>
<head>
<title>Change Window Display State</title>
<script src="AIRAliases.js"/>
<script type="text/javascript">
     function onMaximize(){
        window.nativeWindow.maximize();
     }

     function onMinimize(){
        window.nativeWindow.minimize();
     }

     function onRestore(){
        window.nativeWindow.restore();
     }

     function onClose(){
        window.nativeWindow.close();
     }
</script>
</head>

<body>
     <h1>AIR window display state commands</h1>
        <button onClick="onMaximize()">Maximize</button>
        <button onClick="onMinimize()">Minimize</button>
        <button onClick="onRestore()">Restore</button>
    <button onClick="onClose()">Close</button>
</body>
</html>
```

ウィンドウのサイズ変更と移動

Adobe AIR 1.0 およびそれ以降

ウィンドウがシステムクロムを使用している場合は、クロムがウィンドウのサイズ変更とデスクトップ上での移動に対するドラッグを制御します。システムクロムを使用しない場合は、独自のコントロールを追加して、ユーザーがウィンドウのサイズ変更および移動を行うことができるようにする必要があります。

注意：ウィンドウのサイズ変更または移動を行うには、最初に、NativeWindow インスタンスに対する参照を取得する必要があります。ウィンドウの参照を取得する方法について詳しくは、97 ページの「NativeWindow インスタンスの取得」を参照してください。

ウィンドウのサイズ変更

ユーザーがインタラクティブにウィンドウのサイズを変更できるようにするには、NativeWindow の startResize() メソッドを使用します。mouseDown イベントからこのメソッドを呼び出すと、マウスの動作に応じてサイズ変更操作が行われ、オペレーティングシステムが mouseUp イベントを受信すると完了します。startResize() を呼び出すときは、ウィンドウのサイズ変更の基準になる辺または角を指定する引数を渡します。

ウィンドウサイズをプログラムによって設定するには、ウィンドウの width、height または bounds の各プロパティを目的のサイズに設定します。境界を設定すると、ウィンドウのサイズと位置をすべて同時に変更できます。ただし、変更が行われる順序は保証されません。一部の Linux ウィンドウマネージャーでは、デスクトップ画面の境界の外側にウィンドウを拡張することはできません。このような場合、プロパティの設定順序により、通常は変更の純粋な影響により正式なウィンドウが作成される場合でも、最終的なウィンドウのサイズが制限される可能性があります。例えば、画面の下部付近でウィンドウの高さと y 位置の両方を変更する場合、y 位置の変更の前に高さの変更が適用されると、高さが完全に変更されない可能性があります。

注意：Linux では、ウィンドウプロパティは非同期で変更されます。プログラムのある行でウィンドウのサイズを変更し、次の行でサイズを読み取る場合、読み取られた値には古い値が反映されます。すべてのプラットフォームで、ウィンドウのサイズが変更されると、NativeWindow オブジェクトから resize イベントが送出されます。ウィンドウの新しいサイズまたは状態に基づいて何らかのアクション（ウィンドウ内のコントロールのレイアウトなど）を実行する必要がある場合、常に resize イベントハンドラーで行ってください。104 ページの「ウィンドウイベントのリッスン」を参照してください。

ウィンドウの移動

サイズを変更しないでウィンドウを移動するには、NativeWindow startMove() メソッドを使用します。startResize() メソッドと同様に、mouseDown イベントから startMove() メソッドを呼び出すと、マウスの操作に応じて移動処理が行われ、オペレーティングシステムが mouseUp イベントを受信すると完了します。

startResize() メソッドおよび startMove() メソッドについて詳しくは、『HTML 開発者用 Adobe AIR API リファレンスガイド』を参照してください。

ウィンドウをプログラムによって移動するには、ウィンドウの x、y または bounds の各プロパティを目的の位置に設定します。境界を設定すると、ウィンドウのサイズと位置を両方同時に変更できます。

注意：Linux では、ウィンドウプロパティは非同期で変更されます。プログラムのある行でウィンドウを移動し、次の行で位置を読み取る場合、読み取られた値には古い値が反映されます。すべてのプラットフォームで、位置が変更されると、NativeWindow オブジェクトから move イベントが送出されます。ウィンドウの新しい位置に基づいて何らかのアクションを実行する必要がある場合、常に move イベントハンドラーで行ってください。104 ページの「ウィンドウイベントのリッスン」を参照してください。

例：ウィンドウのサイズ変更と移動

Adobe AIR 1.0 およびそれ以降

次の例では、ウィンドウでのサイズ変更および移動操作を開始する方法を示します。

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<script src="AIRAliases.js"/>
<script type="text/javascript">
     function onResize(type){
        nativeWindow.startResize(type);
     }

     function onNativeMove(){
        nativeWindow.startMove();
     }
</script>
<style type="text/css" media="screen">

.drag {
    width:200px;
    height:200px;
    margin:0px auto;
    padding:15px;
    border:1px dashed #333;
    background-color:#eee;
    }

.resize {
    background-color:#FF0000;
    padding:10px;
    }
.left {
        float:left;
        }
.right {
    float:right;
}

</style>
<title>Move and Resize the Window</title>
</head>

<body>
<div class="resize left" onmousedown="onResize(air.NativeWindowResize.TOP_LEFT)">Drag to resize</div>
<div class="resize right" onmousedown="onResize(air.NativeWindowResize.TOP_RIGHT)">Drag to resize</div>
<div class="drag" onmousedown="onNativeMove()">Drag to move</div>
<div class="resize left" onmousedown="onResize(air.NativeWindowResize.BOTTOM_LEFT)">Drag to resize</div>
<div class="resize right" onmousedown="onResize(air.NativeWindowResize.BOTTOM_RIGHT)">Drag to resize</div>
</body>
</html>
```

ウィンドウイベントのリッスン

Adobe AIR 1.0 およびそれ以降

ウィンドウによって送出されるイベントをリッスンするには、リスナーをウィンドウのインスタンスに登録します。例えば、closing イベントをリッスンするには、次のようにリスナーをウィンドウに登録します。

```
window.nativeWindow.addEventListener(air.Event.CLOSING, onClosingEvent);
```

イベントが送出されると、target プロパティはイベントを送信したウィンドウを示します。

ほとんどのウィンドウイベントには、2 つの関連メッセージがあります。第 1 のメッセージはウィンドウの変更が行われようとしていること（およびキャンセルできること）を通知し、第 2 のメッセージは変更が行われたことを通知します。例えば、ユーザーがウィンドウの閉じるボタンをクリックすると、closing イベントのメッセージが送出されます。リスナーがイベントをキャンセルしないと、ウィンドウが閉じ、close イベントがリスナーに送出されます。

通常、closing などの警告イベントは、システムクロムを使用してイベントがトリガーされた場合にのみ送出されます。例えばウィンドウの close() メソッドなどを呼び出しても、closing イベントは自動的には送出されず、close イベントのみが送出されます。ただし、closing イベントオブジェクトを作成し、ウィンドウの dispatchEvent() メソッドを使用して送出することはできます。

Event オブジェクトを送出するウィンドウイベントは次のとおりです。

イベント	**説明**
activate	ウィンドウがフォーカスを受け取ると送出されます。
deactivate	ウィンドウがフォーカスを失うと送出されます。
closing	ウィンドウが閉じられようとするときに送出されます。このイベントが自動的に発生するのは、システムクロムの閉じるボタンが押されたとき、または Mac OS X で Quit コマンドが呼び出されたときのみです。
close	ウィンドウが閉じると送出されます。

NativeWindowBoundsEvent オブジェクトを送出するウィンドウイベントは次のとおりです。

イベント	**説明**
moving	移動、サイズ変更またはウィンドウ表示状態の変更の結果として、ウィンドウの左上隅の位置が変更される直前に送出されます。
move	左上隅の位置が変更された後で送出されます。
resizing	サイズ変更または表示状態変更の結果としてウィンドウの幅または高さが変更される直前に送出されます。
resize	ウィンドウのサイズが変更された後で送出されます。

NativeWindowBoundsEvent イベントの場合は、beforeBounds および afterBounds プロパティを使用して、変更の直前と直後のウィンドウ境界を取得できます。

NativeWindowDisplayStateEvent オブジェクトを送出するウィンドウイベントは次のとおりです。

イベント	**説明**
displayStateChanging	ウィンドウの表示状態が変更される直前に送出されます。
displayStateChange	ウィンドウの表示状態が変更された後で送出されます。

NativeWindowDisplayStateEvent イベントの場合は、beforeDisplayState および afterDisplayState プロパティを使用して、変更の直前と直後のウィンドウの表示状態を取得できます。

一部の Linux ウィンドウマネージャーでは、最大サイズが設定されているウィンドウを最大化しても、表示状態変更イベントは送出されません（ウィンドウは最大表示状態に設定されますが、サイズは変更されません）。

フルスクリーンウィンドウの表示

Adobe AIR 1.0 およびそれ以降

Stage の displayState プロパティを StageDisplayState.FULL_SCREEN_INTERACTIVE に設定すると、ウィンドウはフルスクリーンモードになり、このモードでのキーボード入力が許可されます（ブラウザーで実行する SWF コンテンツでは、キーボード入力は許可されません）。ユーザーが Esc キーを押すと、フルスクリーンモードは終了します。

注意：一部の Linux ウィンドウマネージャーでは、ウィンドウに最大サイズが設定されている場合、ウィンドウのサイズはスクリーンいっぱいになりません（ただし、ウィンドウシステムクロムは削除されます）。

例えば、次の Flex コードは、単純なフルスクリーン端末を設定する簡単な AIR アプリケーションを定義しています。

```
 <?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    applicationComplete="init()" backgroundColor="0x003030" focusRect="false">
    <mx:Script>
        <![CDATA[
            private function init():void
            {
                stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
                focusManager.setFocus(terminal);
                terminal.text = "Welcome to the dumb terminal app. Press the ESC key to exit..\n";
                terminal.selectionBeginIndex = terminal.text.length;
                terminal.selectionEndIndex = terminal.text.length;
            }
        ]]>
    </mx:Script>
    <mx:TextArea
        id="terminal"
        height="100%" width="100%"
        scroll="false"
        backgroundColor="0x003030"
        color="0xCCFF00"
        fontFamily="Lucida Console"
        fontSize="44"/>
</mx:WindowedApplication>
```

次に示す Flash 用の ActionScript の例は、簡単なフルスクリーンテキスト端末をシミュレーションします。

```
    import flash.display.Sprite;
    import flash.display.StageDisplayState;
    import flash.text.TextField;
    import flash.text.TextFormat;

    public class FullScreenTerminalExample extends Sprite
    {
        public function FullScreenTerminalExample():void
        {
            var terminal:TextField = new TextField();
            terminal.multiline = true;
            terminal.wordWrap = true;
            terminal.selectable = true;
            terminal.background = true;
            terminal.backgroundColor = 0x00333333;

            this.stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;

            addChild(terminal);
            terminal.width = 550;
            terminal.height = 400;

            terminal.text = "Welcome to the dumb terminal application. Press the ESC key to exit.\n_";

            var tf:TextFormat = new TextFormat();
            tf.font = "Courier New";
            tf.color = 0x00CCFF00;
            tf.size = 12;
            terminal.setTextFormat(tf);

            terminal.setSelection(terminal.text.length - 1, terminal.text.length);
        }
    }
```

次に示す HTML ページは、フルスクリーンテキスト端末をシミュレーションします。

```
<html>
<head>
<title>Fullscreen Mode</title>
<script language="JavaScript" type="text/javascript">
function setDisplayState() {
    window.nativeWindow.stage.displayState =
        runtime.flash.display.StageDisplayState.FULL_SCREEN_INTERACTIVE;
}
</script>
<style type="text/css">
body, .mono {
    font-family: Courier New, Courier, monospace;
    font-size: x-large;
    color:#CCFF00;
    background-color:#003030;
}
</style>
</head>
<body onload="setDisplayState();">
    <p class="mono">Welcome to the dumb terminal app. Press the ESC key to exit...</p>
    <textarea name="dumb" class="mono" cols="100" rows="40">%</textarea>
</body>
</html>
```

第 8 章：AIR の表示スクリーン

Adobe AIR 1.0 およびそれ以降

Adobe® AIR® Screen クラスを使用すると、コンピューターまたはデバイスに接続された表示スクリーンに関する情報にアクセスできます。

関連項目

flash.display.Screen

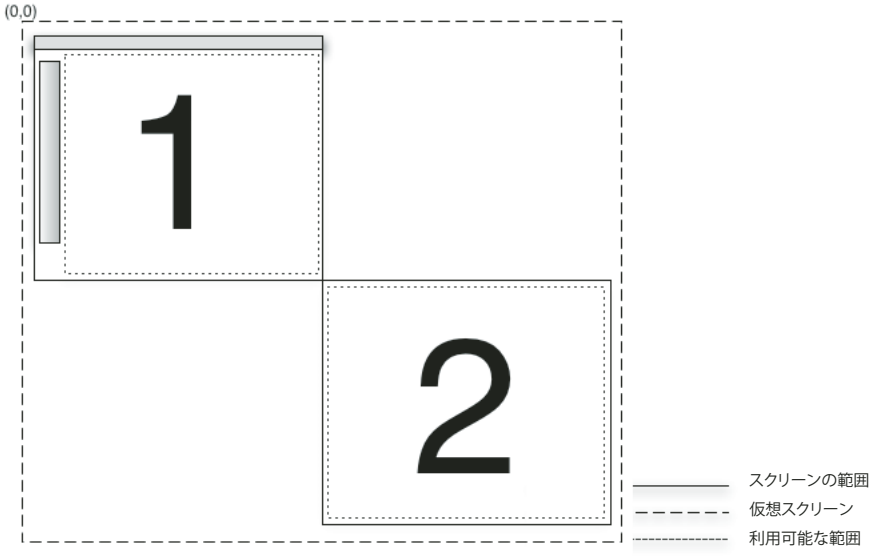
AIR の表示スクリーンの基礎

Adobe AIR 1.0 およびそれ以降

スクリーン API には Screen という 1 つのクラスが含まれており、このクラスには、システムのスクリーンに関する情報を取得するための静的メンバーと、特定のスクリーンを表すためのインスタンスメンバーが用意されています。

コンピューターシステムには複数のモニターまたはディスプレイを接続でき、仮想空間に配置された複数のデスクトップスクリーンにそれらを対応させることができます。AIR の Screen クラスは、スクリーン、スクリーンの相対的な配置、スクリーンの利用可能な領域に関する情報を提供します。複数のモニターを同じスクリーンにマップした場合、1 つだけのスクリーンが存在することになります。スクリーンのサイズがモニターの表示領域より大きい場合、スクリーンのどの部分が現在表示されているかを特定する方法はありません。

1 つのスクリーンは、1 つの独立したデスクトップ表示領域を表します。仮想デスクトップ内では、スクリーンは矩形で表されます。プライマリディスプレイとして指定されているスクリーンの左上隅が、仮想デスクトップの座標系の原点です。スクリーンを表すために使用する値は、すべてピクセル単位で指定します。



このスクリーン配置では、2 つのスクリーンが仮想デスクトップ上に存在します。メインスクリーン（#1）の左上隅の座標は、常に（0,0）です。スクリーン配置を変更してスクリーン #2 をメインスクリーンとして指定すると、スクリーン #1 の座標は負になります。スクリーンの利用可能な範囲の報告時、メニューバー、タスクバーおよびドックは除外されます。

スクリーン API のクラス、メソッド、プロパティおよびイベントについて詳しくは、『HTML 開発者用 Adobe AIR API リファレンスガイド』を参照してください。

スクリーンの列挙

Adobe AIR 1.0 およびそれ以降

スクリーンの次のメソッドとプロパティを使用して、仮想デスクトップのスクリーンを列挙できます。

メソッドまたはプロパティ	説明
Screen.screens	使用可能なスクリーンを表す Screen オブジェクトの配列を提供します。配列の順序は重要ではありません。
Screen.mainScreen	メインスクリーンの Screen オブジェクトを提供します。Mac OS X では、メインスクリーンはメニューバーを表示するスクリーンです。Windows では、メインスクリーンはシステム指定のプライマリスクリーンです。
Screen.getScreensForRectangle()	指定された矩形に重なるスクリーンを表す Screen オブジェクトの配列を提供します。このメソッドに渡す矩形は、仮想デスクトップ上のピクセル座標で指定します。矩形に重なるスクリーンがない場合、配列は空になります。このメソッドを使用して、ウィンドウが表示されるスクリーンを調べることができます。

Screen クラスのメソッドとプロパティから返された値は保存しないでください。使用可能なスクリーンとその配置は、ユーザーまたはオペレーティングシステムによって常に変更される可能性があります。

次の例では、スクリーン API を使用して、矢印キーの押下に応答して複数のスクリーン間でウィンドウを移動します。ウィンドウを次のスクリーンに移動するため、この例では、screens 配列を取得し、その配列を縦方向または横方向に並べ替えます（方向は押された矢印キーによって決まります）。次に、並べ替えた配列に従って、各スクリーンを現在のスクリーンの座標と比較します。ウィンドウの現在のスクリーンを特定するため、この例では、Screen.getScreensForRectangle() を呼び出してウィンドウの境界を渡します。

```
<html>
    <head>
        <script src="AIRAliases.js" type="text/javascript"></script>
        <script type="text/javascript">
        function onKey(event){
            if(air.Screen.screens.length > 1){
                switch(event.keyCode){
                    case air.Keyboard.LEFT :
                        moveLeft();
                        break;
                    case air.Keyboard.RIGHT :
                        moveRight();
                        break;
                    case air.Keyboard.UP :
                        moveUp();
                        break;
                    case air.Keyboard.DOWN :
                        moveDown();
                        break;
                }

            }
        }

        function moveLeft(){
            var currentScreen = getCurrentScreen();
            var left = air.Screen.screens;
            left.sort(sortHorizontal);
```

```
    for(var i = 0; i < left.length - 1; i++){
        if(left[i].bounds.left < window.nativeWindow.bounds.left){
            window.nativeWindow.x += left[i].bounds.left - currentScreen.bounds.left;
            window.nativeWindow.y += left[i].bounds.top - currentScreen.bounds.top;
        }
    }
}

function moveRight(){
    var currentScreen = getCurrentScreen();
    var left = air.Screen.screens;
    left.sort(sortHorizontal);
    for(var i = left.length - 1; i > 0; i--){
        if(left[i].bounds.left > window.nativeWindow.bounds.left){
            window.nativeWindow.x += left[i].bounds.left - currentScreen.bounds.left;
            window.nativeWindow.y += left[i].bounds.top - currentScreen.bounds.top;
        }
    }
}

function moveUp(){
    var currentScreen = getCurrentScreen();
    var top = air.Screen.screens;
    top.sort(sortVertical);
    for(var i = 0; i < top.length - 1; i++){
        if(top[i].bounds.top < window.nativeWindow.bounds.top){
            window.nativeWindow.x += top[i].bounds.left - currentScreen.bounds.left;
            window.nativeWindow.y += top[i].bounds.top - currentScreen.bounds.top;
            break;
        }
    }
}

function moveDown(){
    var currentScreen = getCurrentScreen();

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

function sortHorizontal(a,b){
    if (a.bounds.left > b.bounds.left){
        return 1;
    } else if (a.bounds.left < b.bounds.left){
        return -1;
    } else {return 0;}
}

function sortVertical(a,b){
    if (a.bounds.top > b.bounds.top){
        return 1;
    } else if (a.bounds.top < b.bounds.top){
        return -1;
```

```
            } else {return 0;}
        }

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

        function init(){
            window.nativeWindow.stage.addEventListener("keyDown",onKey);
        }
        </script>
        <title>Screen Hopper</title>
    </head>
    <body  onload="init()">
        <p>Use the arrow keys to move the window between monitors.</p>
    </body>
</html>
```

第 9 章：メニューの操作

Flash Player 9 以降、Adobe AIR 1.0 以降

ネイティブメニュー API のクラスを使用して、Adobe® AIR® でアプリケーション、ウィンドウ、コンテキストおよびポップアップの各メニューを定義できます。

メニューの基本

Flash Player 9 以降、Adobe AIR 1.0 以降

AIR アプリケーションでのネイティブメニューの作成に関する簡単な説明およびコード例については、Adobe Developer Connection で次のクイックスタートの記事を参照してください。

- Adding native menus to an AIR application

ネイティブメニュークラスを使用すると、アプリケーションを実行しているオペレーティングシステムのネイティブメニュー機能にアクセスすることができます。NativeMenu オブジェクトは、アプリケーションメニュー（Mac OS X で使用可能）、ウィンドウメニュー（Windows および Linux で使用可能）、コンテキストメニューおよびポップアップメニューに使用できます。

AIR の外部で、コンテキストメニュークラスを使用することにより、アプリケーションでユーザーがオブジェクトを右クリックまたは Command キーを押しながらクリックしたときに Flash Player で自動的に表示するコンテキストメニューを変更できます（自動コンテキストメニューは AIR アプリケーションでは表示されません）。

メニュークラス

Flash Player 9 以降、Adobe AIR 1.0 以降

メニュークラスには、次のクラスが含まれます。

パッケージ	クラス
flash.display	• NativeMenu • NativeMenuItem
flash.events	• Event

メニューのタイプ

Flash Player 9 以降、Adobe AIR 1.0 以降

AIR では、次のタイプのメニューをサポートしています。

コンテキストメニュー コンテキストメニューは、SWF コンテンツ内のインタラクティブオブジェクトまたは HTML コンテンツ内のドキュメントエレメントを右クリックまたは Command キーを押しながらクリックすると開くメニューです。

Flash Player ランタイムでは、コンテキストメニューが自動的に表示されます。ContextMenu および ContextMenuItem クラスを使用して、メニューに独自のコマンドを追加できます。また、ビルトインコマンドの一部（全部ではなく）を削除できます。

AIR ランタイムでは、NativeMenu クラスまたは ContextMenu クラスを使用して、コンテキストメニューを作成できます。AIR の HTML コンテンツでは、Webkit HTML や JavaScript の API を使用して、コンテキストメニューを HTML エレメントに追加できます。

アプリケーション メニュー（AIR のみ）アプリケーションメニューは、アプリケーション全体に適用されるグローバルなメニューです。アプリケーションメニューは Mac OS X でサポートされ、Windows および Linux ではサポートされません。Mac OS X では、オペレーティングシステムで自動的にアプリケーションメニューが作成されます。AIR のメニュー API を使用すると、標準のメニューにアイテムやサブメニューを追加したり、既存のメニューコマンドを処理するリスナーを追加したり、既存のアイテムを削除したりすることができます。

ウィンドウメニュー（AIR のみ）ウィンドウメニューは、1 つのウィンドウに関連付けられるメニューで、タイトルバーの下に表示されます。NativeMenu オブジェクトを作成して NativeWindow オブジェクトの menu プロパティに割り当てることで、ウィンドウにメニューを追加できます。ウィンドウメニューは Windows および Linux オペレーティングシステムでサポートされ、Mac OS X ではサポートされません。ネイティブウィンドウメニューは、システムクロムを持つウィンドウでのみ使用できます。

ドックおよびシステムトレイアイコンメニュー（AIR のみ）これらのアイコンメニューは、コンテキストメニューに似ており、Mac OS X の場合はドック内、Windows および Linux の場合はタスクバーの通知領域内に表示されるアプリケーションアイコンに割り当てられます。ドックおよびシステムトレイアイコンメニューでは、NativeMenu クラスが使用されます。Mac OS X では、メニューのアイテムは、オペレーティングシステムの標準のアイテムの上に追加されます。Windows または Linux では、標準のメニューはありません。

ポップアップメニュー（AIR のみ）AIR のポップアップメニューはコンテキストメニューに似ていますが、特定のアプリケーションオブジェクトまたはコンポーネントに必ずしも関連付けられません。ポップアップメニューは、任意の NativeMenu オブジェクトの display() メソッドを呼び出すことで、ウィンドウ内の任意の位置に表示できます。

カスタムメニュー ネイティブメニューは、完全にオペレーティングシステムによって描画されるため、Flash や HTML のレンダリングモデル内に含めることはできません。ネイティブメニューを使用する代わりに、MXML、ActionScript または JavaScript（AIR の場合）を使用して独自のカスタム非ネイティブメニューをいつでも作成できます。そのようなメニューは、アプリケーションコンテンツ内で完全にレンダリングされる必要があります。

デフォルトメニュー（AIR のみ）
オペレーティングシステムまたは AIR ビルトインクラスで、次のデフォルトのメニューが提供されます。

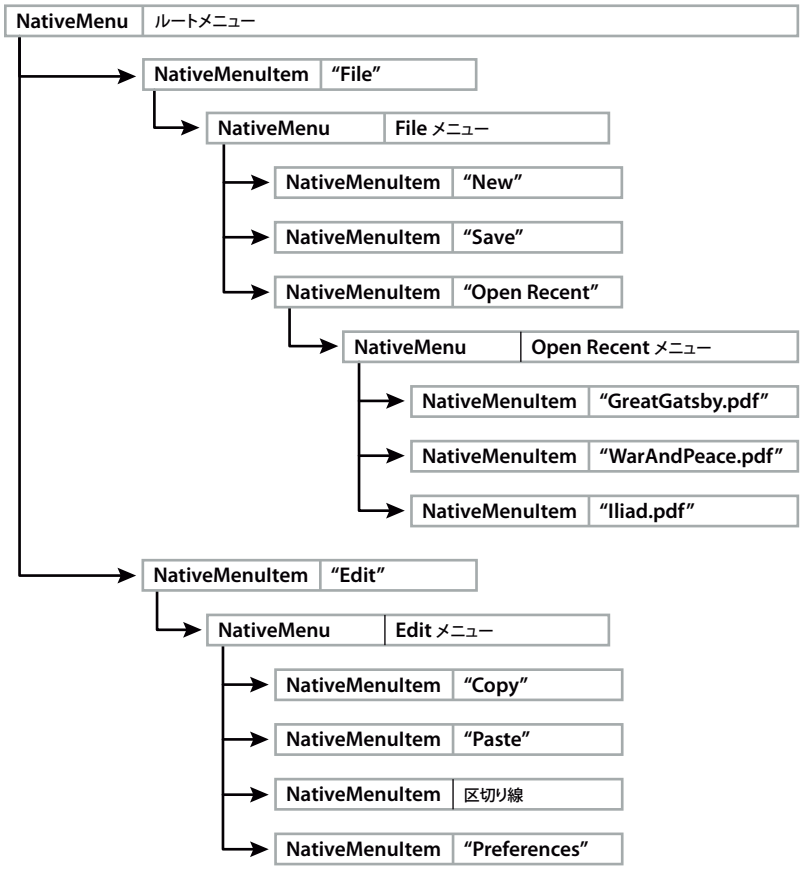
- Mac OS X のアプリケーションメニュー
- Mac OS X のドックアイコンメニュー
- HTML コンテンツの選択されたテキストおよびイメージに対応するコンテキストメニュー
- TextField オブジェクト（または TextField を拡張するオブジェクト）の選択されたテキストに対応するコンテキストメニュー

ネイティブメニューの構造（AIR）

Adobe AIR 1.0 およびそれ以降

ネイティブメニューは、性質として階層的な構造をしています。NativeMenu オブジェクトには、子の NativeMenuItem オブジェクトが含まれます。さらに、サブメニューを表す NativeMenuItem オブジェクトに NativeMenu オブジェクトを含めることができます。構造内の最上位レベル、つまりルートレベルのメニューオブジェクトは、アプリケーションメニューおよびウィンドウメニューのメニューバーを表します（コンテキスト、アイコンおよびポップアップの各メニューにはメニューバーはありません）。

次の図に、典型的なメニューの構造を示します。ルートメニューはメニューバーを表しており、**File** サブメニューおよび **Edit** サブメニューを参照する 2 つのメニューアイテムが含まれています。この構造の File サブメニューには、2 つのコマンドアイテムと、**Open Recent Menu** サブメニュー（このサブメニュー自体にも 3 つのアイテムが含まれています）を参照するアイテムが含まれています。Edit サブメニューには、3 つのコマンドと 1 つのセパレーターが含まれています。



サブメニューを定義するには、NativeMenu オブジェクトと NativeMenuItem オブジェクトの両方が必要です。NativeMenuItem オブジェクトでは親メニューに表示されるラベルが定義され、ユーザーはこのオブジェクトを使用してサブメニューを開くことができます。NativeMenu オブジェクトは、サブメニューのアイテムのコンテナとして機能します。NativeMenuItem オブジェクトでは、NativeMenuItem の submenu プロパティを通して NativeMenu オブジェクトを参照します。

このメニューを作成するコードの例については、122 ページの「ネイティブメニューの例：ウィンドウメニューとアプリケーションメニュー（AIR）」を参照してください。

メニューイベント

Adobe AIR 1.0 およびそれ以降

NativeMenu オブジェクトと NativeMenuItem オブジェクトは、どちらも preparing イベント、displaying イベント、select イベントを送出します。

preparing：オブジェクトがユーザー操作を開始する直前に、登録されているすべてのリスナーに対して、メニューおよびそのメニューアイテムから preparin イベントが送出されます。操作には、メニューを開く操作やキーボードショートカットによるアイテムの選択などが含まれます。

注意：preparing イベントは、Adobe AIR 2.6 以降のみで利用できます。

displaying： メニューが表示される直前に、メニューおよびそのメニューアイテムから、登録されているすべてのリスナーに displaying イベントが送出されます。

preparing イベントと displaying イベントにより、メニューの内容やアイテムの外観を、ユーザーに表示される前に更新できます。例えば、最近使用したファイルを開くメニューの displaying イベントのリスナーでは、最近表示したドキュメントの最新の一覧を反映するようにメニューアイテムを変更することができます。

preparing イベントをトリガーしたキーボードショートカットに対応するメニューアイテムを削除すると、メニュー操作は事実上キャンセルされ、select イベントは送出されません。

イベントの target プロパティと currentTarget プロパティはどちらもリスナーの登録先のオブジェクト（メニューそのものかメニューアイテムの 1 つ）です。

preparing イベントは displaying イベントの前に送出されます。通常はこれらのイベントの両方をリッスンすることはなく、一方のみをリッスンします。

select： コマンドアイテムがユーザーによって選択されると、そのアイテムから、登録されているすべてのリスナーに select イベントが送出されます。サブメニューアイテムとセパレーターアイテムは選択できないため、これらから select イベントが送出されることはありません。

select イベントは、メニューアイテムからそれを含むメニューへと、ルートメニューに達するまでバブルします。select イベントは、アイテムで直接リッスンすることも、メニュー構造の上位でリッスンすることもできます。select イベントをメニューでリッスンする場合は、そのイベントの target プロパティを使用して、選択されたアイテムを識別できます。メニュー階層の上方にイベントがバブルしていくのに合わせて、イベントオブジェクトの currentTarget プロパティは、現在のメニューオブジェクトを識別します。

注意：ContextMenu オブジェクトと ContextMenuItem オブジェクトでは、menuItemSelect イベントと menuSelect イベントに加え、select イベント、preparing イベント、displaying イベントも送出します。

ネイティブメニューコマンドと等価なキー（AIR）

Adobe AIR 1.0 およびそれ以降

メニューコマンドにキーボードショートカット（アクセラレーター）を割り当てることができます。キーまたはキーの組み合わせが押されると、メニューアイテムから、登録されているすべてのリスナーに select イベントが送出されます。アイテムが格納されるメニューは、コマンドが呼び出されるアプリケーションまたはアクティブなウィンドウのメニューの一部である必要があります。

キーボードショートカットは 2 つの部分から構成されます。主キーを表すストリングと、一緒に押される必要がある修飾キーの配列です。主キーを割り当てるには、メニューアイテムの keyEquivalent プロパティに、そのキーの 1 文字のストリングを設定します。大文字を使用した場合、修飾キーの配列に Shift キーが自動的に追加されます。

Mac OS X では、デフォルトの修飾キーは Command キー（Keyboard.COMMAND）です。Windows および Linux では、Ctrl キー（Keyboard.CONTROL）です。これらのデフォルトのキーは、修飾キーの配列に自動的に追加されます。別の修飾キーを割り当てるには、そのキーコードを含む新しい配列を keyEquivalentModifiers プロパティに割り当てます。デフォルトの配列は上書きされます。デフォルトの修飾キーを使用するか独自の修飾キーの配列を割り当てるかに関係なく、keyEquivalent プロパティに割り当てたストリングが大文字の場合は Shift キーが追加されます。修飾キーに使用するキーコードの定数は、Keyboard クラスで定義されています。

割り当てられたキーボードショートカットのストリングは、自動的にメニューアイテム名の横に表示されます。形式は、ユーザーのオペレーティングシステムおよびシステムの環境設定によって異なります。

注意：Windows オペレーティングシステムでキー修飾の配列に Keyboard.COMMAND 値を割り当てた場合、メニューにキーボードショートカットは表示されません。ただし、そのメニューコマンドをアクティブにするには Ctrl キーを使用する必要があります。

次の例では、メニューアイテムのキーボードショートカットとして Ctrl+Shift+G を割り当てます。

```
var item = new air.NativeMenuItem("Ungroup");
item.keyEquivalent = "G";
```

次の例では、修飾キーの配列を直接設定することにより、キーボードショートカットとして Ctrl+Shift+G を割り当てます。

```
var item = new air.NativeMenuItem("Ungroup");
item.keyEquivalent = "G";
item.keyEquivalentModifiers = [air.Keyboard.CONTROL];
```

注意：キーボードショートカットでトリガーできるのは、アプリケーションメニューまたはウィンドウメニューのみです。コンテキストメニューまたはポップアップメニューにキーボードショートカットを追加した場合、メニューラベルにはキーボードショートカットが表示されますが、関連付けられたメニューコマンドを呼び出すことはできません。

ニーモニック（AIR）

Adobe AIR 1.0 およびそれ以降

ニーモニックは、オペレーティングシステムのメニューのキーボードインターフェイスの一部です。Linux、Mac OS X および Windows では、いずれもキーボードでメニューを開いてコマンドを選択する操作ができますが、その方法には微妙な違いがあります。

Mac OS X では、ユーザーはメニューまたはコマンドの最初の 1 文字か 2 文字を入力して、Return キーを押します。mnemonicIndex プロパティは無視されます。

Windows では、これを 1 文字だけで実行できます。デフォルトでは、有効な文字はラベルの最初の文字ですが、メニューアイテムにニーモニックを割り当てると、有効な文字は指定した文字になります。メニュー内の 2 つのアイテムの有効な文字が同じ場合（ニーモニックが割り当てられているかどうかに関係なく）、メニューのキーボード操作が少し異なります。1 文字入力してメニューまたはコマンドを選択するのではなく、ユーザーは必要な回数だけその文字を入力して対象のアイテムを強調表示してから、Enter キーを押して選択を完了する必要があります。動作の一貫性を維持するために、ウィンドウメニューでは、メニュー内のアイテムそれぞれに異なるニーモニックを割り当てることをお勧めします。

Linux では、既定のニーモニックは設定されません。メニューアイテムの mnemonicIndex プロパティ値を設定することでニーモニックを指定する必要があります。

ニーモニック文字は、ラベルストリングのインデックスとして指定します。ラベルの最初の文字のインデックスは 0 です。したがって、「Format」というラベルのメニューアイテムのニーモニックとして「r」を使用するには、mnemonicIndex プロパティを 2 に設定します。

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

メニューアイテムの状態

Adobe AIR 1.0 およびそれ以降

メニューアイテムには、状態を表すプロパティとして、checked および enabled という 2 つのプロパティがあります。

checked true に設定すると、アイテムラベルの横にチェックマークが表示されます。

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

enabled 値を true と false で切り替えて、コマンドを有効にするかどうかを制御します。無効にしたアイテムは「グレー表示」され、そのアイテムからは select イベントは送出されません。

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

メニューアイテムへのオブジェクトの関連付け

Adobe AIR 1.0 およびそれ以降

NativeMenuItem クラスの data プロパティを使用して、各アイテムの任意のオブジェクトを参照できます。例えば、最近使用したファイルを開くメニューでは、それぞれのメニューアイテムに各ドキュメントの File オブジェクトを割り当てることができます。

```
var file = air.File.applicationStorageDirectory.resolvePath("GreatGatsby.pdf")
var menuItem = docMenu.addItem(new air.NativeMenuItem(file.name));
menuItem.data = file;
```

ネイティブメニューの作成（AIR）

Adobe AIR 1.0 およびそれ以降

このトピックでは、AIR でサポートされる様々なタイプのネイティブメニューを作成する方法を示します。

ルートメニューオブジェクトの作成

Adobe AIR 1.0 およびそれ以降

メニューのルートとして機能する NativeMenu オブジェクトを作成するには、NativeMenu コンストラクターを使用します。

```
var root = new air.NativeMenu();
```

アプリケーションメニューおよびウィンドウメニューでは、ルートメニューはメニューバーを表し、サブメニューを開くアイテムのみが格納されます。コンテキストメニューおよびポップアップメニューにはメニューバーがないため、ルートメニューにコマンドやセパレーター線、およびサブメニューを含めることはできません。

メニューを作成したら、メニューアイテムを追加できます。アイテムは、追加した順序で表示されます。アイテムを特定のインデックスに表示するには、メニューオブジェクトの addItemAt() メソッドを使用します。

以降の節で示すように、メニューはアプリケーション、ウィンドウまたはアイコンのいずれかのメニューとして割り当てるか、ポップアップメニューとして表示します。

アプリケーションメニューの設定またはウィンドウメニューの設定

コードには、アプリケーションメニュー（Mac OS でサポート）とウィンドウメニュー（その他のオペレーティングシステムでサポート）の両方を含めることが重要です。

```
var root = new air.NativeMenu();
if (air.NativeApplication.supportsMenu)
{
    air.NativeApplication.nativeApplication.menu = root;
}
else if (NativeWindow.supportsMenu)
{
    nativeWindow.menu = root;
}
```

注意：Mac OS では、すべてのアプリケーション用の標準アイテムを含むメニューを定義します。新しい NativeMenu オブジェクトを NativeApplication オブジェクトの menu プロパティに割り当てると、標準のメニューが置き換えられます。置き換えずに標準のメニューを使用することもできます。

Adobe Flex の FlexNativeMenu クラスを使用すると、様々なプラットフォームで機能するメニューを簡単に作成できます。Flex フレームワークを使用している場合は、NativeMenu クラスではなく FlexNativeMenu クラスを使用します。

ドックアイコンメニューの設定またはシステムトレイアイコンメニューの設定

```
air.NativeApplication.nativeApplication.icon.menu = root;
```

注意：Mac OS X では、アプリケーションのドックアイコン用に標準のメニューが定義されています。新しい NativeMenu を DockIcon オブジェクトの menu プロパティに割り当てると、そのメニューに含まれるアイテムが標準のアイテムの上に表示されます。標準のメニューアイテムの削除、アクセスまたは変更を行うことはできません。

ポップアップとしてのメニューの表示

```
root.display(window.nativeWindow.stage, x, y);
```

関連項目

Developing cross-platform AIR applications

サブメニューの作成

Adobe AIR 1.0 およびそれ以降

サブメニューを作成するには、NativeMenuItem オブジェクトを親メニューに追加し、そのアイテムの submenu プロパティにサブメニューを定義する NativeMenu オブジェクトを割り当てます。AIR には、サブメニューアイテムとそれに関連するメニューオブジェクトを作成する方法が 2 つあります。

addSubmenu() メソッドを使用すると、メニューアイテムおよび関連するメニューオブジェクトを 1 つの手順で作成できます。

```
var editMenuItem = root.addSubmenu(new air.NativeMenu(), "Edit");
```

メニューアイテムの作成とその submenu プロパティへのメニューオブジェクトの割り当てを、別々に行うこともできます。

```
var editMenuItem = root.addItem("Edit", false);
editMenuItem.submenu = new air.NativeMenu();
```

メニューコマンドの作成

Adobe AIR 1.0 およびそれ以降

メニューコマンドを作成するには、NativeMenuItem オブジェクトをメニューに追加し、メニューコマンドを実装する関数を参照するイベントリスナーを追加します。

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

select イベントは、この例のようにコマンドアイテム自体でリッスンすることも、親メニューオブジェクトでリッスンすることもできます。

注意：サブメニューおよびセパレーター線を表すメニューアイテムからは select イベントは送出されないため、それらのアイテムをコマンドとして使用することはできません。

メニューセパレーター線の作成

Adobe AIR 1.0 およびそれ以降

セパレーター線を作成するには、NativeMenuItem を作成し、そのコンストラクターで isSeparator パラメーターを true に設定します。次に、セパレーターアイテムをメニュー内の適切な位置に追加します。

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

セパレーターにラベルを指定しても、そのラベルは表示されません。

コンテキストメニューについて：HTML（AIR）

Adobe AIR 1.0 およびそれ以降

HTMLLoader オブジェクトを使用して表示される HTML コンテンツでは、contextmenu イベントを使用してコンテキストメニューを表示できます。デフォルトでは、ユーザーが選択したテキストで（テキストを右クリックするか Command キーを押しながらクリックして）コンテキストメニューを呼び出すと、自動的にコンテキストメニューが表示されます。デフォルトのメニューが開かないようにするには、contextmenu イベントをリッスンし、そのイベントオブジェクトの preventDefault() メソッドを呼び出します。

```
function showContextMenu(event){
    event.preventDefault();
}
```

その後、DHTML の手法を使用するか AIR のネイティブコンテキストメニューを表示して、カスタムコンテキストメニューを表示できます。次の例では、HTML の contextmenu イベントに応答してメニューの display() メソッドを呼び出すことにより、ネイティブコンテキストメニューを表示します。

```
<html>
<head>
<script src="AIRAliases.js" language="JavaScript" type="text/javascript"></script>
<script language="javascript" type="text/javascript">

function showContextMenu(event){
    event.preventDefault();
    contextMenu.display(window.nativeWindow.stage, event.clientX, event.clientY);
}

function createContextMenu(){
    var menu = new air.NativeMenu();
    var command = menu.addItem(new air.NativeMenuItem("Custom command"));
    command.addEventListener(air.Event.SELECT, onCommand);
    return menu;
}

function onCommand(){
    air.trace("Context command invoked.");
}

var contextMenu = createContextMenu();
</script>
</head>
<body>
<p oncontextmenu="showContextMenu(event)" style="-khtml-user-select:auto;">Custom context menu.</p>
</body>
</html>
```

ポップアップネイティブメニューの表示（AIR）

Adobe AIR 1.0 およびそれ以降

NativeMenu オブジェクトは、メニューの display() メソッドを呼び出すことで、いつでもウィンドウ上の任意の位置に表示できます。このメソッドではステージを参照する必要があるため、メニューをポップアップとして表示できるのは、アプリケーションサンドボックス内のコンテンツのみになります。

次のメソッドでは、マウスクリックに応答して、popupMenu という NativeMenu オブジェクトで定義されたメニューを表示します。

```
function onMouseClick(event) {
    popupMenu.display(window.nativeWindow.stage, event.clientX, event.clientY);
}
```

注意：メニューを表示するのに、必ずしもイベントに直接応答する必要はありません。display() 関数は、どのメソッドでも呼び出せます。

メニューイベントの処理

Flash Player 9 以降、Adobe AIR 1.0 以降

ユーザーがメニューを選択したり、メニューアイテムを選択したりすると、メニューからイベントが送出されます。

メニュークラスのイベントの概要

Flash Player 9 以降、Adobe AIR 1.0 以降

メニューイベントを処理するには、メニューまたは個々のアイテムにイベントリスナーを追加します。

オブジェクト	送出されるイベント
NativeMenu（AIR）	Event.PREPARING（Adobe AIR 2.6 以降） Event.DISPLAYING Event.SELECT（子のアイテムおよびサブメニューから伝達される）
NativeMenuItem（AIR）	Event.PREPARING（Adobe AIR 2.6 以降） Event.SELECT Event.DISPLAYING（親メニューから伝達される）

Select メニューイベント

Adobe AIR 1.0 およびそれ以降

メニューアイテムがクリックされたときにそれに対処するには、select イベントのイベントリスナーを NativeMenuItem オブジェクトに追加します。

```
var menuCommandX = new NativeMenuItem("Command X");
menuCommand.addEventListener(air.Event.SELECT, doCommandX)
```

select イベントは、上位の格納元メニューにバブルするため、親メニューでリッスンすることもできます。メニューレベルでリッスンする場合は、イベントオブジェクトの target プロパティを使用して、選択されたメニューコマンドを判断できます。次の例では、選択されたコマンドのラベルをトレースします。

```
var colorMenuItem = new air.NativeMenuItem("Choose a color");
var colorMenu = new air.NativeMenu();
colorMenuItem.submenu = colorMenu;

var red = new air.NativeMenuItem("Red");
var green = new air.NativeMenuItem("Green");
var blue = new air.NativeMenuItem("Blue");
colorMenu.addItem(red);
colorMenu.addItem(green);
colorMenu.addItem(blue);

if(air.NativeApplication.supportsMenu){
    air.NativeApplication.nativeApplication.menu.addItem(colorMenuItem);
    air.NativeApplication.nativeApplication.menu.addEventListener(air.Event.SELECT,
                                                                  colorChoice);
} else if (air.NativeWindow.supportsMenu){
    var windowMenu = new air.NativeMenu();
    window.nativeWindow.menu = windowMenu;
    windowMenu.addItem(colorMenuItem);
    windowMenu.addEventListener(air.Event.SELECT, colorChoice);
}

function colorChoice(event) {
    var menuItem = event.target;
    air.trace(menuItem.label + " has been selected");
}
```

ContextMenuItem クラスを使用する場合、select イベントまたは menuItemSelect イベントのいずれかをリッスンできます。menuItemSelect イベントでは、コンテキストメニューを所有するオブジェクトに関する追加の情報が提供されます。ただし、このイベントは、上位の格納元メニューにはバブルしません。

Displaying メニューイベント

Adobe AIR 1.0 およびそれ以降

メニューが開かれたときにそれに対処するには、displaying イベントのリスナーを追加します。このイベントは、メニューが表示される前に送出されます。displaying イベントは、メニューを更新する場合に使用できます。例えば、アイテムを追加または削除したり、個々のアイテムについて有効化やチェックの状態を更新できます。また、ContextMenu オブジェクトから menuSelect イベントをリッスンすることもできます。

AIR 2.6 以降では、preparing イベントを使用して、メニューの表示またはキーボードショートカットによるアイテムの選択に応答して、メニューを更新できます。

ネイティブメニューの例：ウィンドウメニューとアプリケーションメニュー（AIR）

Adobe AIR 1.0 およびそれ以降

次の例では、113 ページの「ネイティブメニューの構造（AIR）」で説明したメニューを作成します。

メニューは、Windows（ウィンドウメニューのみがサポートされます）および Mac OS X（アプリケーションメニューのみがサポートされます）の両方で機能するように設計されています。それらを区別するために、MenuExample クラスコンストラクターで、NativeWindow クラスおよび NativeApplication クラスの静的プロパティ supportsMenu を確認しています。NativeWindow.supportsMenu が true の場合は、ウィンドウの NativeMenu オブジェクトを作成してから、File サブメニューと Edit サブメニューを作成して追加します。NativeApplication.supportsMenu が true の場合は、File メニューと Edit メニューを作成して、それらを Mac OS X オペレーティングシステムで提供される既存のメニューに追加します。

次の例では、メニューイベント処理についても示しています。select イベントはアイテムレベルで処理され、さらにメニューレベルでも処理されます。選択されたアイテムを格納するメニューからルートメニューまでのチェーンにある各メニューで、select イベントに応答しています。displaying イベントは、最近使用したファイルを開くメニューで使用されています。メニューが開かれる直前に、メニュー内のアイテムが最新の Documents 配列で更新されます（この例では実際には変更されません）。この例では示していませんが、displaying イベントは個々のアイテムでリッスンすることもできます。

```
<html>
<head>
<script src="AIRAliases.js" type="text/javascript"></script>
<script type="text/javascript">
var application = air.NativeApplication.nativeApplication;
var recentDocuments =
    new Array(new air.File("app-storage:/GreatGatsby.pdf"),
              new air.File("app-storage:/WarAndPeace.pdf"),
              new air.File("app-storage:/Iliad.pdf"));

function MenuExample(){
    var fileMenu;
    var editMenu;

    if (air.NativeWindow.supportsMenu &&
        nativeWindow.systemChrome != air.NativeWindowSystemChrome.NONE) {
        nativeWindow.menu = new air.NativeMenu();
        nativeWindow.menu.addEventListener(air.Event.SELECT, selectCommandMenu);
        fileMenu = nativeWindow.menu.addItem(new air.NativeMenuItem("File"));
        fileMenu.submenu = createFileMenu();

        editMenu = nativeWindow.menu.addItem(new air.NativeMenuItem("Edit"));
        editMenu.submenu = createEditMenu();
    }

    if (air.NativeApplication.supportsMenu) {
        application.menu.addEventListener(air.Event.SELECT, selectCommandMenu);
        fileMenu = application.menu.addItem(new air.NativeMenuItem("File"));
        fileMenu.submenu = createFileMenu();
        editMenu = application.menu.addItem(new air.NativeMenuItem("Edit"));
        editMenu.submenu = createEditMenu();
    }
}

function createFileMenu() {
    var fileMenu = new air.NativeMenu();
    fileMenu.addEventListener(air.Event.SELECT,selectCommandMenu);
```

```
    var newCommand = fileMenu.addItem(new air.NativeMenuItem("New"));
    newCommand.addEventListener(air.Event.SELECT, selectCommand);
    var saveCommand = fileMenu.addItem(new air.NativeMenuItem("Save"));
    saveCommand.addEventListener(air.Event.SELECT, selectCommand);
    var openFile = fileMenu.addItem(new air.NativeMenuItem("Open Recent"));
    openFile.submenu = new air.NativeMenu();
    openFile.submenu.addEventListener(air.Event.DISPLAYING, updateRecentDocumentMenu);
    openFile.submenu.addEventListener(air.Event.SELECT, selectCommandMenu);

    return fileMenu;
}

function createEditMenu() {
    var editMenu = new air.NativeMenu();
    editMenu.addEventListener(air.Event.SELECT,selectCommandMenu);

    var copyCommand = editMenu.addItem(new air.NativeMenuItem("Copy"));
    copyCommand.addEventListener(air.Event.SELECT,selectCommand);
    copyCommand.keyEquivalent = "c";
    var pasteCommand = editMenu.addItem(new air.NativeMenuItem("Paste"));
    pasteCommand.addEventListener(air.Event.SELECT, selectCommand);
    pasteCommand.keyEquivalent = "v";
    editMenu.addItem(new air.NativeMenuItem("", true));
    var preferencesCommand = editMenu.addItem(new air.NativeMenuItem("Preferences"));
    preferencesCommand.addEventListener(air.Event.SELECT,selectCommand);

    return editMenu;
}

function updateRecentDocumentMenu(event) {
    air.trace("Updating recent document menu.");
    var docMenu = air.NativeMenu(event.target);

    for (var i = docMenu.numItems - 1; i >= 0; i--) {
        docMenu.removeItemAt(i);
    }

    for (var file in recentDocuments) {
        var menuItem =
            docMenu.addItem(new air.NativeMenuItem(recentDocuments[file].name));
        menuItem.data = recentDocuments[file];
        menuItem.addEventListener(air.Event.SELECT, selectRecentDocument);
    }
}

function selectRecentDocument(event) {
    air.trace("Selected recent document: " + event.target.data.name);
}

function selectCommand(event) {
    air.trace("Selected command: " + event.target.label);
}

function selectCommandMenu(event) {
    if (event.currentTarget.parent != null) {
        var menuItem = findItemForMenu(event.currentTarget);
        if(menuItem != null){
            air.trace("Select event for \"" + event.target.label +
            "\" command handled by menu: " + menuItem.label);
        }
    } else {
        air.trace("Select event for \"" + event.target.label +
```

```
                "\" command handled by root menu.");
    }
}

function findItemForMenu(menu){
    for (var item in menu.parent.items) {
        if (item != null) {
            if (item.submenu == menu) {
                return item;
            }
        }
    }
    return null;
}
</script>
<title>AIR menus</title>
</head>
<body onload="MenuExample()"></body>
</html>
```

MenuBuilder フレームワークの使用

Adobe AIR 1.0 およびそれ以降

Adobe AIR では、メニューを作成するときに標準のメニュークラスのほかにメニュービルダの JavaScript も利用できます。MenuBuilder フレームワークを使用すると、メニューの構造を XML 形式または JSON 形式で宣言的に定義できます。AIR アプリケーションで使用できる種類のメニューをヘルパーメソッドで作成することもできます。AIR でのネイティブメニューの使用方法について詳しくは、112 ページの「メニューの基本」を参照してください。

MenuBuilder フレームワークでのメニューの作成

Adobe AIR 1.0 およびそれ以降

MenuBuilder フレームワークを使用すると、メニューの構造を XML または JSON を使用して定義できます。このフレームワークには、メニュー構造を含むファイルを読み込んで解析するためのメソッドが含まれています。メニュー構造が読み込まれると、追加のメソッドを使用してアプリケーションでのメニューの使用方法を指定できます。例えば、Mac OS X のアプリケーションメニュー、ウィンドウメニューまたはコンテキストメニューとしてメニューを設定できます。

MenuBuilder フレームワークは、ランタイムに組み込まれていません。このフレームワークを使用するには、次に示すように、Adobe AIR SDK 内の AIRMenuBuilder.js ファイルをアプリケーションコードに含めます。

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

MenuBuilder フレームワークは、アプリケーションサンドボックスで実行するように設計されています。フレームワークのメソッドをクラシックサンドボックスから呼び出すことはできません。

フレームワークのすべての開発者用メソッドは、air.ui.Menu クラスにクラスメソッドとして定義されています。

MenuBuilder の基本的なワークフロー

Adobe AIR 1.0 およびそれ以降

通常、MenuBuilder フレームワークでは、いずれの種類のメニューを作成する場合でも、3 つの手順に従います。

1. **メニュー構造の定義** メニュー構造を定義する XML または JSON を含むファイルを作成します。メニューの種類によっては、最上位アイテムはメニューになります（ウィンドウメニューやアプリケーションメニューの場合など）。他の種類のメニューでは、最上位アイテムは個別のメニューコマンドです（コンテキストメニューの場合など）。メニュー構造の定義形式について詳しくは、127 ページの「MenuBuilder メニュー構造の定義」を参照してください。
2. **メニュー構造の読み込み** 必要に応じてメニュークラスメソッドとして Menu.createFromXML() または Menu.createFromJSON() を呼び出し、メニュー構造ファイルを読み込んで実際のメニューオブジェクトとして解析します。いずれのメソッドからも NativeMenu オブジェクトが返されます。このオブジェクトをフレームワークのいずれかのメニュー設定メソッドに渡すことができます。
3. **メニューの割り当て** メニューの用途に応じたメニュークラスメソッドを呼び出します。使用できるオプションは次のとおりです。
   - Menu.setAsMenu() ウィンドウメニューまたはアプリケーションメニューで使用
   - Menu.setAsContextMenu() DOM エレメントのコンテキストメニューとしてメニューを表示
   - Menu.setAsIconMenu() システムトレイまたはドックアイコンのコンテキストメニューとしてメニューを設定

コードを実行するタイミングが重要になります。特に、ウィンドウメニューは実際のオペレーティングシステムウィンドウが作成される前に割り当てる必要があります。ウィンドウメニューとしてメニューを設定する setAsMenu() 呼び出しは、onload などのイベントハンドラーではなく、HTML ページで直接実行する必要があります。メニューを作成するコードは、オペレーティングシステムによってウィンドウが開かれる前に実行する必要があります。同時に、DOM エレメントを参照する setAsContextMenu() 呼び出しは、DOM エレメントの作成後に発生する必要があります。最も安全な方法としては、メニュー割り当てコードを含む <script> ブロックを HTML ページの最後の </body> タグの内側に挿入します。

メニュー構造の読み込み

Adobe AIR 1.0 およびそれ以降

メニューの用途に関係なく、メニュー構造は XML 構造または JSON 構造を含む独立したファイルとして定義します。アプリケーションでメニューを割り当てる前に、フレームワークを使用してメニュー構造ファイルを読み込んで解析します。メニュー構造ファイルを読み込んで解析するには、次の 2 つのフレームワークメソッドのいずれかを使用します。

- Menu.createFromXML() XML 形式のメニュー構造ファイルの読み込みおよび解析用
- Menu.createFromJSON() JSON 形式のメニュー構造ファイルの読み込みおよび解析用

いずれのメソッドでも、引数としてメニュー構造ファイルのファイルパスを使用します。いずれのメソッドでも、そのファイルパスからファイルが読み込まれます。ファイルが読み込まれると、ファイルの内容が解析されてファイル内に定義されたメニュー構造の NativeMenu オブジェクトが返されます。例えば、次のコードでは読み込み元の HTML ファイルと同じディレクトリから「windowMenu.xml」というメニュー構造ファイルが読み込まれます。

```
var windowMenu = air.ui.Menu.createFromXML("windowMenu.xml");
```

次のコード例では、「menus」というディレクトリから「contextMenu.js」というメニュー構造ファイルが読み込まれます。

```
var contextMenu = air.ui.Menu.createFromJSON("menus/contextMenu.js");
```

注意：生成された NativeMenu オブジェクトは、アプリケーションメニューまたはウィンドウメニューとして 1 回だけ使用できます。ただし、コンテキストメニューまたはアイコンメニューとしては、アプリケーション内で繰り返して使用できます。MenuBuilder フレームワークを Mac OS X で使用する際に、同じ NativeMenu をアプリケーションメニューおよび別の種類のメニューとして割り当てると、アプリケーションメニューとしてのみ使用されます。

MenuBuilder フレームワークで使用できる特定のメニュー構造について詳しくは、127 ページの「MenuBuilder メニュー構造の定義」を参照してください。

アプリケーションメニューまたはウィンドウメニューの作成

Adobe AIR 1.0 およびそれ以降

MenuBuilder フレームワークを使用してアプリケーションメニューまたはウィンドウメニューを作成すると、メニューデータ構造の最上位のオブジェクトまたはノードはメニューバーに表示されるアイテムに対応します。最上位のアイテム内にネストされたアイテムによって、各メニューコマンドが定義されます。それらのメニューアイテム内に他のアイテムが含まれる場合もあります。その場合、メニューアイテムはコマンドではなくサブメニューになります。ユーザーがメニューアイテムを選択すると、その中のメニューアイテムが展開されます。

呼び出しが実行されるウィンドウのアプリケーションメニューまたはウィンドウメニューとしてメニューを設定するには、Menu.setAsMenu() メソッドを使用します。setAsMenu() メソッドは、NativeMenu オブジェクトをパラメーターとして使用します。次の例では、XML ファイルを読み込み、生成されたメニューをアプリケーションメニューまたはウィンドウメニューとして設定します。

```
var windowMenu = air.ui.Menu.createFromXML("windowMenu.xml");
air.ui.Menu.setAsMenu(windowMenu);
```

ウィンドウメニューをサポートするオペレーティングシステムでは、setAsMenu() 呼び出しにより、現在のウィンドウ（window.nativeWindow として示されるウィンドウ）のウィンドウメニューとしてメニューが設定されます。アプリケーションメニューをサポートするオペレーティングシステムでは、アプリケーションメニューとしてメニューが使用されます。

Mac OS X では、標準メニューのセットがデフォルトのアプリケーションメニューとして定義され、同じメニューアイテムのセットが各アプリケーションで使用されます。これらのメニューに含まれるアプリケーションメニューの名前は、アプリケーション名、編集メニューおよびウィンドウメニューと一致します。Menu.setAsMenu() メソッドを呼び出して NativeMenu オブジェクトをアプリケーションメニューとして割り当てると、NativeMenu 内のアイテムが編集メニューとウィンドウメニューの間の標準メニュー構造内に追加されます。標準メニューは変更または置換されません。

必要に応じて、標準メニューに追加せずに標準メニューを置換することができます。既存のメニューを置き換えるには、次の例に示すように、二番目の引数に値 true を設定して setAsMenu() 呼び出しに渡します。

```
air.ui.Menu.setAsMenu(windowMenu, true);
```

DOM エレメントのコンテキストメニューの作成

Adobe AIR 1.0 およびそれ以降

MenuBuilder フレームワークを使用して DOM エレメントのコンテキストメニューを作成するには、2 つの手順を使用します。まず、Menu.createFromXML() メソッドまたは Menu.createFromJSON() メソッドを使用してメニュー構造を定義する NativeMenu インスタンスを作成します。次に、Menu.setAsContextMenu() メソッドを呼び出して、そのメニューを DOM エレメントのコンテキストメニューとして割り当てます。コンテキストメニューは単一メニューで構成されるので、メニューデータ構造の最上位のメニューアイテムが単一メニュー内のアイテムになります。子のメニューアイテムを含むメニューアイテムはサブメニューになります。NativeMenu を DOM エレメントのコンテキストメニューとして割り当てるには、Menu.setAsContextMenu() メソッドを呼び出します。このメソッドでは、コンテキストメニューとして設定する NativeMenu とそれを割り当てる DOM エレメントの id（ストリング）を 2 つのパラメーターとして使用します。

```
var treeContextMenu = air.ui.Menu.createFromXML("treeContextMenu.xml");
air.ui.Menu.setAsContextMenu(treeContextMenu, "navTree");
```

DOM エレメントのパラメーターを省略すると、呼び出し元の HTML ドキュメントがデフォルト値として使用されます。つまり、メニューは HTML ドキュメントのウィンドウ全体のコンテキストメニューとして設定されます。HTML ウィンドウ全体からデフォルトのコンテキストメニューを簡単に削除するには、次の例に示すように、最初のパラメーターに null を渡します。

```
air.ui.Menu.setAsContextMenu(null);
```

DOM エレメントから割り当てたコンテキストメニューを削除することもできます。setAsContextMenu() メソッドを呼び出して null とエレメント id を、2 つの引数として渡します。

アイコンのコンテキストメニューの作成

Adobe AIR 1.0 およびそれ以降

Adobe AIR アプリケーションでは、アプリケーションウィンドウ内の DOM エレメント用のコンテキストメニューに加えて、他の 2 つの特別なコンテキストメニューをサポートしています。ドックアイコンメニューと、システムトレイを用いるオペレーティングシステムのシステムトレイアイコンメニューです。これらのメニューを設定するには、まず、Menu.createFromXML() メソッドまたは Menu.createFromJSON() メソッドを使用して NativeMenu を作成します。次に、Menu.setAsIconMenu() メソッドを呼び出して、その NativeMenu をドックアイコンメニューまたはシステムトレイアイコンメニューとして割り当てます。

このメソッドでは 2 つの引数を使用します。最初の引数は必須で、アイコンメニューとして使用する NativeMenu です。2 番目の引数は Array で、アイコンとして使用する画像へのファイルパスのストリングまたはアイコンの画像データを含む BitmapData オブジェクトが入ります。この引数は、application.xml ファイルにデフォルトアイコンが指定されていない場合は、必須になります。application.xml ファイルにデフォルトアイコンが指定されている場合は、そのアイコンがシステムトレイアイコンのデフォルトとして使用されます。

次の例では、メニューデータを読み込み、そのメニューをドックアイコンまたはシステムトレイアイコンのコンテキストメニューとして割り当てる方法を示します。

```
// Assumes that icons are specified in the application.xml file.
// Otherwise the icons would need to be specified using a second
// parameter to the setAsIconMenu() function.
var iconMenu = air.ui.Menu.createFromXML("iconMenu.xml");
air.ui.Menu.setAsIconMenu(iconMenu);
```

注意：Mac OS X では、アプリケーションのドックアイコン用に標準のコンテキストメニューが定義されています。メニューをドックアイコンのコンテキストメニューとして割り当てると、メニュー内のアイテムが OS の標準メニューアイテムの上に表示されます。標準のメニューアイテムの削除、アクセスまたは変更を行うことはできません。

MenuBuilder メニュー構造の定義

Adobe AIR 1.0 およびそれ以降

Menu.createFromXML() メソッドまたは Menu.createFromJSON() メソッドを使用して NativeMenu オブジェクトを作成すると、XML エレメントやオブジェクトの構造が結果のメニュー構造に反映されます。作成されたメニューの構造やプロパティは実行時に変更できます。実行時にメニューアイテムを変更するには、NativeMenu オブジェクトの階層構造内を移動して NativeMenuItem オブジェクトにアクセスします。

MenuBuilder フレームワークは、メニューデータソースを解析する際に、特定の XML 属性またはオブジェクトプロパティを検索します。それらの属性またはプロパティが存在する場合、その値によって作成されるメニューの構造が決まります。

メニュー構造として XML を使用する場合は、その XML ファイル内にルートノードが必要です。ルートノードの子ノードは、最上位のメニューアイテムのノードとして使用されます。XML ノードには任意の名前を付けることができます。XML ノードの名前はメニュー構造に影響しません。メニューの定義には、ノードの階層構造と属性値のみが使用されます。

メニューアイテムのタイプ

Adobe AIR 1.0 およびそれ以降

メニューデータソースの各エントリ（各 XML エレメントまたは JSON オブジェクト）では、それが表わすメニューアイテムのタイプとタイプ別の情報を指定できます。Adobe AIR では、次のメニューアイテムのタイプがサポートされています。これらのタイプは、データソースの type 属性またはプロパティの値として設定できます。

メニューアイテムのタイプ	説明
normal	デフォルトのタイプです。normal タイプのアイテムを選択すると、select イベントがトリガーされ、データソースの onSelect フィールドに指定された関数が呼び出されます。または、アイテムに子がいる場合は、メニューアイテムから preparing イベント、次に displaying イベントが送出され、その後サブメニューが開きます。
check	check タイプのアイテムを選択すると、NativeMenuItem の checked プロパティの true 値と false 値が切り替えられ、select イベントがトリガーされ、データソースの onSelect フィールドに指定された関数が呼び出されます。メニューアイテムに true が設定されている状態では、メニューのアイテムラベルの横にチェックマークが表示されます。
separator	separator タイプのアイテムは、メニュー内のアイテムをグループ分けするための単純な横線を描きます。

normal メニューアイテムに子がある場合は、サブメニューとして扱われます。XML データソースの場合は、メニューアイテムに他の XML エレメントが含まれることを意味します。JSON データソースの場合は、メニューアイテムを表すオブジェクトに、他のオブジェクトの配列を含む items というプロパティが渡されます。

メニューデータソースの属性またはプロパティ

Adobe AIR 1.0 およびそれ以降

メニューデータソースのアイテムには、アイテムの表示方法と動作を決定する XML 属性またはオブジェクトプロパティをいくつか指定できます。次の表は、指定できる属性、属性のデータ型、用途、およびデータソースでの属性の表現方法を示します。

属性またはプロパティ	型	説明
altKey	Boolean	アイテムのキーボードショートカットに Alt キーを含める必要があるかどうかを指定します。
cmdKey	Boolean	アイテムのキーボードショートカットに Command キーを含める必要があるかどうかを指定します。この値には、defaultKeyEquivalentModifiers フィールドも影響します。
ctrlKey	Boolean	アイテムのキーボードショートカットに Control キーを含める必要があるかどうかを指定します。この値には、defaultKeyEquivalentModifiers フィールドも影響します。
defaultKeyEquivalentModifiers	Boolean	アイテムのキーボードショートカットにオペレーティングシステムのデフォルトの修飾キー（Mac OS X の場合は Command キー、Windows の場合は Ctrl キー）を含める必要があるかどうかを指定します。この属性を指定しないと、MenuBuilder フレームワークではこのアイテムの値が true の場合と同じように扱われます。

属性またはプロパティ	型	説明
enabled	Boolean	ユーザーがメニューアイテムを選択できる場合は true、選択できない場合は false を指定します。この属性を指定しないと、MenuBuilder フレームワークではこのアイテムの値が true の場合と同じように扱われます。
商品	Array	（JSON のみ）メニューアイテム自体をメニューとして指定します。配列内のオブジェクトは、メニュー内の子メニューアイテムです。
keyEquivalent	String	押したときにメニューアイテムの選択イベントがトリガーされるキーボード文字を指定します。 この値が大文字である場合は、アイテムのキーボードショートカットに Shift キーを含める必要があります。
label	String	コントロールに表示されるテキストを指定します。このアイテムは、separator 以外のすべてのタイプのメニューアイテムに使用します。
mnemonicIndex	Integer	メニューアイテムのニーモニックとして使用するラベル内の文字のインデックス位置を指定します。また、ラベル内の文字の左側にアンダースコアを挿入して、その文字がメニューアイテムのニーモニックであることを示すこともできます。
onSelect	String または Function	関数の名前（String）または関数への参照（Function オブジェクト）を指定します。指定した関数は、ユーザーがメニューアイテムを選択したときにイベントリスナーとして呼び出されます。詳しくは、133 ページの「MenuBuilder メニューイベントの処理」を参照してください。
shiftKey	String	アイテムのキーボードショートカットに Shift キーを含める必要があるかどうかを指定します。 この値は、keyEquivalent の値によって指定される場合もあります。keyEquivalent の値が大文字である場合は、キーボードショートカットに Shift キーを含める必要があります。
toggled	Boolean	check アイテムを選択するかどうかを指定します。このプロパティを選択しないと、MenuBuilder フレームワークではアイテムの値が false の場合と同じように扱われ、アイテムが選択されません。
type	String	メニューアイテムのタイプを指定します。意味のある値は、separator および check です。MenuBuilder フレームワークでは、type 項目がない他のすべての値、エレメントまたはオブジェクトが、normal のメニュー項目として扱われます。

MenuBuilder フレームワークでは、他のすべてのオブジェクトプロパティまたは XML 属性が無視されます。

例：XML MenuBuilder データソース

Adobe AIR 1.0 およびそれ以降

次の例では、MenuBuilder フレームワークを使用してテキスト領域のコンテキストメニューを定義します。XML をデータソースとするメニュー構造の定義方法を示します。同じメニュー構造を JSON 配列で指定するアプリケーションについては、131 ページの「例：JSON MenuBuilder データソース」を参照してください。

アプリケーションは 2 つのファイルで構成されます。

最初のファイルはメニューデータソースで、「textContextMenu.xml」というファイルです。この例で使用するメニューアイテムノードの名前は「menuitem」ですが、XML ノードの実際の名前は重要ではありません。前述したように、XML の構造とその属性値のみが、生成されたメニューに影響します。

```
<?xml version="1.0" encoding="utf-8" ?>
<root>
    <menuitem label="MenuItem A"/>
    <menuitem label="MenuItem B" type="check" toggled="true"/>
    <menuitem label="MenuItem C" enabled="false"/>
    <menuitem type="separator"/>
    <menuitem label="MenuItem D">
        <menuitem label="SubMenuItem D-1"/>
        <menuitem label="SubMenuItem D-2"/>
        <menuitem label="SubMenuItem D-3"/>
    </menuitem>
</root>
```

2 番目のファイルは、アプリケーションユーザーインターフェイスのソースコードで、application.xml ファイルに初期ウィンドウとして指定された HTML ファイルです。

```
<html>
    <head>
        <title>XML-based menu data source example</title>
        <script type="text/javascript" src="AIRAliases.js"></script>
        <script type="text/javascript" src="AIRMenuBuilder.js"></script>
        <style type="text/css">
            #contextEnabledText
            {
                margin-left: auto;
                margin-right: auto;
                margin-top: 100px;
                width: 50%
            }
        </style>
    </head>
    <body>
        <div id="contextEnabledText">This block of text is context menu enabled. Right click or Command-
click on the text to view the context menu.</div>
        <script type="text/javascript">
            // Create a NativeMenu from "textContextMenu.xml" and set it
            // as context menu for the "contextEnabledText" DOM element:
            var textMenu = air.ui.Menu.createFromXML("textContextMenu.xml");
            air.ui.Menu.setAsContextMenu(textMenu, "contextEnabledText");

            // Remove the default context menu from the page:
            air.ui.Menu.setAsContextMenu(null);
        </script>
    </body>
</html>
```

例：JSON MenuBuilder データソース

Adobe AIR 1.0 およびそれ以降

次の例では、MenuBuilder フレームワークを使用して JSON 配列をデータソースとするテキスト領域のコンテキストメニューを定義します。同じメニュー構造を XML で指定するアプリケーションについては、130 ページの「例：XML MenuBuilder データソース」を参照してください。

アプリケーションは 2 つのファイルで構成されます。

最初のファイルはメニューデータソースで、「textContextMenu.js」というファイルです。

```
[
    {label: "MenuItem A"},
    {label: "MenuItem B", type: "check", toggled: "true"},
    {label: "MenuItem C", enabled: "false"},
    {type: "separator"},
    {label: "MenuItem D", items:
        [
            {label: "SubMenuItem D-1"},
            {label: "SubMenuItem D-2"},
            {label: "SubMenuItem D-3"}
        ]
    }
]
```

2 番目のファイルは、アプリケーションユーザーインターフェイスのソースコードで、application.xml ファイルに初期ウィンドウとして指定された HTML ファイルです。

```
<html>
    <head>
        <title>JSON-based menu data source example</title>
        <script type="text/javascript" src="AIRAliases.js"></script>
        <script type="text/javascript" src="AIRMenuBuilder.js"></script>
        <style type="text/css">
            #contextEnabledText
            {
                margin-left: auto;
                margin-right: auto;
                margin-top: 100px;
                width: 50%
            }
        </style>
    </head>
    <body>
        <div id="contextEnabledText">This block of text is context menu enabled. Right click or Command-
click on the text to view the context menu.</div>
        <script type="text/javascript">
            // Create a NativeMenu from "textContextMenu.js" and set it
            // as context menu for the "contextEnabledText" DOM element:
            var textMenu = air.ui.Menu.createFromJSON("textContextMenu.js");
            air.ui.Menu.setAsContextMenu(textMenu, "contextEnabledText");

            // Remove the default context menu from the page:
            air.ui.Menu.setAsContextMenu(null);
        </script>
    </body>
</html>
```

MenuBuilder によるメニューへのキーボード機能の追加

Adobe AIR 1.0 およびそれ以降

オペレーティングシステム固有のメニューで使用できるキーボードショートカットは、Adobe AIR でも使用できます。メニューデータソースでは、メニューコマンド用およびニーモニック用の 2 種類のキーボードショートカットを指定できます。

メニューのキーボードショートカットの指定

Adobe AIR 1.0 およびそれ以降

ウィンドウメニューまたはアプリケーションメニューコマンドには、キーボードショートカット（アクセラレーター）を指定できます。キーまたはキーの組み合わせを押すと、データソースに指定された select イベントと onSelect イベントハンドラーが NativeMenuItem から送出されます。この動作は、ユーザーがメニューアイテムを選択した場合と同じです。

メニューのキーボードショートカットについて詳しくは、115 ページの「ネイティブメニューコマンドと等価なキー（AIR）」を参照してください。

MenuBuilder フレームワークを使用して、データソースの対応するノードに、メニューアイテムのキーボードショートカットを指定できます。データソースに keyEquivalent フィールドがある場合、MenuBuilder フレームワークによってその値がキーボードショートカットとして使用されます。

キーボードショートカットに含める修飾キーを指定することもできます。修飾キーを追加するには、altKey フィールド、ctrlKey フィールド、cmdKey フィールドまたは shiftKey フィールドに true を指定します。指定したキーがキーボードショートカットに含まれます。デフォルトでは、Windows の場合は Ctrl キーが追加され、Mac OS X では Command キーが追加されます。このデフォルトの動作をオーバーライドするには、defaultKeyEquivalentModifiers フィールドを false に設定して含めます。

次の例は、キーボードショートカットを含む XML ベースのメニューデータソースのデータ構造で、「keyEquivalentMenu.xml」というファイルに含まれています。

```
<?xml version="1.0" encoding="utf-8" ?>
<root>
    <menuitem label="File">
        <menuitem label="New" keyEquivalent="n"/>
        <menuitem label="Open" keyEquivalent="o"/>
        <menuitem label="Save" keyEquivalent="s"/>
        <menuitem label="Save As..." keyEquivalent="s" shiftKey="true"/>
        <menuitem label="Close" keyEquivalent="w"/>
    </menuitem>
    <menuitem label="Edit">
        <menuitem label="Cut" keyEquivalent="x"/>
        <menuitem label="Copy" keyEquivalent="c"/>
        <menuitem label="Paste" keyEquivalent="v"/>
    </menuitem>
</root>
```

次の例のアプリケーションでは、「keyEquivalentMenu.xml」からメニュー構造を読み込み、それをアプリケーションのウィンドウメニューまたはアプリケーションメニューの構造として使用します。

```
<html>
    <head>
        <title>XML-based menu with key equivalents example</title>
        <script type="text/javascript" src="AIRAliases.js"></script>
        <script type="text/javascript" src="AIRMenuBuilder.js"></script>
    </head>
    <body>
        <script type="text/javascript">
            // Create a NativeMenu from "keyEquivalentMenu.xml" and set it
            // as the application/window menu
            var keyEquivMenu = air.ui.Menu.createFromXML("keyEquivalentMenu.xml");
            air.ui.Menu.setAsMenu(keyEquivMenu);
        </script>
    </body>
</html>
```

メニューアイテムのニーモニックの指定

Adobe AIR 1.0 およびそれ以降

メニューアイテムのニーモニックとは、メニューアイテムに関連付けられたキーのことです。メニューが表示されているときに、そのキーを押すと、関連付けられたメニューアイテムコマンドがトリガーされます。この動作は、ユーザーがそのメニューアイテムをマウスで選択した場合と同じです。通常、メニューアイテムのニーモニックには、オペレーティングシステムによって、メニューアイテム名内の該当する文字にアンダーラインが付けられます。

ニーモニックについて詳しくは、116 ページの「ニーモニック（AIR）」を参照してください。

MenuBuilder フレームワークでは、メニューアイテムのニーモニックを指定する最も簡単な方法として、メニューアイテムの label フィールドにアンダースコア文字（「_」）を含めます。メニューアイテムのニーモニックとする文字の直後にアンダースコア文字を挿入します。例えば、MenuBuilder フレームワークを使用して読み込まれたデータソースで次の XML ノードが使用されている場合、このコマンドのニーモニックは、2 番目の単語の先頭文字（「A」）です。

```
<menuitem label="Save _As"/>
```

NativeMenu オブジェクトの作成時には、このアンダースコア文字がラベルに含まれません。その代わり、アンダースコアの後続の文字がメニューアイテムのニーモニックになります。メニューアイテム名にリテラルアンダースコア文字を含めるには、アンダースコア文字を 2 つ（「__」）使用します。この 2 つのアンダースコア文字は、メニューアイテムラベルで 1 つのアンダースコアに変換されます。

label フィールドにアンダースコア文字を使用する代わりに、ニーモニック文字の整数のインデックス位置を指定することもできます。メニューアイテムのデータソースまたは XML エレメントの mnemonicIndex フィールドにインデックスを指定します。

MenuBuilder メニューイベントの処理

Adobe AIR 1.0 およびそれ以降

NativeMenu のユーザー操作はイベント駆動型です。ユーザーがメニューアイテムを選択したり、メニューやサブメニューを開くと、NativeMenuItem オブジェクトによってイベントが送出されます。MenuBuilder フレームで作成した NativeMenu オブジェクトでは、個別の NativeMenuItem オブジェクトや NativeMenu にイベントリスナーを登録できます。これらのイベントの登録および応答の方法は、NativeMenu オブジェクトおよび NativeMenuItem オブジェクトを MenuBuilder フレームワークを使用せずに手動で作成した場合と同じです。詳しくは、114 ページの「メニューイベント」を参照してください。

MenuBuilder フレームワークでは、標準のイベント処理を補完する方法として、メニューデータソース内のメニューアイテムに select イベントハンドラー関数を指定できます。メニューアイテムデータソースに onSelect フィールドを指定すると、ユーザーがそのメニューアイテムを選択したときに、指定した関数が呼び出されます。例えば、MenuBuilder フレームワークを使用して読み込んだデータソースに次の XML ノードが含まれているとします。該当するメニューアイテムを選択すると、doSave() という関数が呼び出されます。

```
<menuitem label="Save" onSelect="doSave"/>
```

onSelect フィールドは、XML データソースでは String です。JSON 配列では、関数名の String を使用できます。また、JSON 配列でのみ、関数への変数参照をオブジェクトとして使用できます。ただし、JSON 配列で関数への変数参照を使用する場合は、onload イベントハンドラーの前または実行時にメニューを作成する必要があります。そうしない場合は、JavaScript セキュリティ違反が発生します。いずれの場合でも、指定した関数はグローバルスコープで定義する必要があります。

指定した関数が呼び出されると、その関数に対してランタイムから 2 つの引数が渡されます。最初の引数は、select イベントから送出されるイベントオブジェクトです。このオブジェクトは Event クラスのインスタンスです。関数に渡される 2 番目の引数は、メニューアイテムの作成に使用されたデータを含む匿名オブジェクトです。このオブジェクトには次のプロパティがあります。各プロパティの値は、元のデータ構造の値と一致します。元のデータ構造に設定されていないプロパティの値は null になります。

- altKey
- cmdKey
- ctrlKey
- defaultKeyEquivalentModifiers
- enabled
- keyEquivalent
- label
- mnemonicIndex
- onSelect
- shiftKey
- toggled
- type

次の例では、NativeMenu イベントを試すことができます。この例には 2 つのメニューが含まれています。ウィンドウおよびアプリケーションメニューは、XML データソースを使用して作成されます。 エレメントおよび <li> エレメントで示されるアイテムのリストのコンテキストメニューは、JSON 配列データソースを使用して作成されます。画面のテキスト領域には、ユーザーがメニューアイテムを選択したときに、各イベントに関する情報が表示されます。

アプリケーションのソースコードを次に示します。

```
<html>
    <head>
        <title>Menu event handling example</title>
        <script type="text/javascript" src="AIRAliases.js"></script>
        <script type="text/javascript" src="AIRMenuBuilder.js"></script>
        <script type="text/javascript" src="printObject.js"></script>
        <script type="text/javascript">
            function fileMenuCommand(event, data) {
                print("fileMenuCommand", event, data);
            }

            function editMenuCommand(event, data) {
                print("editMenuCommand", event, data);
            }

            function moveItemUp(event, data) {
                print("moveItemUp", event, data);
            }

            function moveItemDown(event, data) {
                print("moveItemDown", event, data);
            }

            function print(command, event, data) {
                var result = "";
                result += "<h1>Command: " + command + '</h1>';
                result += "<p>" + printObject(event) + "</p>";
                result += "<p>Data:</p>";
                result += "<ul>";
                for (var s in data) {
                    result += "<li>" + s + ": " + printObject(data[s]) + "</li>";
                }
                result += "</ul>";

                var o = document.getElementById("output");
                o.innerHTML = result;
            }
        </script>
        <style type="text/css">
            #contextList {
                position: absolute; left: 0; top: 25px; bottom: 0; width: 100px;
                background: #eeeeee;
            }
            #output {
                position: absolute; left: 125px; top: 25px; right: 0; bottom: 0;
            }
        </style>
    </head>
    <body>
        <div id="contextList">
            <ul>
```

```
                <li>List item 1</li>
                <li>List item 2</li>
                <li>List item 3</li>
            </ul>
        </div>
        <div id="output">
            Choose menu commands. Information about the events displays here.
        </div>
        <script type="text/javascript">
            var mainMenu = air.ui.Menu.createFromXML("mainMenu.xml");
            air.ui.Menu.setAsMenu(mainMenu);

            var listContextMenu = air.ui.Menu.createFromJSON("listContextMenu.js");
            air.ui.Menu.setAsContextMenu(listContextMenu, "contextList")

            // clear the default context menu
            air.ui.Menu.setAsContextMenu(null);
        </script>
    </body>|
</html>
```

メインメニューのデータソース（「mainMenu.xml」）を次に示します。

```
<?xml version="1.0" encoding="utf-8" ?>
<root>
    <menuitem label="File">
        <menuitem label="New" keyEquivalent="n" onSelect="fileMenuCommand"/>
        <menuitem label="Open" keyEquivalent="o" onSelect="fileMenuCommand"/>
        <menuitem label="Save" keyEquivalent="s" onSelect="fileMenuCommand"/>
        <menuitem label="Save As..." keyEquivalent="S" onSelect="fileMenuCommand"/>
        <menuitem label="Close" keyEquivalent="w" onSelect="fileMenuCommand"/>
    </menuitem>
    <menuitem label="Edit">
        <menuitem label="Cut" keyEquivalent="x" onSelect="editMenuCommand"/>
        <menuitem label="Copy" keyEquivalent="c" onSelect="editMenuCommand"/>
        <menuitem label="Paste" keyEquivalent="v" onSelect="editMenuCommand"/>
    </menuitem>
</root>
```

コンテキストメニューのデータソース（「listContextMenu.js」）を次に示します。

```
[
    {label: "Move Item Up", onSelect: "moveItemUp"},
    {label: "Move Item Down", onSelect: "moveItemDown"}
]
```

printObject.js ファイルのコードを次に示します。このファイルに含まれている printObject() 関数は、アプリケーションで使用されますが、例でのメニュー操作には影響しません。

```
function printObject(obj) {
    if (!obj) {
        if (typeof obj == "undefined") { return "[undefined]"; };
        if (typeof obj == "object") { return "[null]"; };
        return "[false]";
    } else {
        if (typeof obj == "boolean") { return "[true]"; };
        if (typeof obj == "object") {
            if (typeof obj.length == "number") {
                var ret = [];
                for (var i=0; i<obj.length; i++) {
                    ret.push(printObject(obj[i]));
                }
                return ["[", ret.join(", "), "]"].join(" ");
            } else {
                var ret = [];
                var hadChildren = false;
                for (var k in obj) {
                    hadChildren = true;
                    ret.push ([k, " => ", printObject(obj[k])]);
                }
                if (hadChildren) {
                    return ["{\n", ret.join(",\n"), "\n}"].join("");
                }
            }
        }
        if (typeof obj == "function") { return "[Function]"; }
        return String(obj);
    }
}
```

第 10 章：AIR のタスクバーアイコン

Adobe AIR 1.0 およびそれ以降

多くのオペレーティングシステムには、アプリケーションを表すアイコンを格納できるタスクバー（Mac OS X のドックなど）があります。Adobe® AIR® には、NativeApplication.nativeApplication.icon プロパティを通じてアプリケーションタスクバーアイコンを操作するためのインターフェイスが用意されています。

関連項目

flash.desktop.NativeApplication

flash.desktop.DockIcon

flash.desktop.SystemTrayIcon

タスクバーアイコンについて

Adobe AIR 1.0 およびそれ以降

AIR では、NativeApplication.nativeApplication.icon オブジェクトが自動的に作成されます。オブジェクトのタイプは、オペレーティングシステムに応じて、DockIcon または SystemTrayIcon のいずれかになります。現在のオペレーティングシステムでこれらの InteractiveIcon サブクラスのどちらがサポートされるかは、NativeApplication.supportsDockIcon プロパティと NativeApplication.supportsSystemTrayIcon プロパティを使用して確認できます。InteractiveIcon 基本クラスには、width プロパティ、height プロパティおよび bitmaps プロパティが用意されており、これらを使用してアイコンで使用するイメージを変更することができます。ただし、DockIcon または SystemTrayIcon 固有のプロパティを対応していないオペレーティングシステムで調べると、ランタイムエラーが発生します。

アイコンに使用するイメージを設定または変更するには、1 つ以上のイメージを格納する配列を作成し、その配列を NativeApplication.nativeApplication.icon.bitmaps プロパティに割り当てます。タスクバーアイコンのサイズは、オペレーティングシステムの種類によって異なる場合があります。サイズ調整による画質の劣化を防ぐには、複数のサイズのイメージを bitmaps 配列に追加します。複数のイメージを追加した場合は、AIR によってタスクバーアイコンの現在の表示サイズに最も近いサイズが選択され、必要な場合にのみサイズが調整されます。次の例では、2 つのイメージを使用してタスクバーアイコンのイメージを設定しています。

```
air.NativeApplication.nativeApplication.icon.bitmaps =
            [bmp16x16.bitmapData, bmp128x128.bitmapData];
```

アイコンイメージを変更するには、1 つ以上の新しいイメージを格納する配列を bitmaps プロパティに割り当てます。enterFrame イベントまたは timer イベントに応答してイメージを変更すると、アイコンをアニメーション化することができます。

Windows および Linux で通知領域からアイコンを削除するか、または Mac OS X でアイコンの外観をデフォルトに戻すには、bitmaps に空の配列を設定します。

```
air.NativeApplication.nativeApplication.icon.bitmaps = [];
```

ドックアイコン

Adobe AIR 1.0 およびそれ以降

NativeApplication.supportsDockIcon が true の場合、AIR でドックアイコンがサポートされます。NativeApplication.nativeApplication.icon プロパティは、ドックのアプリケーションアイコンを表します（ウィンドウドックアイコンは除く）。

注意：AIR では、Mac OS X でのドックのウィンドウアイコンの変更はサポートされません。また、アプリケーションドックアイコンに対する変更はアプリケーションの実行中にのみ適用され、アプリケーションが終了されるとアイコンの外観は標準の外観に戻ります。

ドックアイコンのメニュー

Adobe AIR 1.0 およびそれ以降

コマンドを格納する NativeMenu オブジェクトを作成し、そのオブジェクトを NativeApplication.nativeApplication.icon.menu プロパティに割り当てることで、標準のドックメニューにコマンドを追加することができます。メニューのアイテムは、標準のドックアイコンメニューアイテムの上に表示されます。

ドックのバウンス

Adobe AIR 1.0 およびそれ以降

NativeApplication.nativeApplication.icon.bounce() メソッドを呼び出すことにより、ドックアイコンをバウンスさせることができます。bounce() priority パラメーターを informational に設定すると、アイコンが 1 回バウンスします。critical に設定すると、ユーザーがアプリケーションをアクティブにするまでアイコンがバウンスします。priority パラメーターの定数は、NotificationType クラスで定義されています。

注意：アプリケーションが既にアクティブになっている場合は、アイコンはバウンスしません。

ドックアイコンのイベント

Adobe AIR 1.0 およびそれ以降

ドックアイコンがクリックされると、NativeApplication オブジェクトから invoke イベントが送出されます。アプリケーションが実行されていない場合は、システムによってアプリケーションが起動されます。それ以外の場合は、invoke イベントは実行中のアプリケーションインスタンスに送られます。

システムトレイアイコン

Adobe AIR 1.0 およびそれ以降

NativeApplication.supportsSystemTrayIcon が true の場合、AIR でシステムトレイアイコンがサポートされます（現在、Windows と大半の Linux ディストリビューションのみ該当します）。Windows および Linux では、システムトレイアイコンはタスクバーの通知領域に表示されます。デフォルトではアイコンは表示されません。アイコンを表示するには、BitmapData オブジェクトを格納する配列をアイコンの bitmaps プロパティに割り当てます。アイコンイメージを変更するには、新しいイメージを格納する配列を bitmaps に割り当てます。アイコンを削除するには、bitmaps を null に設定します。

システムトレイアイコンのメニュー

Adobe AIR 1.0 およびそれ以降

NativeMenu オブジェクトを作成し、そのオブジェクトを NativeApplication.nativeApplication.icon.menu プロパティに割り当てることで、システムトレイアイコンにメニューを追加することができます（オペレーティングシステムのデフォルトのメニューはありません）。システムトレイアイコンのメニューにアクセスするには、アイコンを右クリックします。

システムトレイアイコンのツールチップ

Adobe AIR 1.0 およびそれ以降

アイコンにツールチップを追加するには、tooltip プロパティを設定します。

```
air.NativeApplication.nativeApplication.icon.tooltip = "Application name";
```

システムトレイアイコンのイベント

Adobe AIR 1.0 およびそれ以降

NativeApplication.nativeApplication.icon プロパティによって参照される SystemTrayIcon オブジェクトから、click、mouseDown、mouseUp、rightClick、rightMouseDown、rightMouseUp の各イベントに対して ScreenMouseEvent が送出されます。これらのイベントをアイコンメニューと一緒に使用すると、アプリケーションに可視のウィンドウがない場合にユーザーがそのアプリケーションを操作できるようになります。

例：ウィンドウがないアプリケーションの作成

Adobe AIR 1.0 およびそれ以降

次の例では、システムトレイアイコンがあり、可視のウィンドウはない AIR アプリケーションを作成します（アプリケーションの visible プロパティは、アプリケーション記述で true に設定しないようにする必要があります。さもなければ、アプリケーションの起動時にウィンドウが表示されます）。

注意：Flex WindowedApplication コンポーネントを使用している場合、WindowedApplication タグの visible 属性を false に設定する必要があります。この属性は、アプリケーション記述の設定より優先されます。

```
<html>
<head>
<script src="AIRAliases.js" language="JavaScript" type="text/javascript"></script>
<script language="JavaScript" type="text/javascript">
    var iconLoadComplete = function(event)
    {
        air.NativeApplication.nativeApplication.icon.bitmaps = [event.target.content.bitmapData];
    }

    air.NativeApplication.nativeApplication.autoExit = false;
    var iconLoad = new air.Loader();
    var iconMenu = new air.NativeMenu();
    var exitCommand = iconMenu.addItem(new air.NativeMenuItem("Exit"));
    exitCommand.addEventListener(air.Event.SELECT,function(event){
            air.NativeApplication.nativeApplication.icon.bitmaps = [];
            air.NativeApplication.nativeApplication.exit();
    });

    if (air.NativeApplication.supportsSystemTrayIcon) {
        air.NativeApplication.nativeApplication.autoExit = false;
        iconLoad.contentLoaderInfo.addEventListener(air.Event.COMPLETE,iconLoadComplete);
        iconLoad.load(new air.URLRequest("icons/AIRApp_16.png"));
        air.NativeApplication.nativeApplication.icon.tooltip = "AIR application";
        air.NativeApplication.nativeApplication.icon.menu = iconMenu;
    }

    if (air.NativeApplication.supportsDockIcon) {
        iconLoad.contentLoaderInfo.addEventListener(air.Event.COMPLETE,iconLoadComplete);
        iconLoad.load(new air.URLRequest("icons/AIRApp_128.png"));
        air.NativeApplication.nativeApplication.icon.menu = iconMenu;
    }

</script>
</head>
<body>
</body>
</html>
```

注意：この例では、AIRApp_16.png および AIRApp_128.png という名前のイメージファイルがアプリケーションの icons サブディレクトリに含まれていることを前提にしています（サンプルのアイコンファイルは AIR SDK に含まれており、プロジェクトフォルダーにコピーできます）。

ウィンドウのタスクバーのアイコンとボタン

Adobe AIR 1.0 およびそれ以降

ウィンドウを表すアイコンは、通常はタスクバーまたはドックのウィンドウ領域に表示されます。ユーザーはこれらを使用して、バックグラウンドで実行されているウィンドウや最小化されているウィンドウに簡単にアクセスすることができます。Mac OS X のドックには、アプリケーションのアイコンと最小化された各ウィンドウのアイコンが表示されます。Microsoft Windows および Linux では、アプリケーションで表示する通常種類の各ウィンドウを表すボタンがタスクバーに表示され、各ボタンにプログラムアイコンとタイトルが表示されます。

タスクバーのウィンドウボタンの強調表示

Adobe AIR 1.0 およびそれ以降

ウィンドウがバックグラウンドで実行されているときに、ウィンドウに関連するイベントが発生したことをユーザーに通知することができます。Mac OS X では、アプリケーションドックアイコンをバウンスさせてユーザーに通知できます（139 ページの「ドックのバウンス」の説明を参照）。Windows および Linux では、NativeWindow インスタンスの notifyUser() メソッドを呼び出すと、そのウィンドウに対応するタスクバーボタンが強調表示されます。メソッドに渡される type パラメーターによって、通知の緊急度が決まります。

- NotificationType.CRITICAL：ユーザーがウィンドウを前面に表示するまでウィンドウアイコンが点滅します。
- NotificationType.INFORMATIONAL：別の色でウィンドウアイコンが強調表示されます。

  注意：Linux では、サポートされている通知の種類は INFORMATIONAL のみです。notifyUser() 関数にどの type 値を渡しても同じ効果になります。

  次のステートメントでは、ウィンドウのタスクバーボタンを強調表示します。

  ```
  window.nativeWindow.notifyUser(air.NotificationType.INFORMATIONAL);
  ```

  ウィンドウレベルの通知がサポートされていないオペレーティングシステムでは、NativeWindow.notifyUser() メソッドを呼び出しても効果はありません。ウィンドウの通知がサポートされているかどうかを確認するには、NativeWindow.supportsNotification プロパティを使用します。

タスクバーのボタンまたはアイコンがないウィンドウの作成

Adobe AIR 1.0 およびそれ以降

Windows オペレーティングシステムでは、**utility** または **lightweight** のタイプで作成されたウィンドウはタスクバーに表示されません。また、不可視のウィンドウもタスクバーに表示されません。

初期ウィンドウのタイプは必ず **normal** であるため、ウィンドウがタスクバーに表示されないアプリケーションを作成するには、初期ウィンドウを閉じるか、不可視のままにしておく必要があります。アプリケーションを終了しないでアプリケーションのすべてのウィンドウを閉じるには、最後のウィンドウを閉じる前に、NativeApplication オブジェクトの autoExit プロパティを false に設定します。単に初期ウィンドウが可視にならないようにするには、<visible>false</visible> をアプリケーション記述ファイルの <initalWindow> エレメントに追加します（さらに、visible プロパティを true に設定したり、ウィンドウの activate() メソッドを呼び出したりしないようにします）。

アプリケーションで開かれる新しいウィンドウでは、ウィンドウコンストラクターに渡される NativeWindowInitOption オブジェクトの type プロパティを NativeWindowType.UTILITY または NativeWindowType.LIGHTWEIGHT に設定します。

Mac OS X では、最小化されたウィンドウはドックタスクバーに表示されます。最小化されたアイコンが表示されないようにするには、ウィンドウを最小化するのではなく非表示にします。次の例では、nativeWindowDisplayState 変更イベントをリッスンし、ウィンドウが最小化される場合にキャンセルします。その代わりに、ハンドラーでウィンドウの visible プロパティを false に設定します。

```
function preventMinimize(event){
    if(event.afterDisplayState == air.NativeWindowDisplayState.MINIMIZED){
        event.preventDefault();
        event.target.visible = false;
    }
}
```

visible プロパティを false に設定している場合に Mac OS X ドックでウィンドウが最小化されても、ドックアイコンが削除されるわけではありません。ユーザーは、引き続きアイコンをクリックしてウィンドウを再表示することができます。

第 11 章：ファイルシステムの操作

Flash Player 9 以降、Adobe AIR 1.0 以降

Adobe® AIR® ファイルシステム API により、ホストコンピューターのファイルシステムに対して様々な操作を実行できます。これらのクラスを使用すると、ディレクトリやファイルにアクセスしてそれらを管理したり、ディレクトリやファイルを作成したり、ファイルにデータを書き込んだりすることができます。

関連項目

flash.filesystem.File

flash.filesystem.FileStream

AIR ファイルシステム API の使用

Adobe AIR 1.0 およびそれ以降

Adobe AIR ファイルシステム API には、次のクラスがあります。

- File
- FileMode
- FileStream

AIR ファイルシステム API を利用して、次のような処理を実行できます（これに限りません）。

- ディレクトリのコピー、作成、削除、移動
- ファイルおよびディレクトリに関する情報の取得
- ファイルの読み取りと書き込み

AIR ファイルの基礎

Adobe AIR 1.0 およびそれ以降

AIR でのファイルシステムの操作に関する簡単な説明およびコード例については、Adobe Developer Connection で次のクイックスタートの記事を参照してください。

- テキストファイルエディターの構築
- ディレクトリ検索アプリケーションの構築
- XML 環境設定ファイルの読み取りと書き込み

Adobe AIR には、ファイルとフォルダーの両方に対するアクセス、作成および管理に使用できるクラスが用意されています。これらのクラスは、flash.filesystem パッケージに含まれており、次のように使用します。

Adobe AIR には、ファイルとフォルダーの両方に対するアクセス、作成および管理に使用できるクラスが用意されています。これらのクラスは、runtime.flash.filesystem パッケージに含まれており、次のように使用します。

File クラス	説明
File	File オブジェクトは、ファイルまたはディレクトリへのパスを表します。File オブジェクトを使用してファイルまたはフォルダーへのポインターを作成することで、ファイルまたはフォルダーの操作を実行できるようになります。
FileMode	FileMode クラスは、fileMode パラメーターで使用されるストリング定数を定義します。このパラメーターは、FileStream クラスの open() メソッドおよび openAsync() メソッドで指定します。これらのメソッドの fileMode パラメーターによって、ファイルが開かれたときに FileStream オブジェクトで使用できる機能（書き込み、読み取り、追加および更新）が決まります。
FileStream	FileStream オブジェクトは、ファイルを開いて読み取りと書き込みを行うために使用されます。新規または既存のファイルを参照する File オブジェクトを作成してそのポインターを FileStream オブジェクトに渡すと、そのファイルを開いて、データの読み取りまたは書込みができるようになります。

File クラスのメソッドには、同期バージョンと非同期バージョンの両方を持つものがあります。

- File.copyTo() と File.copyToAsync()
- File.deleteDirectory() と File.deleteDirectoryAsync()
- File.deleteFile() と File.deleteFileAsync()
- File.getDirectoryListing() と File.getDirectoryListingAsync()
- File.moveTo() と File.moveToAsync()
- File.moveToTrash() と File.moveToTrashAsync()

また、FileStream 操作の処理も、同期的に行われる場合と非同期で行われる場合があります。どちらで処理されるかは、FileStream オブジェクトでファイルを開く際に open() メソッドを呼び出したか openAsync() メソッドを呼び出したかによって決まります。

非同期バージョンで処理を開始すると、バックグラウンドで処理が実行され、完了時（またはエラーイベントの発生時）にイベントが送出されます。これらの非同期のバックグラウンド処理を実行している間に、他のコードを実行することができます。非同期バージョンの操作では、イベントリスナー関数を設定する必要があります。イベントリスナー関数を設定するには、その関数を呼び出す File オブジェクトまたは FileStream オブジェクトの addEventListener() メソッドを使用します。

同期バージョンでは、イベントリスナーを設定する必要がない、より単純なコードを記述できます。ただし、同期メソッドの実行中は他のコードを実行できないので、表示オブジェクトのレンダリングやアニメーションなどの重要な処理が遅延することがあります。

AIR での File オブジェクトの操作

Adobe AIR 1.0 およびそれ以降

File オブジェクトは、ファイルシステム内のファイルまたはディレクトリへのポインターです。

File クラスは FileReference クラスを拡張します。FileReference クラス（Adobe® Flash® Player および AIR で利用可能）はファイルへのポインターを表します。File クラスには、セキュリティ上の考慮事項により Flash Player（ブラウザーで実行している SWF ファイル）で公開されていない追加のプロパティおよびメソッドがあります。

File クラスについて

Adobe AIR 1.0 およびそれ以降

File クラスを使用して以下を実行できます。

- 特殊なディレクトリへのパスの取得（ユーザーディレクトリ、ユーザーのドキュメントディレクトリ、アプリケーションが起動されたディレクトリ、アプリケーションディレクトリなど）

- ファイルおよびディレクトリのコピー
- ファイルおよびディレクトリの移動
- ファイルおよびディレクトリの削除（またはごみ箱への移動）
- ディレクトリに格納されているファイルおよびディレクトリの一覧の出力
- 一時ファイルおよび一時フォルダーの作成

File オブジェクトでファイルパスを参照すると、FileStream クラスでそれを使用してファイルデータの読み取りと書き込みができるようになります。

File オブジェクトでは、まだ存在しないファイルまたはディレクトリのパスを参照できます。このような File オブジェクトは、ファイルまたはディレクトリの作成に使用できます。

File オブジェクトのパス

Adobe AIR 1.0 およびそれ以降

各 File オブジェクトには、そのオブジェクトのパスをそれぞれ定義する 2 つのプロパティがあります。

プロパティ	説明
nativePath	ファイルへのプラットフォーム固有のパスを指定します。例えば、Windows では「c:¥Sample directory¥test.txt」、Mac OS では「/Sample directory/test.txt」のようになります。nativePath プロパティでは、ディレクトリの区切り文字として、Windows では円記号（¥）、Mac OS および Linux ではスラッシュ（/）を使用します。
url	file URL スキームを使用してファイルを参照できます。例えば、Windows では「file:///c:/Sample%20directory/test.txt」、Mac OS では「file:///Sample%20directory/test.txt」のようになります。ランタイムには、file 以外にも特殊な URL スキームが含まれます。それらについては、151 ページの「サポートされる AIR URL スキーム」で説明します。

File クラスには、Mac OS、Windows および Linux の標準のディレクトリを参照するための静的プロパティが含まれます。これらのプロパティは次のとおりです。

- File.applicationStorageDirectory - インストールされている各 AIR アプリケーションに固有の記憶領域ディレクトリ。このディレクトリは、動的なアプリケーションのアセットとユーザー環境設定の格納に適した場所です。データが大量の場合は、他の場所への格納を検討してください。
- File.applicationDirectory - アプリケーションがインストールされている（インストールされるアセットがある場合はそれらも含まれる）ディレクトリ。一部のオペレーティングシステムでは、物理ディレクトリではなく単一のパッケージファイルにアプリケーションが格納されます。この場合、ネイティブパスを使用してコンテンツにアクセスできない可能性があります。アプリケーションディレクトリは読み取り専用です。
- File.desktopDirectory - ユーザーのデスクトップディレクトリ。デスクトップディレクトリがプラットフォームで定義されていない場合は、ファイルシステム上の別の場所が使用されます。
- File.documentsDirectory - ユーザーのドキュメントディレクトリ。ドキュメントディレクトリがプラットフォームで定義されていない場合は、ファイルシステム上の別の場所が使用されます。
- File.userDirectory - ユーザーディレクトリ。ユーザーディレクトリがプラットフォームで定義されていない場合は、ファイルシステム上の別の場所が使用されます。

注意：デスクトップディレクトリ、ドキュメントディレクトリ、またはユーザーディレクトリの標準の場所がプラットフォームで定義されていない場合、File.documentsDirectory、File.desktopDirectory、および File.userDirectory は同じディレクトリを参照できます。

これらのプロパティには、オペレーティングシステムに応じて異なる値が設定されています。例えば、Mac と Windows で設定されているユーザーのデスクトップディレクトリのネイティブパスはそれぞれ異なります。ただし、File.desktopDirectory プロパティは、すべてのプラットフォーム上の適切なディレクトリパスを参照します。複数のプラットフォームにわたって正常に機能するアプリケーションを作成するには、アプリケーションが使用する他のディレクトリやファイルを参照する土台としてこれらのプロパティを使用します。次に、resolvePath() メソッドを使用してパスの詳細を設定します。例えば、次のコードは、アプリケーション記憶領域ディレクトリの preferences.xml ファイルを参照しています。

```
var prefsFile:File = air.File.applicationStorageDirectory;
prefsFile = prefsFile.resolvePath("preferences.xml");
```

File クラスでは特定のファイルパスを参照できますが、その場合、アプリケーションが複数のプラットフォームにわたって機能しなくなる可能性があります。例えば、C:¥Documents and Settings¥joe¥ というパスは、Windows 上でのみ機能します。このような理由から、File.documentsDirectory などの File クラスの静的プロパティを使用することをお勧めします。

一般的なディレクトリの場所

プラットフォーム	ディレクトリの種類	一般的なファイルシステムの場所
Linux	アプリケーション	/opt/filename/share
	アプリケーション記憶領域	/home/userName/.appdata/applicationID/Local Store
	デスクトップ	/home/userName/Desktop
	ドキュメント	/home/userName/Documents
	一時	/tmp/FlashTmp.randomString
	ユーザー	/home/userName
Mac	アプリケーション	/Applications/filename.app/Contents/Resources
	アプリケーション記憶領域	/Users/userName/Library/Preferences/applicationID/Local Store
	デスクトップ	/Users/userName/Desktop
	ドキュメント	/Users/userName/Documents
	一時	/private/var/folders/JY/randomString/TemporaryItems/FlashTmp
	ユーザー	/Users/userName
Windows	アプリケーション	C:¥Program Files¥filename
	アプリケーション記憶領域	C:¥Documents and settings¥userName¥ApplicationData¥applicationID¥Local Store
	デスクトップ	C:¥Documents and settings¥userName¥Desktop
	ドキュメント	C:¥Documents and settings¥userName¥My Documents
	一時	C:¥Documents and settings¥userName¥Local Settings¥Temp¥randomString.tmp
	ユーザー	C:¥Documents and settings¥userName

これらのディレクトリの実際のネイティブパスは、オペレーティングシステムとコンピューターの構成によって異なります。この表には、典型的なパスの例を示しています。プラットフォーム上でアプリケーションを正しく動作させるには、これらのディレクトリを参照する適切な File クラスの静的プロパティを使用する必要があります。実際の AIR アプリケーションでは、この表に示す applicationID と filename の値はアプリケーション記述子から取得されます。アプリケーション記述子に発行者 ID を指定すると、その発行者 ID がこれらのパスのアプリケーション ID に追加されます。userName の値は、インストールを行うユーザーのアカウント名です。

File オブジェクトでのディレクトリの参照

Adobe AIR 1.0 およびそれ以降

File オブジェクトでディレクトリの参照を設定するには、様々な方法があります。

ユーザーのホームディレクトリの参照

Adobe AIR 1.0 およびそれ以降

File オブジェクトでユーザーのホームディレクトリを参照できます。次のコードでは、ホームディレクトリの AIR Test サブディレクトリを参照するように File オブジェクトを設定しています。

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

ユーザーのドキュメントディレクトリの参照

Adobe AIR 1.0 およびそれ以降

File オブジェクトでユーザーのドキュメントディレクトリを参照できます。次のコードでは、ドキュメントディレクトリの AIR Test サブディレクトリを参照するように File オブジェクトを設定しています。

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

デスクトップディレクトリの参照

Adobe AIR 1.0 およびそれ以降

File オブジェクトでデスクトップを参照できます。次のコードでは、デスクトップの AIR Test サブディレクトリを参照するように File オブジェクトを設定しています。

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

アプリケーション記憶領域ディレクトリの参照

Adobe AIR 1.0 およびそれ以降

File オブジェクトでアプリケーション記憶領域ディレクトリを参照できます。AIR アプリケーションにはそれぞれ、アプリケーション記憶領域ディレクトリを定義する一意のパスが関連付けられています。このディレクトリは、アプリケーションおよびユーザーごとに一意に割り当てられます。このディレクトリは、ユーザーおよびアプリケーションに固有のデータ（ユーザーデータや環境設定ファイルなど）を格納するために使用できます。例えば、次のコードでは、アプリケーション記憶領域ディレクトリに格納されている prefs.xml という環境設定ファイルを File オブジェクトで参照します。

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

通常、アプリケーション記憶領域ディレクトリの場所は、ユーザー名とアプリケーション ID に基づきます。次に示すファイルシステムの場所は、アプリケーションをデバッグする場合に使用できます。このディレクトリ内のファイルを解決するには、File.applicationStorage プロパティまたは app-storage: URI スキームを常に使用する必要があります。

- Mac OS の場合

  /Users/**user name**/Library/Preferences/**applicationID**/Local Store/

  次に、例を示します。

  ```
  /Users/babbage/Library/Preferences/com.example.TestApp/Local Store
  ```

- Windows の場合（Documents and Settings ディレクトリ内）

  C:¥Documents and Settings\user name\Application Data**applicationID**¥Local Store¥

次に、例を示します。

```
C:\Documents and Settings\babbage\Application Data\com.example.TestApp\Local Store
```

- Linux の場合

  /home/**user name**/.appdata/**applicationID**/Local Store/

  次に、例を示します。

```
/home/babbage/.appdata/com.example.TestApp/Local Store
```

注意：アプリケーションに発行者 ID がある場合、発行者 ID はアプリケーション記憶領域ディレクトリへのパスとしても使用されます。

File.applicationStorageDirectory で作成された File オブジェクトの URL （および url プロパティ）では、次のように app-storage URL スキーム（151 ページの「サポートされる AIR URL スキーム」を参照）が使用されます。

```
var dir = air.File.applicationStorageDirectory;
dir = dir.resolvePath("prefs.xml");
air.trace(dir.url); // app-storage:/preferences
```

アプリケーションディレクトリの参照
Adobe AIR 1.0 およびそれ以降

File オブジェクトで、アプリケーションがインストールされたディレクトリ（アプリケーションディレクトリ）を参照できます。このディレクトリを参照するには、File.applicationDirectory プロパティを使用します。このディレクトリでは、アプリケーション記述ファイルなど、アプリケーションと共にインストールされたリソースを確認できます。例えば、次のコードでは、アプリケーションディレクトリにある **images** というディレクトリを File オブジェクトで参照します。

```
var dir = air.File.applicationDirectory;
dir = dir.resolvePath("images");
```

File.applicationDirectory で作成された File オブジェクトの URL （および url プロパティ）では、次のように app URL スキーム（151 ページの「サポートされる AIR URL スキーム」を参照）が使用されます。

```
var dir = air.File.applicationDirectory;
dir = dir.resolvePath("images");
air.trace(dir.url); // app:/images
```

ファイルシステムルートの参照
Adobe AIR 1.0 およびそれ以降

File.getRootDirectories() メソッドは、すべてのルートボリュームの一覧を出力します。Windows コンピューターの場合は、C: およびマウントされたボリュームが含まれます。Mac OS および Linux の場合は、常にマシンの一意のルートディレクトリ（「/」ディレクトリ）が返されます。StorageVolumeInfo.getStorageVolumes() メソッドは、マウントされたストレージボリュームに関する詳細を提供します（161 ページの「ストレージボリュームの操作」を参照）。

明示的なディレクトリの参照
Adobe AIR 1.0 およびそれ以降

File オブジェクトの nativePath プロパティを設定することにより、File オブジェクトで明示的なディレクトリを参照できます。Windows の場合の例を次に示します。

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

重要：このような方法で明示的なパスを参照すると、コードが複数のプラットフォームにわたって機能しなくなる可能性があります。例えば、前の例は Windows 上でのみ機能します。クロスプラットフォームで機能するディレクトリを見つけるには、File.applicationStorageDirectory などの File オブジェクトの静的プロパティを使用します。次に、resolvePath() メソッド（次の節を参照）を使用して、相対パスに移動します。

相対パスのナビゲーション
Adobe AIR 1.0 およびそれ以降

resolvePath() メソッドを使用すると、別の特定のパスを基準とした相対的なパスを取得できます。例えば、次のコードでは、ユーザーのホームディレクトリの AIR Test サブディレクトリを参照するように File オブジェクトを設定しています。

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

また、次のように、File オブジェクトの url プロパティを使用すると、URL ストリングに基づいてディレクトリを参照することもできます。

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

詳しくは、151 ページの「File パスの変更」を参照してください。

参照するディレクトリの選択
Adobe AIR 1.0 およびそれ以降

File クラスの browseForDirectory() メソッドを使用すると、ユーザーがオブジェクトに割り当てるディレクトリを選択できるシステムダイアログボックスを表示できます。browseForDirectory() メソッドは、非同期のメソッドです。File オブジェクトでは、ユーザーがディレクトリを選択して「開く」ボタンをクリックすると select イベントを送出し、ユーザーが「キャンセル」ボタンをクリックすると cancel イベントを送出します。

例えば、次のコードでは、ユーザーがディレクトリを選択できるようにし、その選択に応じたディレクトリパスを出力します。

```
var file = new air.File();
file.addEventListener(air.Event.SELECT, dirSelected);
file.browseForDirectory("Select a directory");
function dirSelected(event) {
    alert(file.nativePath);
}
```

アプリケーションを起動したディレクトリの参照
Adobe AIR 1.0 およびそれ以降

アプリケーションを起動したディレクトリの場所を取得するには、アプリケーションの起動時に送出された InvokeEvent オブジェクトの currentDirectory プロパティを確認します。詳しくは、291 ページの「コマンドライン引数のキャプチャ」を参照してください。

File オブジェクトでのファイルの参照
Adobe AIR 1.0 およびそれ以降

File オブジェクトで参照するファイルを設定するには、様々な方法があります。

明示的なファイルパスの参照
Adobe AIR 1.0 およびそれ以降

重要：明示的なパスを参照すると、コードが複数のプラットフォームにわたって機能しなくなる可能性があります。例えば、C:¥foo.txt というパスは、Windows 上でのみ機能します。クロスプラットフォームで機能するディレクトリを見つけるには、File.applicationStorageDirectory などの File オブジェクトの静的プロパティを使用します。次に、resolvePath() メソッド（151 ページの「File パスの変更」を参照）を使用して、相対パスに移動します。

次のように、File オブジェクトの url プロパティを使用すると、URL ストリングに基づいてファイルまたはディレクトリを参照できます。

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

URL は、次のように File() コンストラクター関数に渡すこともできます。

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

url プロパティでは、常に URI エンコード形式の URL が返されます（例えば、スペースは "%20 に置き換えられます）。

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

また、File オブジェクトの nativePath プロパティを使用して明示的なパスを設定することもできます。例えば、次のコードを Windows コンピューターで実行すると、C: ドライブの AIR Test サブディレクトリにある test.txt ファイルを参照するように File オブジェクトが設定されます。

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

このパスは、次のように File() コンストラクター関数に渡すこともできます。

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

nativePath プロパティのパス区切り文字としては、スラッシュ（/）を使用します。Windows では、円記号（¥）も使用できますが、その場合、アプリケーションが複数のプラットフォームにわたって機能しなくなる可能性があります。

詳しくは、151 ページの「File パスの変更」を参照してください。

ディレクトリ内のファイルの列挙
Adobe AIR 1.0 およびそれ以降

File オブジェクトの getDirectoryListing() メソッドを使用すると、ディレクトリのルートレベルにあるファイルおよびサブディレクトリを参照する File オブジェクトの配列を取得することができます。詳しくは、157 ページの「ディレクトリの列挙」を参照してください。

参照するファイルの選択
Adobe AIR 1.0 およびそれ以降

File クラスには、ユーザーがオブジェクトに割り当てるファイルを選択できるシステムダイアログボックスを表示する以下のメソッドがあります。

- browseForOpen()
- browseForSave()
- browseForOpenMultiple()

これらのメソッドでは、いずれも非同期で処理が行われます。browseForOpen() メソッドと browseForSave() メソッドでは、ユーザーがファイル（browseForSave() の場合はターゲットパス）を選択すると select イベントを送出します。browseForOpen() メソッドと browseForSave() メソッドのターゲットの File オブジェクトは、その選択に応じて、選択されたファイルを参照します。browseForOpenMultiple() メソッドでは、ユーザーがファイルを選択すると selectMultiple イベントを送出します。selectMultiple イベントは FileListEvent 型のイベントで、その files プロパティに（選択されたファイルを参照する）File オブジェクトの配列が格納されます。

例えば、次のコードでは、ファイルを選択するための「開く」ダイアログボックスが表示されます。

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

function selectTextFile(root)
{
    var txtFilter = new air.FileFilter("Text", "*.as;*.css;*.html;*.txt;*.xml");
    root.browseForOpen("Open", new window.runtime.Array(txtFilter));
    root.addEventListener(air.Event.SELECT, fileSelected);
}

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

アプリケーションで別の参照ダイアログボックスが開かれている場合に参照メソッドを呼び出すと、ランタイムから Error 例外がスローされます。

File パスの変更

Adobe AIR 1.0 およびそれ以降

resolvePath() メソッドを呼び出すか、オブジェクトの nativePath プロパティまたは url プロパティを変更することにより、既存の File オブジェクトのパスを変更することもできます。Windows の場合の例を次に示します。

```
file1 = air.File.documentsDirectory;
file1 = file1.resolvePath("AIR Test");
alert(file1.nativePath); // C:\Documents and Settings\userName\My Documents\AIR Test
var file2 = air.File.documentsDirectory;
file2 = file2.resolvePath("..");
alert(file2.nativePath); // C:\Documents and Settings\userName
var file3 = air.File.documentsDirectory;
file3.nativePath += "/subdirectory";
alert(file3.nativePath); // C:\Documents and Settings\userName\My Documents\subdirectory
var file4 = new air.File();
file4.url = "file:///c:/AIR Test/test.txt";
alert(file4.nativePath); // C:\AIR Test\test.txt
```

nativePath プロパティを使用する場合、ディレクトリの区切り文字としてスラッシュ（/）を使用します。Windows では、円記号（¥）も使用できますが、コードがクロスプラットフォームで機能しなくなるため、それは避ける必要があります。

サポートされる AIR URL スキーム

Adobe AIR 1.0 およびそれ以降

AIR における File オブジェクトの url プロパティの定義では、以下のいずれかの URL スキームを使用できます。

URL スキーム	説明
file	ファイルシステムのルートを基準とした相対的なパスを指定する場合に使用します。次に、例を示します。 `file:///c:/AIR Test/test.txt` file URL を指定する標準的な URL の形式は、file://<host>/<path> です。例外として、<host> のストリングは空にすることができます。その場合、「URL の解釈を行っているマシン」を示すものと解釈されます。そのため、file URL では、スラッシュが 3 つ連なる（///）ことがよくあります。
app	インストールされたアプリケーションのルートディレクトリ（インストールされたアプリケーションの application.xml ファイルが格納されたディレクトリ）を基準とした相対的なパスを指定する場合に使用します。例えば、次のパスは、インストールされているアプリケーションのディレクトリの images サブディレクトリを参照しています。 `app:/images`
app-storage	アプリケーション記憶領域ディレクトリを基準とした相対的なパスを指定する場合に使用します。AIR では、インストールされたアプリケーションごとに、一意のアプリケーション記憶領域ディレクトリが定義されます。このディレクトリは、そのアプリケーション固有のデータを格納するための便利な場所として使用できます。例えば、次のパスは、アプリケーション記憶領域ディレクトリの settings サブディレクトリにある prefs.xml ファイルを参照しています。 `app-storage:/settings/prefs.xml`

2 つのファイル間の相対パスの取得

Adobe AIR 1.0 およびそれ以降

getRelativePath() メソッドを使用すると、2 つのファイル間の相対パスを取得できます。

```
var file1 = air.File.documentsDirectory
file1 = file1.resolvePath("AIR Test");
var file2 = air.File.documentsDirectory
file2 = file2.resolvePath("AIR Test/bob/test.txt");

alert(file1.getRelativePath(file2)); // bob/test.txt
```

getRelativePath() メソッドの 2 番目のパラメーターである useDotDot パラメーターを使用すると、.. シンタックスを結果として返して、親ディレクトリを示すことができるようになります。

```
var file1 = air.File.documentsDirectory;
file1 = file1.resolvePath("AIR Test");
var file2 = air.File.documentsDirectory;
file2 = file2.resolvePath("AIR Test/bob/test.txt");
var file3 = air.File.documentsDirectory;
file3 = file3.resolvePath("AIR Test/susan/test.txt");

alert(file2.getRelativePath(file1, true)); // ../..
alert(file3.getRelativePath(file2, true)); // ../../bob/test.txt
```

標準化バージョンのファイル名の取得

Adobe AIR 1.0 およびそれ以降

Windows および Mac OS では、ファイル名およびパス名で大文字と小文字が区別されません。次のような場合、2 つの File オブジェクトで同じファイルが参照されます。

```
 File.documentsDirectory.resolvePath("test.txt");
File.documentsDirectory.resolvePath("TeSt.TxT");
```

ただし、実際のドキュメント名やディレクトリ名には大文字も使用されます。例えば、次の例では、ドキュメントディレクトリに AIR Test というフォルダーがあることを前提にしています。

```
var file = air.File.documentsDirectory;
file = file.resolvePath("AIR test");
trace(file.nativePath); // ... AIR test
file.canonicalize();
alert(file.nativePath); // ... AIR Test
```

canonicalize() メソッドでは、大文字と小文字が区別された正しいファイル名またはディレクトリ名を使用するように nativePath オブジェクトが変換されます。Linux などの大文字と小文字を区別するファイルシステムでは、大文字と小文字の区別だけが異なる複数のファイルが存在する場合、canonicalize() メソッドにより、最初に見つかったファイル（ファイルシステムによって決められた順序による）に合わせてパスが調整されます。

canonicalize() メソッドは、次のように、Windows で短いファイル名（「8.3」形式の名前）を長いファイル名に変換する場合にも使用できます。

```
var path = new air.File();
path.nativePath = "C:\\AIR~1";
path.canonicalize();
alert(path.nativePath); // C:\AIR Test
```

パッケージおよびシンボリックリンクの操作

Adobe AIR 1.0 およびそれ以降

各種のオペレーティングシステムで、パッケージファイルおよびシンボリックリンクファイルがサポートされています。

パッケージ - Mac OS では、ディレクトリをパッケージとして指定して、Mac OS Finder にディレクトリではなく単一のファイルとして表示することができます。

シンボリックリンク - Mac OS、Linux および Windows Vista では、シンボリックリンクがサポートされています。シンボリックリンクを使用すると、ファイルからディスク上の別のファイルやディレクトリを参照できるようになります。シンボリックリンクはエイリアスに似ていますが、同じではありません。エイリアスは常に（ディレクトリではなく）ファイルとして報告され、エイリアスまたはショートカットに対して読み取りまたは書き込みを行っても、その参照先である元のファイルまたはディレクトリが影響を受けることはありません。これに対し、シンボリックリンクは、参照先のファイルまたはディレクトリとまったく同じように動作します。ファイルまたはディレクトリとして報告することができ、シンボリックリンクに対して読み取りまたは書き込みを行うと、シンボリックリンク自体ではなく、参照先のファイルまたはディレクトリに反映されます。さらに、Windows では、ジャンクションポイント（NTFS ファイルシステムで使用される）を参照する File オブジェクトの isSymbolicLink プロパティが true に設定されています。

File クラスには、File オブジェクトがパッケージまたはシンボリックリンクを参照しているかどうかを確認するための isPackage プロパティおよび isSymbolicLink プロパティが用意されています。

次のコードでは、ユーザーのデスクトップディレクトリを繰り返し処理して、パッケージではないサブディレクトリの一覧を出力します。

```
var desktopNodes = air.File.desktopDirectory.getDirectoryListing();
for (i = 0; i < desktopNodes.length; i++)
{
    if (desktopNodes[i].isDirectory && !!desktopNodes[i].isPackage)
    {
        air.trace(desktopNodes[i].name);
    }
}
```

次のコードでは、ユーザーのデスクトップディレクトリを繰り返し処理して、シンボリックリンクではないファイルおよびディレクトリの一覧を出力します。

```
var desktopNodes = air.File.desktopDirectory.getDirectoryListing();
for (i = 0; i < desktopNodes.length; i++)
{
    if (!desktopNodes[i].isSymbolicLink)
    {
        air.trace(desktopNodes[i].name);
    }
}
```

canonicalize() メソッドでは、シンボリックリンクのパスを、そのリンクの参照先のファイルまたはディレクトリを参照するように変更します。次のコードでは、ユーザーのデスクトップディレクトリを繰り返し処理して、シンボリックリンクであるファイルによって参照されているパスを報告します。

```
var desktopNodes = air.File.desktopDirectory.getDirectoryListing();
for (i = 0; i < desktopNodes.length; i++)
{
    if (desktopNodes[i].isSymbolicLink)
    {
        var linkNode = desktopNodes[i];
        linkNode.canonicalize();
        air.trace(desktopNodes[i].name);
    }
}
```

ボリューム上の利用可能な領域の特定

Adobe AIR 1.0 およびそれ以降

File オブジェクトの spaceAvailable プロパティは、ファイルの場所として使用できる領域をバイト単位で示します。例えば、次のコードでは、アプリケーション記憶領域ディレクトリで利用可能な領域を確認します。

```
air.trace(air.File.applicationStorageDirectory.spaceAvailable);
```

File オブジェクトがディレクトリを参照している場合、spaceAvailable プロパティはファイルが使用できるディレクトリ内の領域を示します。File オブジェクトがファイルを参照している場合、spaceAvailable プロパティはファイルが増大可能な領域を示します。ファイルの場所が存在しない場合、spaceAvailable プロパティは 0 に設定されます。File オブジェクトがシンボリックリンクを参照している場合、spaceAvailable プロパティは、シンボリックリンクが指している場所で利用可能な領域に設定されます。

通常、ディレクトリやファイルに利用可能な領域は、ディレクトリまたはファイルを格納するボリュームで利用可能な領域と同じです。ただし、利用可能な領域では、クォータやディレクトリごとの制限が考慮される場合があります。

ファイルやディレクトリをボリュームに追加するには、一般的に、実際のファイルのサイズやディレクトリのコンテンツのサイズよりも大きな領域が必要です。例えば、オペレーティングシステムの場合、インデックス情報を格納するためにより多くの領域が必要になる可能性があります。また、必要なディスクセクタのために、追加の領域が使用される場合があります。さらに、利用可能な領域は動的に変化します。したがって、報告されたすべての領域をファイル記憶域として割り当てることはできません。ファイルシステムへの書き込みについては、162 ページの「ファイルの読み取りと書き込み」を参照してください。

StorageVolumeInfo.getStorageVolumes() メソッドは、マウントされたストレージボリュームに関する詳細を提供します（161 ページの「ストレージボリュームの操作」を参照）。

デフォルトのシステムアプリケーションでファイルを開く

Adobe AIR 2 以降

AIR 2 では、オペレーティングシステムによって登録されているアプリケーションを使用してファイルを開くことができます。例えば、AIR アプリケーションで DOC ファイルを開く場合は、DOC ファイルを開くように登録されているアプリケーションを使用します。File オブジェクトの openWithDefaultApplication() メソッドを使用すると、ファイルを開くことができます。例えば、次のコードでは、test.doc というファイルがユーザーのデスクトップに、DOC ファイル用のデフォルトのアプリケーションを使用して開かれます。

```
var file = air.File.deskopDirectory;
file = file.resolvePath("test.doc");
file.openWithDefaultApplication();
```

注意：Linux では、ファイル名拡張子でなく、ファイルの MIME タイプによって、ファイルのデフォルトのアプリケーションが判断されます。

次のコードでは、MP3 ファイルに移動して、MP3 ファイルを再生するためのデフォルトのアプリケーションで開くことができます。

```
var file = air.File.documentsDirectory;
var mp3Filter = new air.FileFilter("MP3 Files", "*.mp3");
file.browseForOpen("Open", [mp3Filter]);
file.addEventListener(Event.SELECT, fileSelected);
function fileSelected(event)
{
    file.openWithDefaultApplication();
}
```

openWithDefaultApplication() メソッドは、アプリケーションディレクトリに配置されているファイルでは使用できません。

AIR では、openWithDefaultApplication() メソッドを使用して開くことができないファイルがあります。Windows では、EXE や BAT など、一定の種類のファイルを開くことができません。Mac OS および Linux では、特定のアプリケーションを起動するファイルは開けません（Mac OS の Terminal と AppletLauncher、および Linux の csh、bash または ruby が含まれます）。openWithDefaultApplication() メソッドを使用して、これらのファイルを開こうとすると、例外が発生します。開くことができないファイルの種類について詳しくは、File.openWithDefaultApplication() メソッドのリファレンスガイドを参照してください。

注意：ネイティブインストーラーを使用してインストールされた AIR アプリケーション（拡張デスクトップアプリケーション）の場合、この制限はありません。

ファイルシステム情報の取得

Adobe AIR 1.0 およびそれ以降

File クラスには、ファイルシステムに関するいくつかの有益な情報を示す以下の静的プロパティがあります。

プロパティ	説明
File.lineEnding	ホストオペレーティングシステムで使用される行終了文字シーケンスです。Mac OS および Linux では、これは改行文字です。Windows では、復帰文字の後に改行文字が続いたものです。
File.separator	ホストオペレーティングシステムのパスコンポーネントの区切り文字です。Mac OS および Linux では、これはスラッシュ（/）です。Windows では、円記号（¥）です。
File.systemCharset	ホストオペレーティングシステムでファイルに対して使用されるデフォルトのエンコーディングです。これは、オペレーティングシステムでその言語に応じて使用される文字セットに関連します。

Capabilities クラスにも、ファイルの操作に役立つ有益なシステム情報があります。

プロパティ	説明
Capabilities.hasIME	システムに IME がインストールされているか（true）、インストールされていないか（false）を示します。
Capabilities.language	コンテンツが実行されているシステムの言語コードを示します。
Capabilities.os	現在のオペレーティングシステムを示します。

注意：Capabilities.os を使用してシステムの特性を確認する場合には、慎重に行う必要があります。システム特性を確認するためのより固有なプロパティが存在する場合は、それを使用してください。それ以外の場合、すべてのプラットフォームで正しく機能することがないコードを記述する危険があります。例えば、次のようなコードがあるとします。

```
var separator:String;
if (Capablities.os.indexOf("Mac") > -1)
{
    separator = "/";
}
else
{
    separator = "\\";
}
```

このコードは Linux 上で問題を引き起こします。単に File.separator プロパティを使用することをお勧めします。

ディレクトリの操作

Adobe AIR 1.0 およびそれ以降

ランタイムには、ローカルファイルシステム上のディレクトリを操作する機能が用意されています。

ディレクトリを参照する File オブジェクトの作成について詳しくは、147 ページの「File オブジェクトでのディレクトリの参照」を参照してください。

ディレクトリの作成

Adobe AIR 1.0 およびそれ以降

File.createDirectory() メソッドを使用して、ディレクトリを作成することができます。例えば、次のコードでは、ユーザーのホームディレクトリのサブディレクトリとして AIR Test というディレクトリを作成します。

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

既にそのディレクトリが存在する場合、createDirectory() メソッドでは何も行われません。

また、一部のモードでは、FileStream オブジェクトでファイルを開くときにもディレクトリが作成されます。FileStream インスタンスの作成時に FileStream() コンストラクターの fileMode パラメーターを FileMode.APPEND または FileMode.WRITE に設定すると、存在しない場合にディレクトリが作成されます。詳しくは、163 ページの「ファイルの読み取りと書き込みのワークフロー」を参照してください。

一時ディレクトリの作成

Adobe AIR 1.0 およびそれ以降

File クラスの createTempDirectory() メソッドを使用すると、次のように、システムの一時ディレクトリフォルダーにディレクトリを作成できます。

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

createTempDirectory() メソッドでは、一意の一時ディレクトリが自動的に作成されます（そのため、新しい一意の場所を指定する手間が省けます）。

一時ディレクトリは、アプリケーションのセッションに使用される一時ファイルを一時的に格納するために使用できます。システムの一時ディレクトリに新しい一意の一時ファイルを作成するための createTempFile() メソッドも用意されています。

一時ディレクトリはすべてのデバイスで自動的に削除されるわけではないので、アプリケーションを終了する前に削除する必要がないか確認してください。

ディレクトリの列挙

Adobe AIR 1.0 およびそれ以降

File オブジェクトの getDirectoryListing() メソッドまたは getDirectoryListingAsync() メソッドを使用すると、ディレクトリ内のファイルおよびサブフォルダーを参照する File オブジェクトの配列を取得することができます。

例えば、次のコードでは、ユーザーのドキュメントディレクトリの内容の一覧を出力します（サブディレクトリの内容は確認されません）。

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

このメソッドの非同期バージョンを使用した場合は、directoryListing イベントオブジェクトの files プロパティに、ディレクトリに関連する File オブジェクトの配列が格納されます。

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

function dirListHandler(event)
{
    var contents = event.files;
    for (i = 0; i < contents.length; i++)
    {
        alert(contents[i].name, contents[i].size);
    }
}
```

ディレクトリのコピーおよび移動

Adobe AIR 1.0 およびそれ以降

ファイルをコピーまたは移動する場合と同じメソッドを使用して、ディレクトリをコピーまたは移動することができます。例えば、次のコードでは、同期的にディレクトリがコピーされます。

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

copyTo() メソッドの overwrite パラメーターに true を指定すると、既存のターゲットディレクトリにあるすべてのファイルおよびフォルダーが削除され、ソースディレクトリ内のファイルおよびフォルダーに置き換えられます（ターゲットファイルがソースディレクトリに存在しない場合でも同様に処理されます）。

copyTo() メソッドの newLocation パラメーターとして指定するディレクトリは、結果として生成されるディレクトリへのパスを指定するものであり、そのディレクトリを格納する親ディレクトリを指定するものではありません。

詳しくは、159 ページの「ファイルのコピーおよび移動」を参照してください。

ディレクトリの内容の削除

Adobe AIR 1.0 およびそれ以降

File クラスの deleteDirectory() メソッドおよび deleteDirectoryAsync() メソッドを使用すると、ディレクトリを削除することができます。1 つ目のメソッドでは同期的に処理が行われ、2 つ目のメソッドでは非同期で処理が行われます（143 ページの「AIR ファイルの基礎」を参照）。どちらのメソッドにも deleteDirectoryContents パラメーター（Boolean 型）があり、メソッドの呼び出しでこのパラメーターが true に設定されている場合は空ではないディレクトリが削除され、false （デフォルト値）の場合は空のディレクトリだけが削除されます。

例えば、次のコードでは、ユーザーのドキュメントディレクトリの AIR Test サブディレクトリが同期的に削除されます。

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

次のコードでは、ユーザーのドキュメントディレクトリの AIR Test サブディレクトリが非同期で削除されます。

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

function completeHandler(event) {
    alert("Deleted.")
}
```

また、moveToTrash() メソッドおよび moveToTrashAsync() メソッドを使用すると、ディレクトリをシステムのごみ箱に移動することもできます。詳しくは、161 ページの「ごみ箱へのファイルの移動」を参照してください。

ファイルの操作

Adobe AIR 1.0 およびそれ以降

AIR のファイル API を使用すると、ファイル操作の基本的な機能をアプリケーションに追加することができます。例えば、ファイルの読み取りおよび書き込み、ファイルのコピーおよび削除などを追加できます。アプリケーションからローカルファイルシステムにアクセスできるようになるので、まだ参照していない場合は 68 ページの「AIR のセキュリティ」を参照してください。

注意：ファイルの種類を AIR アプリケーションに関連付けることができます（これによりダブルクリックでアプリケーションを開けるようになります）。詳しくは、297 ページの「ファイルの関連付けの管理」を参照してください。

ファイル情報の取得

Adobe AIR 1.0 およびそれ以降

File クラスには、File オブジェクトが参照するファイルまたはディレクトリに関する情報を示す以下のプロパティがあります。

File プロパティ	説明
creationDate	ローカルディスク上に存在するファイルの作成日です。
creator	使用されなくなりました。extension プロパティを使用してください（このプロパティでは、ファイルの Mac OS クリエータータイプが報告されます。これは、Mac OS X より前のバージョンの Mac OS でのみ使用されます）。
downloaded	（AIR 2 以降）参照先のファイルまたはディレクトリが、インターネット経由でダウンロード済みであるかどうかを示します。プロパティは、ファイルにダウンロード済みのフラグを付けられるオペレーティングシステム（以下参照）でのみ意味を持ちます。 • Windows XP Service Pack 2 以降および Windows Vista • Mac OS 10.5 以降
exists	参照先のファイルまたはディレクトリが存在するかどうかを表します。
extension	ファイル拡張子です。拡張子は、ファイル名の最後のドット（「.」）より後の部分です（ドットは含みません）。ファイル名にドットが含まれていない場合、extension は null になります。
icon	ファイルに対して定義されたアイコンを格納している Icon オブジェクトです。
isDirectory	File オブジェクトの参照先がディレクトリであるかどうかを表します。
modificationDate	ローカルディスク上に存在するファイルまたはディレクトリの最終変更日です。
name	ローカルディスク上に存在するファイルまたはディレクトリの名前です（ファイル拡張子も含みます（ファイル拡張子がある場合））。
nativePath	ホストオペレーティングシステムの表現形式に従った完全パスです。145 ページの「File オブジェクトのパス」を参照してください。
parent	File オブジェクトで表されるフォルダーまたはファイルが格納されているフォルダーです。このプロパティは、File オブジェクトがファイルシステムのルートにあるファイルまたはディレクトリを参照している場合は null になります。
size	ローカルディスク上に存在するファイルのサイズ（バイト単位）です。
type	使用されなくなりました。extension プロパティを使用してください（Mac OS では、このプロパティは 4 文字で表されるファイルの種類になります。これは、Mac OS X より前のバージョンの Mac OS でのみ使用されます）。
url	ファイルまたはディレクトリの URL です。145 ページの「File オブジェクトのパス」を参照してください。

これらのプロパティについて詳しくは、『HTML 開発者用 Adobe AIR API リファレンスガイド』の File クラスの項目を参照してください。

ファイルのコピーおよび移動

Adobe AIR 1.0 およびそれ以降

File クラスには、ファイルまたはディレクトリをコピーするためのメソッドとして、copyTo() と copyToAsync() という 2 つのメソッドがあります。また、ファイルまたはディレクトリを移動するためのメソッドとして、moveTo() と moveToAsync() という 2 つのメソッドがあります。copyTo() メソッドと moveTo() メソッドでは同期的に処理が行われ、copyToAsync() メソッドと moveToAsync() メソッドでは非同期で処理が行われます（143 ページの「AIR ファイルの基礎」を参照）。

ファイルをコピーまたは移動するには、2 つの File オブジェクトを設定します。1 つは、コピーまたは移動する対象のファイルを参照するオブジェクトで、このオブジェクトでコピーまたは移動メソッドを呼び出します。もう 1 つは、コピー先または移動先（結果）のパスを参照します。

次の例では、ユーザーのドキュメントディレクトリの AIR Test サブディレクトリにある test.txt ファイルを、copy.txt という名前で同じディレクトリにコピーします。

```
var original = air.File.documentsDirectory.resolvePath("AIR Test/test.txt");
var newFile = air.File.documentsDirectory.resolvePath("AIR Test/copy.txt");
original.copyTo(newFile, true);
```

この例では、copyTo() メソッドの overwrite パラメーター（2 番目のパラメーター）の値が true に設定されています。overwrite を true に設定することにより、既存のターゲットファイルが上書きされます。このパラメーターはオプションです。このパラメーターを false （デフォルト値）に設定した場合、ターゲットファイルが既に存在すると IOErrorEvent イベントが送出されます（ファイルはコピーされません）。

「非同期」バージョンのコピーメソッドおよび移動メソッドでは、非同期で処理が行われます。タスクの完了またはエラーの状態を監視するには、次のように addEventListener() を使用します。

```
var original = air.File.documentsDirectory;
original = original.resolvePath("AIR Test/test.txt");

var destination = air.File.documentsDirectory;
destination =  destination.resolvePath("AIR Test 2/copy.txt");

original.addEventListener(air.Event.COMPLETE, fileMoveCompleteHandler);
original.addEventListener(air.IOErrorEvent.IO_ERROR, fileMoveIOErrorEventHandler);
original.moveToAsync(destination);

function fileMoveCompleteHandler(event){
    alert(event.target); // [object File]
}
function fileMoveIOErrorEventHandler(event) {
    alert("I/O Error.");
}
```

File クラスには、ファイルまたはディレクトリをシステムのごみ箱に移動する File.moveToTrash() メソッドと File.moveToTrashAsync() メソッドもあります。

ファイルの削除

Adobe AIR 1.0 およびそれ以降

File クラスの deleteFile() メソッドおよび deleteFileAsync() メソッドを使用すると、ファイルを削除することができます。1 つ目のメソッドでは同期的に処理が行われ、2 つ目のメソッドでは非同期で処理が行われます（143 ページの「AIR ファイルの基礎」を参照）。

例えば、次のコードでは、ユーザーのドキュメントディレクトリにある test.txt ファイルが同期的に削除されます。

```
var file = air.File.documentsDirectory.resolvePath("test.txt");
file.deleteFile();
```

次のコードでは、ユーザーのドキュメントディレクトリの test.txt ファイルが非同期で削除されます。

```
var file = air.File.documentsDirectory.resolvePath("test.txt");
file.addEventListener(air.Event.COMPLETE, completeHandler)
file.deleteFileAsync();

function completeHandler(event) {
    alert("Deleted.")
}
```

また、moveToTrash() メソッドおよび moveToTrashAsync() メソッドを使用すると、ファイルまたはディレクトリをシステムのごみ箱に移動することもできます。詳しくは、161 ページの「ごみ箱へのファイルの移動」を参照してください。

ごみ箱へのファイルの移動

Adobe AIR 1.0 およびそれ以降

File クラスの moveToTrash() メソッドおよび moveToTrashAsync() メソッドを使用すると、ファイルまたはディレクトリをシステムのごみ箱に送ることができます。1 つ目のメソッドでは同期的に処理が行われ、2 つ目のメソッドでは非同期で処理が行われます（143 ページの「AIR ファイルの基礎」を参照）。

例えば、次のコードでは、ユーザーのドキュメントディレクトリにある test.txt ファイルがシステムのごみ箱に同期的に移動されます。

```
var file = air.File.documentsDirectory.resolvePath("test.txt");
file.moveToTrash();
```

注意：回復可能なごみ箱フォルダーの概念をサポートしていないオペレーティングシステムでは、ファイルがすぐに削除されます。

一時ファイルの作成

Adobe AIR 1.0 およびそれ以降

File クラスの createTempFile() メソッドを使用すると、次のように、システムの一時ディレクトリフォルダーにファイルを作成できます。

```
var temp = air.File.createTempFile();
```

createTempFile() メソッドでは、一意の一時ファイルが自動的に作成されます（そのため、新しい一意の場所を指定する手間が省けます）。

一時ファイルは、アプリケーションのセッションで使用される情報を一時的に格納するために使用できます。システムの一時ディレクトリに一意の一時ディレクトリを作成するための createTempDirectory() メソッドも用意されています。

一時ファイルはすべてのデバイスで自動的に削除されるわけではないので、アプリケーションを終了する前に削除する必要がないか確認してください。

ストレージボリュームの操作

Adobe AIR 2 以降

AIR 2 では、マスストレージボリュームのマウントまたはアンマウントを検出できます。StorageVolumeInfo クラスは、シングルトンオブジェクト storageVolumeInfo を定義します。StorageVolumeInfo.storageVolumeInfo オブジェクトは、ストレージボリュームがマウントされたときに、storageVolumeMount イベントを送出します。また、ボリュームがアンマウントされたときには storageVolumeUnmount イベントを送出します。StorageVolumeChangeEvent クラスでは、これらのイベントを定義します。

注意：最新の Linux ディストリビューションでは、StorageVolumeInfo オブジェクトは、特定の場所にマウントされた物理ドライブおよびネットワークドライブに対して storageVolumeMount および storageVolumeUnmount イベントのみを送出します。

StorageVolumeChangeEvent クラスの storageVolume プロパティは StorageVolume オブジェクトです。StorageVolume クラスは、次のようなストレージボリュームの基本的なプロパティを定義します。

- drive - Windows のボリ ームドライブ文字（その他のオペレーティングシステムでは null）
- fileSystemType — ストレージボリューム上のファイルシステムのタイプ（「FAT」、「NTFS」、「HFS」、「UFS」など）
- isRemoveable — ボリュームが削除可能（true）か不可能（false）か
- isWritable — ボリュームが書き込み可能（true）か不可能（false）か

- name — ボリュームの名前
- rootDirectory — ボリュームのルートディレクトリに対応する File オブジェクト

StorageVolumeChangeEvent クラスには rootDirectory プロパティも含まれています。rootDirectory プロパティは File オブジェクトで、マウントまたはアンマウントされたストレージボリュームのルートディレクトリを参照します。

StorageVolumeChangeEvent オブジェクトの storageVolume プロパティは、アンマウントされたボリュームに対しては定義されていません（null）。ただし、イベントの rootDirectory プロパティにアクセスすることはできます。

次のコードでは、ストレージボリュームがマウントされたときに、その名前とファイルパスを出力します。

```
air.StorageVolumeInfo.storageVolumeInfo.addEventListener(air.StorageVolumeChangeEvent.STORAGE_VOLUME_MOUNT, onVolumeMount);
function onVolumeMount(event)
{
    air.trace(event.storageVolume.name, event.rootDirectory.nativePath);
}
```

次のコードでは、ストレージボリュームがアンマウントされたときに、そのファイルパスを出力します。

```
air.StorageVolumeInfo.storageVolumeInfo.addEventListener(air.StorageVolumeChangeEvent.STORAGE_VOLUME_UNMOUNT, onVolumeUnmount);
function onVolumeUnmount(event)
{
    air.trace(event.rootDirectory.nativePath);
}
```

StorageVolumeInfo.storageVolumeInfo オブジェクトには getStorageVolumes() メソッドが含まれています。このメソッドは、現在マウントされているストレージボリュームに対応する StorageVolume オブジェクトのベクトルを返します。次のコードは、マウントされているすべてのストレージボリュームの名前とルートディレクトリのリストを表示する方法を示しています。

```
var volumes = air.StorageVolumeInfo.storageVolumeInfo.getStorageVolumes();
for (i = 0; i < volumes.length; i++)
{
    air.trace(volumes[i].name, volumes[i].rootDirectory.nativePath);
}
```

注意：最新の Linux ディストリビューションでは、getStorageVolumes() メソッドは、物理デバイスおよび特定の場所にマウントされたネットワークドライブに対応するオブジェクトを返します。

File.getRootDirectories() メソッドによって、ルートディレクトリがリストされます（148 ページの「ファイルシステムルートの参照」を参照）。ただし、StorageVolume オブジェクト（StorageVolumeInfo.getStorageVolumes() メソッドによって列挙）のほうが、ストレージボリュームに関する情報がより詳細に表示されます。

StorageVolume オブジェクトの rootDirectory プロパティの spaceAvailable プロパティを使用すると、ストレージボリューム上で利用可能な領域を確認できます（154 ページの「ボリューム上の利用可能な領域の特定」を参照）。

関連項目

StorageVolume

StorageVolumeInfo

ファイルの読み取りと書き込み

Adobe AIR 1.0 およびそれ以降

AIR アプリケーションでは、FileStream クラスを使用してファイルシステムの読み取りおよび書き込みを実行できます。

ファイルの読み取りと書き込みのワークフロー

Adobe AIR 1.0 およびそれ以降

ファイルの読み取りおよび書き込みのワークフローを次に示します。

パスを参照する File オブジェクトを初期化します。
File オブジェクトは、操作対象のファイル（または後で作成するファイル）のパスを表します。

```
var file = air.File.documentsDirectory;
file = file.resolvePath("AIR Test/testFile.txt");
```

この例では、File オブジェクトの File.documentsDirectory プロパティと resolvePath() メソッドを使用して File オブジェクトを初期化しています。ただし、File オブジェクトでファイルを参照する方法は、これ以外にも多数あります。詳しくは、149 ページの「File オブジェクトでのファイルの参照」を参照してください。

FileStream オブジェクトを初期化します。

FileStream オブジェクトの open() メソッドまたは openAsync() メソッドを呼び出します。
呼び出すメソッドは、ファイルを開いて同期操作と非同期操作のどちらを行うかによって異なります。open() メソッドの file パラメーターとして File オブジェクトを使用します。fileMode パラメーターに、ファイルの使用方法を示す FileMode クラスの定数を指定します。

例えば、次のコードでは、ファイルを作成して既存のデータを上書きするために使用する FileStream オブジェクトを初期化します。

```
var fileStream = new air.FileStream();
fileStream.open(file, air.FileMode.WRITE);
```

詳しくは、165 ページの「FileStream オブジェクトの初期化とファイルのオープンおよびクローズ」および 164 ページの「FileStream を開く際のモード」を参照してください。

ファイルを（openAsync() メソッドを使用して）非同期で開いた場合は、FileStream オブジェクトのイベントリスナーを追加して設定します。
これらのイベントリスナーメソッドは、様々な状況で FileStream オブジェクトから送出されたイベントに反応します。例えば、ファイルからのデータ読み取り時や I/O エラーの発生時、または書き込み対象のデータの書き込みが完了した時などの状況があります。

詳しくは、168 ページの「非同期プログラミングと非同期で開いた FileStream オブジェクトで生成されるイベント」を参照してください。

必要に応じて、データの読み取りおよび書き込みを行うためのコードを含めます。
FileStream クラスには、読み取りおよび書き込みに関連するメソッドが多数用意されています（それぞれ「read」または「write」で始まります）。データの読み取りおよび書き込みにどのメソッドを使用するかは、ターゲットファイルのデータの形式によって異なります。

例えば、ターゲットファイルのデータが UTF エンコード形式のテキストの場合は、readUTFBytes() メソッドおよび writeUTFBytes() メソッドを使用できます。データをバイト配列として処理する場合は、readByte()、readBytes()、writeByte() および writeBytes() の各メソッドを使用できます。詳しくは、169 ページの「データ形式と使用する読み取りおよび書き込みメソッドの選択」を参照してください。

ファイルを非同期で開いた場合は、読み取りメソッドを呼び出す前に、十分なデータを使用できることを確認します。詳しくは、167 ページの「読み取りバッファーと FileStream オブジェクトの bytesAvailable プロパティ」を参照してください。

ファイルに書き込む前に、利用可能なディスク領域の容量を確認する場合は、File オブジェクトの spaceAvailable プロパティを確認します。詳しくは、154 ページの「ボリューム上の利用可能な領域の特定」を参照してください。

ファイルの処理が完了したら、FileStream オブジェクトの close() メソッドを呼び出します。
close() メソッドを呼び出すと、他のアプリケーションでファイルを使用できるようになります。

詳しくは、165 ページの「FileStream オブジェクトの初期化とファイルのオープンおよびクローズ」を参照してください。

FileStream クラスを使用してファイルの読み取りおよび書き込みを行うサンプルアプリケーションについては、Adobe AIR デベロッパーセンターの次の記事を参照してください。

- テキストファイルエディターの構築
- テキストファイルエディターの構築
- テキストファイルエディターの構築
- XML 環境設定ファイルの読み取りと書き込み
- XML 環境設定ファイルの読み取りと書き込み

FileStream オブジェクトの操作

Adobe AIR 1.0 およびそれ以降

FileStream クラスでは、ファイルのオープン、読み取りおよび書き込みを行うためのメソッドが定義されています。

FileStream を開く際のモード

Adobe AIR 1.0 およびそれ以降

FileStream オブジェクトの open() メソッドおよび openAsync() メソッドには、それぞれ fileMode パラメーターがあります。このパラメーターでは、ファイルストリームについての次のようないくつかのプロパティを定義します。

- ファイルから読み取れるかどうか
- ファイルに書き込めるかどうか
- データを常にファイルの末尾の後に追加するかどうか（書き込み時）
- ファイルが存在しない場合（およびその親ディレクトリが存在しない場合）にどのように処理するか

各ファイルモードを以下に示します（open() メソッドおよび openAsync() メソッドの fileMode パラメーターとして指定できます）。

ファイルモード	説明
FileMode.READ	ファイルを読み取り専用で開くことを指定します。
FileMode.WRITE	ファイルを書き込み用に開くことを指定します。ファイルが存在しない場合は、FileStream オブジェクトを開いたときに作成されます。ファイルが存在する場合は、既存のデータは削除されます。
FileMode.APPEND	ファイルを追加用に開くことを指定します。ファイルが存在しない場合は作成されます。ファイルが存在する場合は、既存のデータは上書きされず、すべてファイルの末尾から書き込まれます。
FileMode.UPDATE	ファイルを読み書き用に開くことを指定します。ファイルが存在しない場合は作成されます。ランダムアクセスでファイルの読み書きを行う場合は、このモードを指定します。ファイルの任意の位置から読み取ることができます。ファイルに書き込む際は、書き込んだバイトのみが既存のデータに上書きされます（それ以外はどのバイトも変更されません）。

FileStream オブジェクトの初期化とファイルのオープンおよびクローズ
Adobe AIR 1.0 およびそれ以降

FileStream オブジェクトを開くとき、そのオブジェクトでファイルに対するデータの読み取りおよび書き込みができるように設定します。FileStream オブジェクトを開くには、File オブジェクトを FileStream オブジェクトの open() メソッドまたは openAsync() メソッドに渡します。

```
var myFile = air.File.documentsDirectory;
myFile = myFile.resolvePath("AIR Test/test.txt");
var myFileStream = new air.FileStream();
myFileStream.open(myFile, air.FileMode.READ);
```

fileMode パラメーター（open() メソッドおよび openAsync() メソッドの 2 番目のパラメーター）で、ファイルを開くモード（読み取り、書き込み、追加または更新）を指定します。詳しくは、前の節の 164 ページの「FileStream を開く際のモード」を参照してください。

openAsync() メソッドを使用して非同期のファイル操作用にファイルを開く場合は、非同期イベントを処理するイベントリスナーを設定します。

```
var myFile = air.File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream = new air.FileStream();
myFileStream.addEventListener(air.Event.COMPLETE, completeHandler);
myFileStream.addEventListener(air.ProgressEvent.PROGRESS, progressHandler);
myFileStream.addEventListener(air.IOErrorEvent.IOError, errorHandler);
myFileStream.open(myFile, air.FileMode.READ);

function completeHandler(event) {
    // ...
}

function progressHandler(event) {
    // ...
}

function errorHandler(event) {
    // ...
}
```

ファイルは、同期操作用または非同期操作用のいずれかで開かれます。どちらで開かれるかは、open() メソッドと openAsync() メソッドのどちらを使用したかによって決まります。詳しくは、143 ページの「AIR ファイルの基礎」を参照してください。

FileStream オブジェクトの open() メソッドで fileMode パラメーターを FileMode.READ または FileMode.UPDATE に設定した場合、FileStream オブジェクトを開くとすぐにデータが読み取りバッファーに読み取られます。詳しくは、167 ページの「読み取りバッファーと FileStream オブジェクトの bytesAvailable プロパティ」を参照してください。

関連付けられたファイルを閉じるには、FileStream オブジェクトの close() メソッドを呼び出します。これにより、そのファイルが他のアプリケーションで使用できるようになります。

FileStream オブジェクトの position プロパティ
Adobe AIR 1.0 およびそれ以降

FileStream オブジェクトの position プロパティでは、次の読み取りまたは書き込みメソッドでデータの読み取りまたは書き込みを行う位置を指定します。

読み取りまたは書き込み操作の前に、position プロパティにファイル内の任意の有効な位置を設定します。

例えば、次のコードでは、ファイル内の位置 8 にストリング「hello」を（UTF エンコード形式で）書き込みます。

```
var myFile = air.File.documentsDirectory;
myFile = myFile.resolvePath("AIR Test/test.txt");
var myFileStream = new air.FileStream();
myFileStream.open(myFile, air.FileMode.UPDATE);
myFileStream.position = 8;
myFileStream.writeUTFBytes("hello");
```

最初に FileStream オブジェクトを開いたときは、position プロパティは 0 に設定されています。

読み取り操作を行う場合はその操作の前に、position の値が 0 以上で、かつファイルのバイト数より小さい値（ファイル内に存在する位置）に設定されている必要があります。

position プロパティの値は、次の場合にのみ変更されます。

- position プロパティを明示的に設定したとき
- 読み取りメソッドを呼び出したとき
- 書き込みメソッドを呼び出したとき

FileStream オブジェクトの読み取りまたは書き込みメソッドを呼び出すと、すぐに position プロパティの値が、読み取りまたは書き込みを行ったバイト数だけインクリメントされます。使用する読み取りメソッドによって、position プロパティの値が、指定された読み取り対象のバイト数だけインクリメントされる場合と、使用できるバイト数だけインクリメントされる場合があります。その後に読み取りまたは書き込みメソッドを呼び出すと、その新しい位置から読み取りまたは書き込みが開始されます。

```
var myFile = air.File.documentsDirectory;
myFile = myFile.resolvePath("AIR Test/test.txt");
var myFileStream = new air.FileStream();
myFileStream.open(myFile, air.FileMode.UPDATE);
myFileStream.position = 4000;
alert(myFileStream.position);  // 4000
myFileStream.writeBytes(myByteArray, 0, 200);
alert(myFileStream.position); // 4200
```

ただし 1 つ例外があり、追加モードで開いた FileStream では、書き込みメソッドの呼び出し後に position プロパティは変更されません（追加モードでは、position プロパティの値に関係なく、常にファイルの末尾にデータが書き込まれます）。

非同期操作用に開いたファイルでは、次のコード行が実行されるまで書き込み操作は実行されません。ただし、複数の非同期メソッドを連続して呼び出すことは可能で、その場合はランタイムによって順番に実行されます。

```
var myFile = air.File.documentsDirectory;
myFile = myFile.resolvePath("AIR Test/test.txt");
var myFileStream = new air.FileStream();
myFileStream.openAsync(myFile, air.FileMode.WRITE);
myFileStream.writeUTFBytes("hello");
myFileStream.writeUTFBytes("world");
myFileStream.addEventListener(air.Event.CLOSE, closeHandler);
myFileStream.close();
air.trace("started.");

closeHandler(event)
{
    air.trace("finished.");
}
```

このコードのトレース出力は次のようになります。

```
started.
finished.
```

position の値は、読み取りまたは書き込みメソッドを呼び出した後すぐに（いつでも）指定できます。次の読み取りまたは書き込み操作は、その位置から開始されます。例えば、次のコードでは、writeBytes() メソッドを呼び出した直後に position プロパティを設定しています。書き込み操作の完了後も、position はその値（300）に設定されています。

```
var myFile = air.File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream = new air.FileStream();
myFileStream.openAsync(myFile, air.FileMode.UPDATE);
myFileStream.position = 4000;
air.trace(myFileStream.position); // 4000
myFileStream.writeBytes(myByteArray, 0, 200);
myFileStream.position = 300;
air.trace(myFileStream.position); // 300
```

読み取りバッファーと FileStream オブジェクトの bytesAvailable プロパティ
Adobe AIR 1.0 およびそれ以降

読み取り機能を指定して FileStream オブジェクトを開いた場合（open() メソッドまたは openAsync() メソッドの fileMode パラメーターを READ または UPDATE に設定した場合）、ランタイムでデータが内部バッファーに格納されます。FileStream オブジェクトでのバッファーへのデータの読み取りは、（FileStream オブジェクトの open() メソッドまたは openAsync() メソッドを呼び出して）ファイルを開くとすぐに開始されます。

同期操作用に（open() メソッドを使用して）開いたファイルでは、いつでも position プロパティに任意の有効な位置（ファイルの範囲内）を指定して、任意の量のデータ（ファイルの範囲内）の読み取りを開始できます。次にコードの例を示します（ファイルに 100 バイト以上あることを前提にしています）。

```
var myFile = air.File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream = new air.FileStream();
myFileStream.open(myFile, air.FileMode.READ);
myFileStream.position = 10;
myFileStream.readBytes(myByteArray, 0, 20);
myFileStream.position = 89;
myFileStream.readBytes(myByteArray, 0, 10);
```

ファイルを同期操作用と非同期操作用のどちらで開いたかに関係なく、読み取りメソッドでは常に、bytesAvalable プロパティで表される「使用可能な」バイトから読み取りを行います。同期的に読み取る場合は、常にファイルのすべてのバイトが使用可能です。非同期で読み取る場合は、非同期バッファーへの入力が progress イベントによって通知されるにつれて、position プロパティで指定された位置以降のバイトが使用可能になります。

同期操作用に開いたファイルでは、bytesAvailable プロパティは、常に position プロパティからファイルの末尾までのバイト数を表すように設定されます（常にファイル内のすべてのバイトを読み取ることができます）。

非同期操作用に開いたファイルでは、読み取りメソッドを呼び出す前に、読み取りバッファーに十分なデータが格納されていることを確認する必要があります。非同期で開いたファイルでは、読み取り操作が進むにつれ、読み取り操作の開始時に position で指定された位置以降のデータがファイルからバッファーに追加され、バイトが読み取られるたびに bytesAvailable プロパティの値がインクリメントされます。bytesAvailable プロパティは、position プロパティで指定された位置にあるバイトからバッファーの末尾までの使用可能なバイト数を示します。定期的に、FileStream オブジェクトから progress イベントが送信されます。

非同期で開いたファイルでは、読み取りバッファーでデータが使用可能になると、FileStream オブジェクトから定期的に progress イベントが送出されます。例えば、次のコードでは、データがバッファーに読み取られると、そのデータを ByteArray オブジェクトの bytes に読み取ります。

```
var bytes = new air.ByteArray();
var myFile = new air.File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream = new air.FileStream();
myFileStream.addEventListener(air.ProgressEvent.PROGRESS, progressHandler);
myFileStream.openAsync(myFile, air.FileMode.READ);

function progressHandler(event)
{
    myFileStream.readBytes(bytes, myFileStream.position, myFileStream.bytesAvailable);
}
```

非同期で開かれたファイルでは、読み取りバッファー内のデータのみを読み取ることができます。また、データを読み取ると、読み取りバッファーからそのデータは削除されます。読み取り操作を行う場合は、読み取り操作を呼び出す前に、読み取りバッファーにデータが存在することを確認する必要があります。例えば、次のコードでは、ファイル内の位置 4000 から始まる 8000 バイトのデータを読み取ります。

```
var myFile = air.File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream = new air.FileStream();
myFileStream.addEventListener(air.ProgressEvent.PROGRESS, progressHandler);
myFileStream.addEventListener(air.Event.COMPLETE, completed);
myFileStream.openAsync(myFile, air.FileMode.READ);
myFileStream.position = 4000;

var str = "";

function progressHandler(event)
{
    if (myFileStream.bytesAvailable > 8000 )
    {
        str += myFileStream.readMultiByte(8000, "iso-8859-1");
    }
}
```

書き込み操作中は、FileStream オブジェクトでは読み取りバッファーにデータを読み取りません。書き込み操作が完了すると（書き込みバッファー内のすべてのデータがファイルに書き込まれると）、FileStream オブジェクトでは新しい読み取りバッファーを開始し（関連付けられている FileStream オブジェクトが読み取り機能を指定して開かれている場合）、position プロパティで指定された位置から、読み取りバッファーへのデータの読み取りを開始します。position プロパティは、書き込まれた最後のバイトの位置にするか、別の位置を指定することもできます。別の位置を指定するには、書き込み操作後に position プロパティに別の値を指定します。

非同期プログラミングと非同期で開いた FileStream オブジェクトで生成されるイベント
Adobe AIR 1.0 およびそれ以降

ファイルを（openAsync() メソッドを使用して）非同期で開いた場合、ファイルの読み取りおよび書き込みは非同期で実行されます。読み取りバッファーへのデータの読み取り中や出力データの書き込み中に、他の ActionScript コードを実行することができます。

そのため、非同期で開いた FileStream オブジェクトによって生成されるイベントに登録する必要があります。

次のように、progress イベントに登録すると、新しいデータが読み取り可能になったときに通知を受けることができます。

```
var myFile = air.File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream = new air.FileStream();
myFileStream.addEventListener(air.ProgressEvent.PROGRESS, progressHandler);
myFileStream.openAsync(myFile, air.FileMode.READ);
var str = "";

function progressHandler(event)
{
    str += myFileStream.readMultiByte(myFileStream.bytesAvailable, "iso-8859-1");
}
```

次のように、complete イベントに登録すると、データ全体を読み取ることができます。

```
var myFile = air.File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream = new air.FileStream();
myFileStream.addEventListener(air.Event.COMPLETE, completed);
myFileStream.openAsync(myFile, air.FileMode.READ);
var str = "";
function completeHandler(event)
{
    str = myFileStream.readMultiByte(myFileStream.bytesAvailable, "iso-8859-1");
}
```

入力データをバッファーに格納して非同期読み取りを有効にする場合とほぼ同じようにして、非同期ストリームに書き込んだデータは、バッファーに格納されて非同期でファイルに書き込まれます。データがファイルに書き込まれると、FileStream オブジェクトから定期的に OutputProgressEvent オブジェクトが送出されます。OutputProgressEvent オブジェクトには、書き込まれる残りのバイト数が設定された bytesPending プロパティがあります。outputProgress イベントに登録すると、このバッファーが実際にファイルに書き込まれたときに通知を受けることができます（進捗状況ダイアログを表示する場合などに便利です）。ただし、一般には、このようにする必要はありません。具体的に言うと、書き込まれていないバイトについては考慮せずに、close() メソッドを呼び出すことができます。FileStream オブジェクトでは引き続きデータの書き込みが行われ、最後のバイトがファイルに書き込まれて書き込み元のファイルが閉じられた後に close イベントが送出されます。

データ形式と使用する読み取りおよび書き込みメソッドの選択
Adobe AIR 1.0 およびそれ以降

すべてのファイルは、ディスク上のバイトのセットからなります。ActionScript では、ファイルのデータを常に ByteArray として表すことができます。例えば、次のコードでは、ファイルのデータを bytes という ByteArray オブジェクトに読み取ります。

```
var myFile = air.File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream = new air.FileStream();
myFileStream.addEventListener(air.Event.COMPLETE, completeHandler);
myFileStream.openAsync(myFile, air.FileMode.READ);
var bytes = new air.ByteArray();

function completeHandler(event)
{
    myFileStream.readBytes(bytes, 0, myFileStream.bytesAvailable);
}
```

同様に、次のコードでは、bytes という ByteArray のデータをファイルに書き込みます。

```
var myFile = air.File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream = new air.FileStream();
myFileStream.open(myFile, air.FileMode.WRITE);
myFileStream.writeBytes(bytes, 0, bytes.length);
```

ただし、ActionScript の ByteArray オブジェクトにデータを格納したくない場合もよくあります。また、多くの場合、データファイルのファイル形式は決められています。

例えば、ファイルのデータがテキストファイル形式であり、そのデータを String オブジェクトで表したい場合があります。

そのため、FileStream クラスには、ByteArray オブジェクト以外の種類のデータを読み書きするための読み取りおよび書き込みメソッドが用意されています。例えば、次のコードのように、readMultiByte() メソッドを使用すると、ファイルからデータを読み取って String オブジェクトに格納できます。

```
var myFile = air.File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream = new air.FileStream();
myFileStream.addEventListener(air.Event.COMPLETE, completed);
myFileStream.openAsync(myFile, air.FileMode.READ);
var str = "";

function completeHandler(event)
{
    str = myFileStream.readMultiByte(myFileStream.bytesAvailable, "iso-8859-1");
}
```

readMultiByte() メソッドの 2 番目のパラメーターでは、ActionScript でデータの解釈に使用するテキスト形式（この例では「iso-8859-1」）を指定します。Adobe AIR では、一般的な文字セットのエンコードがサポートされています（「サポートされている文字セット」を参照）。

FileStream クラスには、読み取りバッファーのデータを、UTF-8 文字セットを使用してストリングに読み取る readUTFBytes() メソッドも用意されています。UTF-8 文字セットの文字は可変長なので、progress イベントに応答するメソッドでは readUTFBytes() は使用しないでください。読み取りバッファーの末尾にあるデータが不完全な文字を表す可能性があります（これは、readMultiByte() メソッドを可変長の文字エンコード形式で使用した場合にも当てはまります）。そのため、FileStream オブジェクトから complete イベントが送出されたら、データセット全体を読み取ります。

同様に、String オブジェクトおよびテキストファイルを処理するための書き込みメソッドとして、writeMultiByte() メソッドおよび writeUTFBytes() メソッドが用意されています。

readUTF() メソッドおよび writeUTF() メソッド（readUTFBytes() および writeUTFBytes() と混同しないようにしてください）でもファイルのテキストデータが読み書きされますが、これらのメソッドでは、テキストデータの前にそのテキストデータの長さを指定するデータがあることが前提になります。このような指定は、標準的なテキストファイルでは一般的ではありません。

一部の UTF エンコード形式のテキストファイルは、エンコード形式（UTF-16 や UTF-32 など）と同様にエンディアン形式を定義する「UTF-BOM」（バイト順序マーク）で始まります。

テキストファイルの読み取りおよび書き込みの例については、171 ページの「例：XML オブジェクトへの XML ファイルの読み取り」を参照してください。

複雑な ActionScript オブジェクトのデータを格納および取得する場合は、readObject() および writeObject() が適しています。データは ActionScript Message Format （AMF）でエンコードされます。Adobe AIR、Flash Player、Flash Media Server、Flex Data Services には、この形式のデータを操作するための API が用意されています。

他にも、読み取りおよび書き込みメソッド（readDouble() と writeDouble() など）がいくつか用意されています。ただし、これらを使用する場合は、これらのメソッドで定義されるデータの形式にファイル形式が適合することを確認してください。

多くの場合、ファイル形式は単純なテキスト形式よりも複雑です。例えば、MP3 ファイルには、MP3 ファイル固有の復元およびデコードアルゴリズムでしか解釈できない圧縮データが含まれています。また、MP3 ファイルには、ファイルに関するメタタグ情報（曲のタイトルやアーティストなど）を格納する ID3 タグが含まれる場合もあります。ID3 形式には複数のバージョンがありますが、最も単純なバージョン（ID3 バージョン 1）について 173 ページの「例：ランダムアクセスによるデータの読み取りと書き込み」の節で説明します。

他のファイル形式（イメージ、データベース、アプリケーションドキュメントなどで使用される形式）でもそれぞれ構造が異なるため、それらのデータを ActionScript で処理するには、データの構造を理解する必要があります。

load() メソッドおよび save() メソッドの使用

Flash Player 10 以降、Adobe AIR 1.5 以降

Flash Player 10 では、FileReference クラスに load() メソッドおよび save() メソッドが追加されています。これらのメソッドは AIR 1.5 にもあり、File クラスがこれらのメソッドを FileReference クラスから継承しています。これらのメソッドは、Flash Player でユーザーが安全にファイルデータの読み込みや保存を行うことができるように設計されています。しかし、AIR アプリケーションでは、ファイルの非同期での読み込みや保存を簡単に実行するための手段として、これらのメソッドを使用することもできます。

例えば、次のコードは、文字列をテキストファイルに保存します。

```
var file = air.File.applicationStorageDirectory.resolvePath("test.txt");
var str = "Hello.";
file.addEventListener(air.Event.COMPLETE, fileSaved);
file.save(str);
function fileSaved(event)
{
    air.trace("Done.");
}
```

data パラメーター（save() メソッド）は、String または ByteArray の値を指定できます。この引数が String 値の場合、UTF-8 エンコード形式のテキストファイルとしてファイルが保存されます。

このコード例を実行すると、ファイルの保存先を選択するためのダイアログボックスが表示されます。

次のコードは、UTF-8 エンコード形式のテキストファイルから文字列を読み込みます。

```
var file = air.File.applicationStorageDirectory.resolvePath("test.txt");
file.addEventListener(air.Event.COMPLETE, loaded);
file.load();
var str;
function loaded(event)
{
    var bytes = file.data;
    str = bytes.readUTFBytes(bytes.length);
    air.trace(str);
}
```

FileStream クラスを使用すると、load() メソッドおよび save() メソッドよりも多くの機能を使用できます。次に例を示します。

- FileStream クラスを使用すると、同期および非同期の両方でデータを読み書きできます。
- FileStream クラスを使用すると、ファイルに対してインクリメンタルな書き込みを行うことができます。
- FileStream クラスを使用すると、ファイルを開いてランダムアクセス（ファイルの任意の場所の読み書き）を実行できます。
- FileStream クラスを使用すると、fileMode パラメーター（open() メソッドまたは openAsync() メソッド）を設定することにより、ファイルに対するファイルアクセスの種類を指定できます。
- FileStream クラスを使用すると、ファイルを開くダイアログボックスや名前を付けて保存ダイアログボックスを表示せずに、データをファイルに保存できます。
- FileStream クラスを使用してデータを読み込む場合は、バイト配列以外の型を直接使用できます。

例：XML オブジェクトへの XML ファイルの読み取り

Adobe AIR 1.0 およびそれ以降

以下の例では、XML データが格納されているテキストファイルの読み取りおよび書き込みを行う方法を示します。

ファイルからの読み取りを行うには、File オブジェクトと FileStream オブジェクトを初期化し、FileStream の readUTFBytes() メソッドを呼び出して、ストリングを XML オブジェクトに変換します。

```
var file = air.File.documentsDirectory.resolvePath("AIR Test/preferences.xml");
var fileStream = new air.FileStream();
fileStream.open(file, air.FileMode.READ);
var prefsXML = fileStream.readUTFBytes(fileStream.bytesAvailable);
fileStream.close();
```

同様に、ファイルへのデータの書き込みも、適切な File オブジェクトと FileStream オブジェクトを設定してから FileStream オブジェクトの書き込みメソッドを呼び出すだけで簡単にできます。書き込みメソッドには、次のように String 型の XML データを渡します。

```
var file = air.File.documentsDirectory.resolvePath("AIR Test/preferences.xml");
fileStream = new air.FileStream();
fileStream.open(file, air.FileMode.WRITE);

var outputString = '<?xml version="1.0" encoding="utf-8"?>\n';
outputString += '<prefs><autoSave>true</autoSave></prefs>'

fileStream.writeUTFBytes(outputString);
fileStream.close();
```

これらの例では、ファイルが UTF-8 形式であるという前提で、readUTFBytes() メソッドと writeUTFBytes() メソッドを使用しています。形式が異なる場合は、別のメソッドを使用する必要がある場合があります（169 ページの「データ形式と使用する読み取りおよび書き込みメソッドの選択」を参照）。

上記の例では、同期操作用に開いた FileStream オブジェクトを使用しています。ファイルは非同期操作用に開くこともできます（イベントに応答するイベントリスナー関数が必要になります）。例えば、次のコードは、XML ファイルを非同期で読み取る方法を示しています。

```
var file = air.File.documentsDirectory.resolvePath("AIR Test/preferences.xml");
var fileStream= new air.FileStream();
fileStream.addEventListener(air.Event.COMPLETE, processXMLData);
fileStream.openAsync(file, air.FileMode.READ);
var prefsXML;

function processXMLData(event)
{
    var xmlString = fileStream.readUTFBytes(fileStream.bytesAvailable);
    prefsXML = domParser.parseFromString(xmlString, "text/xml");
    fileStream.close();
}
```

ファイル全体が読み取りバッファーに読み取られると（FileStream オブジェクトから complete イベントが送出されると）、processXMLData() メソッドが呼び出されます。これにより、readUTFBytes() メソッドが呼び出されて String 型の読み取りデータが取得され、そのストリングに基づいて XML オブジェクトである prefsXML が作成されます。

これらの機能を示すサンプルアプリケーションについては、「XML 環境設定ファイルの読み取りおよび書き込み」を参照してください。

これらの機能を示すサンプルアプリケーションについては、「XML 環境設定ファイルの読み取りおよび書き込み」を参照してください。

例：ランダムアクセスによるデータの読み取りと書き込み

Adobe AIR 1.0 およびそれ以降

MP3 ファイルには、ID3 タグが含まれている場合があります。これは、記録した内容を識別するメタデータが格納された、ファイルの先頭または末尾にあるセクションです。ID3 タグ形式自体、様々なリビジョンがあります。この例では、最も単純な ID3 形式（ID3 バージョン 1.0）を含む MP3 ファイルを、「ファイルデータのランダムアクセス」を使用して読み書きする（つまり、ファイル内の任意の位置で読み書きを行う）方法を示します。

ID3 バージョン 1 のタグを含む MP3 ファイルでは、ファイルの末尾（最後の 128 バイト）に ID3 データが格納されています。

ランダムアクセスで読み書きを行うためにファイルにアクセスする場合、open() メソッドまたは openAsync() メソッドの fileMode パラメーターとして FileMode.UPDATE を指定することが重要です。

```
var file = air.File.documentsDirectory.resolvePath("My Music/Sample ID3 v1.mp3");
var fileStr = new air.FileStream();
fileStr.open(file, air.FileMode.UPDATE);
```

これにより、ファイルに対する読み取りと書き込みの両方が可能になります。

ファイルを開いたら、次のようにして position ポインターをファイルの末尾から 128 バイトの位置に設定できます。

```
fileStr.position = file.size - 128;
```

このコードで position プロパティをファイル内のこの位置に設定する理由は、ID3 v1.0 形式では ID3 タグデータをファイルの末尾 128 バイトに格納するように決められているからです。仕様ではさらに、次のように決められています。

- タグの最初の 3 バイトには、ストリング「TAG」を記述する。
- 次の 30 文字には、MP3 トラックのタイトルをストリングで記述する。
- 次の 30 文字には、アーティストの名前をストリングで記述する。
- 次の 30 文字には、アルバムの名前をストリングで記述する。
- 次の 4 文字には、年をストリングで記述する。
- 次の 30 文字には、コメントをストリングで記述する。
- 次のバイトには、トラックのジャンルを示すコードを記述する。
- テキストデータはすべて ISO 8859-1 形式にする。

id3TagRead() メソッドで、読み取り後（complete イベント後）にデータを確認します。

```
function id3TagRead()
{
    if (fileStr.readMultiByte(3, "iso-8859-1").match(/tag/i))
    {
        var id3Title = fileStr.readMultiByte(30, "iso-8859-1");
        var id3Artist = fileStr.readMultiByte(30, "iso-8859-1");
        var id3Album = fileStr.readMultiByte(30, "iso-8859-1");
        var id3Year = fileStr.readMultiByte(4, "iso-8859-1");
        var id3Comment = fileStr.readMultiByte(30, "iso-8859-1");
        var id3GenreCode =  fileStr.readByte().toString(10);
    }
}
```

ファイルへの書き込みもランダムアクセスで実行できます。例えば、id3Title 変数を解析して（String クラスのメソッドで）大文字小文字が正しいかを確認した後で、次のように、変更後のストリング（newTitle）をファイルに書き込むことができます。

```
fileStr.position = file.length - 125;    // 128 - 3
fileStr.writeMultiByte(newTitle, "iso-8859-1");
```

ID3 バージョン 1 の規格に準拠するには、newTitle のストリングの長さが 30 文字になるように、末尾を文字コード 0（String.fromCharCode(0)）で埋める必要があります。

第 12 章：AIR でのドラッグ&ドロップ

Adobe AIR 1.0 およびそれ以降

ユーザーインターフェイスのドラッグ&ドロップジェスチャをサポートするには、Adobe® AIR™ ドラッグ&ドロップ API のクラスを使用します。ここで言うジェスチャとは、ユーザーが情報をコピー、移動、またはリンクするために、オペレーションシステムとアプリケーションの両方を介して行う操作のことです。ドラッグアウトジェスチャは、ユーザーがコンポーネントまたはアプリケーションの内部からその外部へオブジェクトをドラッグしたときに発生します。ドラッグインジェスチャは、ユーザーがコンポーネントまたはアプリケーションの外部からその内部へオブジェクトをドラッグしたときに発生します。

ドラッグ&ドロップ API を使用すると、ユーザーがアプリケーション間およびアプリケーション内のコンポーネント間でデータをドラッグできるようにすることができます。サポートされている移動形式は次のとおりです。

- ビットマップ
- ファイル
- HTML 形式のテキスト
- テキスト
- URL

HTML でのドラッグ&ドロップ

Adobe AIR 1.0 およびそれ以降

HTML ベースのアプリケーション（または HTMLLoader に表示される HTML）の内部または外部にデータをドラッグするには、HTML ドラッグ&ドロップイベントを使用します。HTML ドラッグ&ドロップ API を使用すると、HTML コンテンツ内の DOM エレメントに、または DOM エレメントからドラッグできます。

注意：また、HTML コンテンツを含む HTMLLoader オブジェクトのイベントをリッスンすることで、AIR NativeDragEvent および NativeDragManager API を使用することもできます。ただし、HTML DOM とより密接に統合されているのは HTML API です。HTML API を使用すると、デフォルト動作を制御できます。NativeDragEvent および NativeDragManager API は HTML ベースのアプリケーションでは一般的に使用しないので、「HTML 開発者用 Adobe AIR API リファレンスガイド」には記載されていません。これらのクラスの使用について詳しくは、「Adobe ActionScript 3.0 開発ガイド」および「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。

デフォルトのドラッグ&ドロップ動作

Adobe AIR 1.0 およびそれ以降

HTML 環境では、テキスト、イメージ、および URL に関連するドラッグ&ドロップ操作のデフォルト動作を使用できます。デフォルト動作を使用する場合、常にこれらの種類のデータをエレメントからドラッグできます。ただし、エレメントにドラッグできるのはテキストだけです。また、ページの編集可能な領域内のエレメントにしかドラッグできません。ページの編集可能な領域間または領域内でテキストをドラッグすると、デフォルト動作によって移動アクションが実行されます。編集不能な領域またはアプリケーションの外部から編集可能な領域にテキストをドラッグすると、デフォルト動作によってコピーアクションが実行されます。

デフォルト動作を上書きするには、ドラッグ & ドロップイベントを独自に処理します。デフォルト動作をキャンセルするには、ドラッグ & ドロップイベントに応じて送出されたオブジェクトの preventDefault() メソッドを呼び出す必要があります。その後、必要に応じてドロップターゲットへのデータの挿入とドラッグ元からのデータの削除を行ってから、選択したアクションを実行することができます。

デフォルトでは、ユーザーは任意のテキストを選択してドラッグしたり、イメージおよびリンクをドラッグしたりできます。WebKit CSS プロパティである -webkit-user-select を使用すると、HTML エレメントの選択方法を制御できます。例えば、-webkit-user-select を none に設定すると、エレメントのコンテンツを選択できなくなります。そのため、ドラッグすることもできません。また、-webkit-user-drag CSS プロパティを使用すると、エレメントをまとめてドラッグできるかどうかを制御できます。ただし、エレメントのコンテンツは個別に扱われます。この場合でも、ユーザーは選択したテキスト部分をドラッグできます。詳しくは、16 ページの「AIR での CSS」を参照してください。

HTML のドラッグ & ドロップイベント

Adobe AIR 1.0 およびそれ以降

ドラッグの開始元であるイニシエータエレメントによって送出されるイベントは次のとおりです。

イベント	説明
dragstart	ユーザーがドラッグジェスチャを開始したときに送出されます。このイベントのハンドラーでは、イベントオブジェクトの preventDefault() メソッドを呼び出すことで、必要に応じてドラッグを防止できます。ドラッグしたデータをコピー、リンク、または移動できるかどうかを制御するには、effectAllowed プロパティを設定します。選択されたテキスト、イメージ、およびリンクは、デフォルト動作によってクリップボードに格納されます。ただし、イベントオブジェクトの dataTransfer プロパティを使用すると、異なるデータをドラッグジェスチャに設定することができます。
drag	ドラッグジェスチャの実行中に継続的に送出されます。
dragend	ユーザーがマウスボタンを放してドラッグジェスチャを終了したときに送出されます。

ドラッグターゲットによって送出されるイベントは次のとおりです。

イベント	説明
dragover	ドラッグジェスチャがエレメントの境界内にあるときに継続的に送出されます。このイベントのハンドラーでは、dataTransfer.dropEffect プロパティを設定して、ユーザーがマウスボタンを放してドロップしたときに実行されるアクション（コピー、移動、またはリンク）を指定する必要があります。
dragenter	ドラッグジェスチャがエレメントの境界内に入ったときに送出されます。 dragenter イベントハンドラーで dataTransfer オブジェクトのプロパティを変更した場合、その変更は次の dragover イベントによって直ちに上書きされます。一方、dragenter イベントと最初の dragover イベントとの間には若干の遅延があります。そのため、プロパティを変更した場合、カーソルが点滅することがあります。多くの場合、両方のイベントに同じイベントハンドラーを使用できます。
dragleave	ドラッグジェスチャがエレメントの境界から離れたときに送出されます。
drop	ユーザーがエレメント上にデータをドロップしたときに送出されます。ドラッグ中のデータには、このイベントのハンドラー内からのみアクセスできます。

これらのイベントに応じて送出されたイベントオブジェクトはマウスイベントに似ています。(clientX, clientY) や (screenX, screenY) などのマウスイベントプロパティを使用すると、マウスの位置を確認できます。

ドラッグイベントオブジェクトの最も重要なプロパティは dataTransfer です。このプロパティには、ドラッグ中のデータが含まれています。dataTransfer オブジェクト自体には、次のプロパティとメソッドがあります。

プロパティまたはメソッド	説明
effectAllowed	ドラッグ元で許可される効果です。通常、この値は、dragstart イベントのハンドラーによって設定されます。詳しくは、178 ページの「HTML のドラッグ効果」を参照してください。
dropEffect	ターゲットまたはユーザーによって選択された効果です。dropEffect を dragover イベントハンドラーまたは dragenter イベントハンドラーで設定した場合、AIR によって、ユーザーがマウスを放したときに生じる効果を示すようにマウスカーソルが更新されます。dropEffect の設定が許可された効果のいずれかに一致しない場合、ドロップは許可されず、使用不可カーソルが表示されます。dropEffect（最新の dragover イベントまたは dragenter イベントに応答）を設定していない場合、ユーザーは、標準のオペレーティングシステム修飾キーを使用して許可された効果を選択できます。 最後の効果は、dragend に応じて送出されたオブジェクトの dropEffect プロパティによって報告されます。ユーザーが有効なターゲットの外部でマウスを放すことでドロップを破棄した場合、dropEffect が none に設定されます。
types	dataTransfer オブジェクトに格納されている各データ形式の MIME タイプのストリングを含む配列です。
getData(mimeType)	mimeType パラメーターで指定された形式のデータを取得します。 getData() メソッドは、drop イベントに応じて呼び出すことしかできません。
setData(mimeType)	mimeType パラメーターで指定された形式のデータを dataTransfer に追加します。複数の形式のデータを追加するには、MIME タイプごとに setData() を呼び出します。デフォルトのドラッグ動作によって dataTransfer オブジェクトに格納されたデータはすべて消去されます。 setData() メソッドは、dragstart イベントに応じて呼び出すことしかできません。
clearData(mimeType)	mimeType パラメーターで指定された形式のデータをすべて消去します。
setDragImage(image, offsetX, offsetY)	カスタムドラッグイメージを設定します。setDragImage() メソッドは、dragstart イベントに応じてのみ、および -webkit-user-drag CSS スタイルを element に設定することで HTML エレメント全体をドラッグする場合にのみ、呼び出すことができます。image パラメーターとしては、JavaScript エレメントまたは Image オブジェクトを指定できます。

HTML ドラッグ & ドロップの MIME タイプ

Adobe AIR 1.0 およびそれ以降

HTML ドラッグアンドドロップイベントの dataTransfer オブジェクトで使用される MIME タイプは次のとおりです。

データ形式	MIME タイプ
テキスト	"text/plain"
HTML	"text/html"
URL	"text/uri-list"
ビットマップ	"image/x-vnd.adobe.air.bitmap"
ファイルリスト	"application/x-vnd.adobe.air.file-list"

アプリケーション定義のストリングなど他の MIME ストリングも使用できます。ただし、他のアプリケーションでは、移動されたデータを認識または使用できない場合があります。必要な形式のデータを dataTransfer オブジェクトに追加する作業は、開発者が行う必要があります。

重要：ドロップされたファイルにアクセスできるのは、アプリケーションサンドボックス内で実行されているコードだけです。非アプリケーションのサンドボックス内から File オブジェクトのプロパティの読み取りまたは設定を実行しようとすると、セキュリティエラーが発生します。詳しくは、182 ページの「非アプリケーションの HTML サンドボックスでのファイルドロップの処理」を参照してください。

HTML のドラッグ効果

Adobe AIR 1.0 およびそれ以降

dataTransfer.effectAllowed プロパティを dragstart イベントのハンドラーで設定すると、ドラッグジェスチャのイニシエータで許可されるドラッグ効果を制限できます。使用できるストリング値は次のとおりです。

ストリング値	説明
"none"	ドラッグ操作は許可されません。
"copy"	ドロップ先にデータがコピーされます。ドラッグ元のデータは変更されません。
"link"	ドラッグ元へのリンクバックを介してドロップ先との間でデータが共有されます。
"move"	ドロップ先にデータがコピーされ、ドラッグ元のデータが削除されます。
"copyLink"	データをコピーまたはリンクすることができます。
"copyMove"	データをコピーまたは移動することができます。
"linkMove"	データをリンクまたは移動することができます。
"all"	データをコピー、移動、またはリンクすることができます。デフォルト動作を使用しない場合は、**All** がデフォルトの効果になります。

ドラッグジェスチャのターゲットでは、dataTransfer.dropEffect プロパティを設定することで、ユーザーがドロップを完了したときに実行されるアクションを指定できます。ドロップ効果が許可されたアクションのいずれかである場合は、該当するコピーカーソル、移動カーソル、またはリンクカーソルが表示されます。許可されたアクションでない場合は、使用不可カーソルが表示されます。ターゲットでドロップ効果が設定されていない場合、ユーザーは修飾キーを使用して許可されたアクションを選択できます。

dropEffect の値を dragover イベントと dragenter イベントの両方のハンドラーで設定します。

```
function doDragStart(event) {
    event.dataTransfer.setData("text/plain","Text to drag");
    event.dataTransfer.effectAllowed = "copyMove";
}

function doDragOver(event) {
    event.dataTransfer.dropEffect = "copy";
}

function doDragEnter(event) {
    event.dataTransfer.dropEffect = "copy";
}
```

注意：dropEffect プロパティを dragenter のハンドラーで必ず設定する必要がありますが、次の dragover イベントによってこのプロパティがデフォルト値にリセットされることに注意してください。両方のイベントに対応する dropEffect を設定します。

HTML エレメントからのデータのドラッグ

Adobe AIR 1.0 およびそれ以降

デフォルト動作を使用すると、HTML ページ内の大部分のコンテンツをドラッグでコピーすることができます。CSS プロパティである -webkit-user-select と -webkit-user-drag を使用すると、ドラッグが許可されるコンテンツを制御できます。

dragstart イベントのハンドラーでデフォルトのドラッグアウト動作を上書きします。イベントオブジェクトの dataTransfer プロパティの setData() メソッドを呼び出して、ドラッグジェスチャに独自のデータを追加します。

デフォルト動作を使用しない場合にソースオブジェクトでサポートされるドラッグ効果を指定するには、dragstart イベントに応じて送出されたイベントオブジェクトの dataTransfer.effectAllowed プロパティを設定します。任意の組み合わせの効果を選択できます。例えば、ソースエレメントでコピー効果とリンク効果の両方をサポートする場合は、このプロパティを "copyLink" に設定します。

ドラッグされるデータの設定

Flash Player 9 以降、Adobe AIR 1.0 以降

dragstart イベントのハンドラーで、dataTransfer プロパティを使用してドラッグジェスチャのデータを追加します。MIME タイプと移動するデータを dataTransfer.setData() メソッドに渡して、クリップボードにデータを格納します。

例えば、**imageOfGeorge** という ID を持つイメージエレメントがアプリケーションに含まれている場合は、次のような dragstart イベントハンドラーを使用できます。この例では、George の写真を複数のデータ形式で追加しています。これにより、ドラッグされたデータを他のアプリケーションで使用できる可能性が高まります。

```
function dragStartHandler(event){
    event.dataTransfer.effectAllowed = "copy";

    var dragImage = document.getElementById("imageOfGeorge");
    var dragFile = new air.File(dragImage.src);
    event.dataTransfer.setData("text/plain","A picture of George");
    event.dataTransfer.setData("image/x-vnd.adobe.air.bitmap", dragImage);
    event.dataTransfer.setData("application/x-vnd.adobe.air.file-list",
                               new Array(dragFile));
}
```

注意：dataTransfer オブジェクトの setData() メソッドを呼び出した場合、デフォルトのドラッグ & ドロップ動作によってデータが追加されることはありません。

HTML エレメントへのデータのドラッグ

Adobe AIR 1.0 およびそれ以降

デフォルト動作では、ページの編集可能な領域にテキストをドラッグすることしかできません。エレメントの開始タグに contenteditable 属性を追加することで、エレメントとその子が編集可能であることを指定できます。また、ドキュメントオブジェクトの designMode プロパティを "on" に設定することで、ドキュメント全体を編集可能にすることもできます。

ドラッグされたデータを受け入れ可能なエレメントの dragenter イベント、dragover イベント、および drop イベントを処理することで、ページ上での代替的なドラッグイン動作をサポートできます。

ドラッグインの有効化

Adobe AIR 1.0 およびそれ以降

ドラッグインジェスチャを処理するには、最初にデフォルト動作をキャンセルする必要があります。ドロップターゲットとして使用する HTML エレメントの dragenter イベントと dragover イベントをリッスンします。これらのイベントのハンドラーで、送出されたイベントオブジェクトの preventDefault() メソッドを呼び出します。デフォルト動作をキャンセルすると、編集不能な領域にドロップできるようになります。

ドロップされたデータの取得

Adobe AIR 1.0 およびそれ以降

ドロップされたデータには、ondrop イベントのハンドラーからアクセスできます。

```
function doDrop(event){
    droppedText = event.dataTransfer.getData("text/plain");
}
```

読み取るデータ形式の MIME タイプを dataTransfer.getData() メソッドに渡して、クリップボードのデータを読み取ります。dataTransfer オブジェクトの types プロパティを使用すると、使用可能なデータ形式を確認できます。types 配列には、使用可能な各形式の MIME タイプのストリングが含まれています。

dragenter イベントまたは dragover イベントのデフォルト動作をキャンセルする場合は、ドロップされたデータをドキュメント内の適切な場所に挿入する必要があります。マウスの位置をエレメント内の挿入場所に変換する API は存在しません。この制約により、挿入タイプのドラッグジェスチャの実装が困難になります。

例：デフォルトの HTML ドラッグインジェスチャの上書き

Adobe AIR 1.0 およびそれ以降

この例では、ドロップされたアイテムで使用可能な各データ形式を示すテーブルを表示するドロップターゲットを実装します。

アプリケーション内でテキスト、リンク、およびイメージをドラッグできるようにするために、デフォルト動作が使用されます。この例では、ドロップターゲットとなる div エレメントのデフォルトのドラッグイン動作を上書きします。編集不能な領域がドラッグインジェスチャを受け入れられるようにするための重要な手順は、dragenter イベントおよび dragover イベントに応じて送出されたイベントオブジェクトの preventDefault() メソッドを呼び出すことです。drop イベントに応じて、ハンドラーは移動されたデータを HTML 行エレメントに変換し、この行を表示用のテーブルに挿入します。

```
<html>
<head>
<title>Drag-and-drop</title>
<script language="javascript" type="text/javascript" src="AIRAliases.js"></script>
<script language="javascript">
    function init(){
        var target = document.getElementById('target');
        target.addEventListener("dragenter", dragEnterOverHandler);
        target.addEventListener("dragover", dragEnterOverHandler);
        target.addEventListener("drop", dropHandler);

        var source = document.getElementById('source');
        source.addEventListener("dragstart", dragStartHandler);
        source.addEventListener("dragend", dragEndHandler);

        emptyRow = document.getElementById("emptyTargetRow");
    }

    function dragStartHandler(event){
        event.dataTransfer.effectAllowed = "copy";
    }

    function dragEndHandler(event){
        air.trace(event.type + ": " + event.dataTransfer.dropEffect);
    }
```

```
    function dragEnterOverHandler(event){
        event.preventDefault();
    }

    var emptyRow;
    function dropHandler(event){
        for(var prop in event){
            air.trace(prop + " = " + event[prop]);
        }
        var row = document.createElement('tr');
        row.innerHTML = "<td>" + event.dataTransfer.getData("text/plain") + "</td>" +
            "<td>" + event.dataTransfer.getData("text/html") + "</td>" +
            "<td>" + event.dataTransfer.getData("text/uri-list") + "</td>" +
            "<td>" + event.dataTransfer.getData("application/x-vnd.adobe.air.file-list") +
            "</td>";

        var imageCell = document.createElement('td');
        if((event.dataTransfer.types.toString()).search("image/x-vnd.adobe.air.bitmap") > -1){
            imageCell.appendChild(event.dataTransfer.getData("image/x-vnd.adobe.air.bitmap"));
        }
        row.appendChild(imageCell);
        var parent = emptyRow.parentNode;
        parent.insertBefore(row, emptyRow);
    }
</script>
</head>
<body onLoad="init()" style="padding:5px">
<div>
    <h1>Source</h1>
       <p>Items to drag:</p>
    <ul id="source">
        <li>Plain text.</li>
           <li>HTML <b>formatted</b> text.</li>
           <li>A <a href="http://www.adobe.com">URL.</a></li>
           <li><img src="icons/AIRApp_16.png" alt="An image"/></li>
           <li style="-webkit-user-drag:none;">
        Uses "-webkit-user-drag:none" style.
           </li>
           <li style="-webkit-user-select:none;">
        Uses "-webkit-user-select:none" style.
           </li>

       </ul>
</div>
<div id="target" style="border-style:dashed;">
    <h1 >Target</h1>
       <p>Drag items from the source list (or elsewhere).</p>
       <table id="displayTable" border="1">
        <tr><th>Plain text</th><th>Html text</th><th>URL</th><th>File list</th><th>Bitmap Data</th></tr>
           <tr
id="emptyTargetRow"><td> </td><td> </td><td> </td><td> </td><td> </td></tr>
       </table>
       </div>
</div>
</body>
</html>
```

非アプリケーションの HTML サンドボックスでのファイルドロップの処理

Adobe AIR 1.0 およびそれ以降

AIR アプリケーションにファイルをドラッグしたときに生成される File オブジェクトに、非アプリケーションのコンテンツからアクセスすることはできません。また、これらの File オブジェクトのいずれかをサンドボックスブリッジ経由でアプリケーションコンテンツに渡すこともできません（オブジェクトのプロパティには、直列化時にアクセスする必要があります）。ただし、HTMLLoader オブジェクトの AIR nativeDragDrop イベントをリッスンすることで、この場合でもアプリケーションにファイルをドロップできます。

ユーザーが非アプリケーションのコンテンツをホストするフレームにファイルをドロップした場合、通常、ドロップイベントは子から親へと伝達されません。ただし、HTMLLoader（AIR アプリケーション内のすべての HTML コンテンツのコンテナ）によって送出されたイベントは HTML イベントフローの一部ではないため、アプリケーションコンテンツのドロップイベントを受け取ることは可能です。

親ドキュメントでは、ファイルドロップイベントを受け取るために、window.htmlLoader によって提供された参照を使用して、HTMLLoader オブジェクトにイベントリスナーを追加します。

```
window.htmlLoader.addEventListener("nativeDragDrop",function(event){
    var filelist = event.clipboard.getData(air.ClipboardFormats.FILE_LIST_FORMAT);
    air.trace(filelist[0].url);
});
```

NativeDragEvent オブジェクトは、対応する HTML イベントと同様に動作します。ただし、両者の間では、一部のプロパティとメソッドの名前およびデータ型が異なります。例えば、HTML イベントの dataTransfer プロパティは、ActionScript イベントの clipboard プロパティと同じ目的に使用されます。これらのクラスの使用について詳しくは、「Adobe ActionScript 3.0 開発ガイド」および「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。

次の例では、リモートサンドボックス（http://localhost/）に子ページを読み込む親ドキュメントを使用しています。親ドキュメントでは、HTMLLoader オブジェクトの nativeDragDrop イベントをリッスンし、ファイルの URL をトレースします。

```
<html>
<head>
<title>Drag-and-drop in a remote sandbox</title>
<script language="javascript" type="text/javascript" src="AIRAliases.js"></script>
<script language="javascript">
    window.htmlLoader.addEventListener("nativeDragDrop",function(event){
        var filelist = event.clipboard.getData(air.ClipboardFormats.FILE_LIST_FORMAT);
        air.trace(filelist[0].url);
    });
</script>
</head>
<body>
       <iframe src="child.html"
               sandboxRoot="http://localhost/"
               documentRoot="app:/"
               frameBorder="0"  width="100%" height="100%">
       </iframe>
</body>
</html>
```

子ドキュメントでは、HTML dragenter および dragover のイベントハンドラーで Event オブジェクトの preventDefault() メソッドを呼び出すことで、有効なドロップターゲットを提供する必要があります。そうしないとドロップイベントが発生しません。

```
 <html>
<head>
    <title>Drag and drop target</title>
    <script language="javascript" type="text/javascript">
        function preventDefault(event){
            event.preventDefault();
        }
    </script>
</head>
<body ondragenter="preventDefault(event)" ondragover="preventDefault(event)">
<div>
<h1>Drop Files Here</h1>
</div>
</body>
</html>
```

ファイルプロミスのドロップ

Adobe AIR 2 以降

ファイルプロミスはドラッグ & ドロップクリップボード形式です。これにより、ユーザーは、まだ存在しないファイルを AIR アプリケーションからドラッグできます。例えば、ファイルプロミスを使用すると、アプリケーションでプロキシアイコンをデスクトップフォルダーにドラッグできます。プロキシアイコンは、ある URL に存在することがわかっているファイルまたはデータを表します。アイコンをドロップすると、ランタイムによってデータがダウンロードされ、ドロップ位置にファイルが書き込まれます。

AIR アプリケーションの URLFilePromise クラスを使用すると、ある URL でアクセス可能なファイルをドラッグ & ドロップできます。URLFilePromise 実装は、AIR 2 SDK の一部として aircore ライブラリで提供されています。SDK frameworks/libs/air ディレクトリにある aircore.swc ファイルまたは aircore.swf ファイルのどちらかを使用してください。

または、ランタイム flash.desktop パッケージに定義されている IFilePromise インターフェイスを使用して、独自のファイルプロミスロジックを実装することもできます。

概念上、ファイルプロミスは、クリップボード上のデータハンドラー関数を使用した遅延レンダリングに似ています。ファイルをドラッグ & ドロップするときは、遅延レンダリングの代わりにファイルプロミスを使用します。遅延レンダリングテクニックを使用すると、データが生成またはダウンロードされるときに、ドラッグ操作が一時停止する場合があり、望ましくありません。遅延レンダリングは、ファイルプロミスがサポートされていないコピー & ペースト操作に使用してください。

ファイルプロミス使用時の制限

ドラッグ & ドロップクリップボードに配置できる他のデータ形式と比較して、ファイルプロミスには次の制限があります。

- ファイルプロミスは AIR アプリケーションからドラッグするのみで、AIR アプリケーションにドロップすることはできません。
- ファイルプロミスは、すべてのオペレーティングシステムでサポートされているわけではありません。Clipboard.supportsFilePromise プロパティを使用して、ホストシステムでファイルプロミスがサポートされているかどうかをテストしてください。ファイルプロミスがサポートされていないシステムでは、ファイルデータをダウンロードまたは生成するための代替のメカニズムを提供する必要があります。
- ファイルプロミスは、コピー & ペーストクリップボード（Clipboard.generalClipboard）では使用できません。

関連項目

flash.desktop.IFilePromise

air.desktop.URLFilePromise

リモートファイルのドロップ

Adobe AIR 2 以降

URL 上のファイルまたはデータを表すファイルプロミスオブジェクトの作成には、URLFilePromise クラスを使用します。FILE_PROMISE_LIST クリップボード形式を使用して、1 つまたは複数のファイルプロミスオブジェクトをクリップボードに追加します。次の例では、http://www.example.com/foo.txt にある単一のファイルがダウンロードされ、ドロップ位置に bar.txt という名前で保存されます。リモートとローカルのファイル名を一致させる必要はありません。

```
<html>
<head>
<script src="AIRAliases.js"></script>
<script src="aircore.swf" type="application/x-shockwave-flash"></script>
<script language="javascript">
    function init(){
        var source = document.getElementById('source');
        source.addEventListener("dragstart", dragStartHandler);
    }

    function dragStartHandler(event){
        event.preventDefault();
        startDrag();
    }
    function startDrag()
    {
        var filePromise = new air.URLFilePromise(); //defined in aircore.swf
        filePromise.request = new air.URLRequest("http://example.com/foo.txt");
        filePromise.relativePath = "bar.txt";
        var fileList = new Array( filePromise );
        var clipboard = new air.Clipboard();
        clipboard.setData( air.ClipboardFormats.FILE_PROMISE_LIST_FORMAT, fileList );
        air.NativeDragManager.doDrag( window.htmlLoader, clipboard );
    }
</script>
</head>
<body onLoad="init()">
    <p id="source" style="-webkit-user-drag:element; -webkit-user-select:none;">
        Drag to file system
    </p>
</body>
</html>
```

クリップボードに割り当てられている配列にファイルプロミスオブジェクトをさらに追加すると、一度に複数のファイルをドラッグできます。また、relativePath プロパティでサブディレクトリを指定して、操作に含まれているファイルの一部または全部をドロップ位置と相対的なサブフォルダーに配置することもできます。

次の例は、複数のファイルプロミスを含むドラッグ操作の開始方法を示しています。この例では、html ページ **article.html** が、2 つのリンクされたイメージファイルと共にファイルプロミスとしてクリップボードに配置されます。イメージは **images** サブフォルダーにコピーされ、相対リンクが維持されます。

```
<html>
<head>
<script src="AIRAliases.js"></script>
<script src="aircore.swf" type="application/x-shockwave-flash"></script>
<script language="javascript">
    function init(){
        var source = document.getElementById('source');
        source.addEventListener("dragstart", dragStartHandler);
    }

    function dragStartHandler(event){
        event.preventDefault();
        startDrag();
    }
    function startDrag()
    {
        var filePromise = new air.URLFilePromise();
        filePromise.request = new air.URLRequest("http://example.com/article.html");
        filePromise.relativePath = "article.html";

        var image1Promise = new air.URLFilePromise();
        image1Promise.request = new air.URLRequest("http://example.com/images/img_1.jpg");
        image1Promise.relativePath = "images/img_1.html";
        var image2Promise = new air.URLFilePromise();
        image2Promise.request = new air.URLRequest("http://example.com/images/img_2.jpg");
        image2Promise.relativePath = "images/img_2.jpg";

        //Put the promise objects onto the clipboard inside an array
        var fileList = new Array( filePromise, image1Promise, image2Promise );
        var clipboard = new air.Clipboard();
        clipboard.setData( air.ClipboardFormats.FILE_PROMISE_LIST_FORMAT, fileList );
        air.NativeDragManager.doDrag( window.htmlLoader, clipboard );
    }
</script>
</head>
<body onLoad="init()">
    <p id="source" style="-webkit-user-drag:element; -webkit-user-select:none;">
        Drag to file system
    </p>
</body>
</html>
```

IFilePromise インターフェイスの実装

Adobe AIR 2 以降

URLFilePromise オブジェクトを使用してアクセスできないリソースにファイルプロミスを提供するために、IFilePromise インターフェイスをカスタムクラスに実装できます。IFilePromise インターフェイスは、ファイルプロミスがドロップされたときにファイルに書き込まれるデータにアクセスするために AIR ランタイムで使用するメソッドおよびプロパティを定義します。

注意：JavaScript 言語はインターフェイスの実装をサポートしていないので、ActionScript を使用しないと、独自のファイルプロミスロジックを実装することはできません。もちろん、<script> タグを使用して、ActionScript クラスを含む SWF ファイルを HTML ページに読み込み、JavaScript コードでそれらのクラスにアクセスすることができます。

IFilePromise 実装は、ファイルプロミスにデータを提供する別のオブジェクトを AIR ランタイムに渡します。このオブジェクトは IDataInput インターフェイスを実装する必要があります。AIR ランタイムでは、これを使用してデータを読み取ります。例えば、IFilePromise を実装する URLFilePromise クラスでは、URLStream オブジェクトをデータプロバイダーとして使用します。

AIR では、データを同期でも非同期でも読み取ることができます。IFilePromise 実装では、isAsync プロパティで対応する値を返すことにより、どのアクセスモードがサポートされているかを報告します。非同期データアクセスが提供されている場合、データプロバイダーオブジェクトは IEventDispatcher インターフェイスを実装し、open、progress、complete などの必要なイベントを送出しなければなりません。

ファイルプロミスのデータプロバイダーとして使用できるのは、カスタムクラス、または次のいずれかのビルトインクラスです。

- ByteArray（同期）
- FileStream（同期または非同期）
- Socket（非同期）
- URLStream（非同期）

IFilePromise インターフェイスを実装するには、次の関数およびプロパティにコードを指定する必要があります。

- open():IDataInput — プロミスファイルのデータの読み取り元であるデータプロバイダーオブジェクトを返します。このオブジェクトは、IDataInput インターフェイスを実装する必要があります。データが非同期で提供されている場合、オブジェクトでは IEventDispatcher インターフェイスも実装し、必要なイベントを送出する必要があります（188 ページの「ファイルプロミスでの非同期データプロバイダーの使用」を参照）。
- get relativePath():String — 作成されたファイルのパスを、ファイル名も含めて指定します。パスは、ユーザーがドラッグ & ドロップ操作で選択したドロップ位置と相対的に解決されます。ホストオペレーティングシステムに適切な区切り文字がパスで使用されていることを確認するには、ディレクトリを含むパスを指定するときに File.separator 定数を使用します。setter 関数を追加するか、コンストラクターパラメーターを使用すると、実行時にパスを設定できます。
- get isAsync():Boolean — データプロバイダーオブジェクトが非同期、同期のどちらでデータを提供するかを AIR ランタイムに通知します。
- close():void — データが完全に読み込まれたとき（または、エラーにより読み取りを続行できない場合）に、ランタイムによって呼び出されます。この関数を使用するとリソースをクリーンアップできます。
- reportError(e:ErrorEvent):void — データの読み取りエラーが発生したときに、ランタイムによって呼び出されます。

ファイルプロミスを含むドラッグ & ドロップ操作中に、IFilePromise のすべてのメソッドがランタイムによって呼び出されます。通常は、アプリケーションロジックがこれらのメソッドを直接呼び出すことはありません。

ファイルプロミスでの同期データプロバイダーの使用

Adobe AIR 2 以降

IFilePromise インターフェイスを実装する最も簡単な方法は、ByteArray や同期 FileStream など、同期のデータプロバイダーオブジェクトを使用することです。次の例では、ByteArray オブジェクトが作成され、データが指定されて、open() メソッドを呼び出したときに返されます。

```
package
{
    import flash.desktop.IFilePromise;
    import flash.events.ErrorEvent;
    import flash.utils.ByteArray;
    import flash.utils.IDataInput;

    public class SynchronousFilePromise implements IFilePromise
    {
        private const fileSize:int = 5000; //size of file data
        private var filePath:String = "SynchronousFile.txt";

        public function get relativePath():String
        {
            return filePath;
        }

        public function get isAsync():Boolean
        {
            return false;
        }

        public function open():IDataInput
        {
            var fileContents:ByteArray = new ByteArray();

            //Create some arbitrary data for the file
            for( var i:int = 0; i < fileSize; i++ )
            {
                fileContents.writeUTFBytes( 'S' );
            }

            //Important: the ByteArray is read from the current position
            fileContents.position = 0;
            return fileContents;
        }

        public function close():void
        {
            //Nothing needs to be closed in this case.
        }

        public function reportError(e:ErrorEvent):void
        {
            trace("Something went wrong: " + e.errorID + " - " + e.type + ", " + e.text );
        }
    }
}
```

実際には、同期ファイルプロミスの用途は限られています。データ量が少ない場合は、一時ディレクトリに同じくらい簡単にファイルを作成し、通常のファイルリスト配列をドラッグ & ドロップクリップボードに追加できます。一方で、データ量が多い場合や、データを作成するとコンピューター処理上の負荷が高い場合は、同期プロセスに長時間かかります。同期プロセスが長引くと、UI の更新がかなり長時間に渡ってブロックされ、アプリケーションが応答していないように見える場合があります。この問題を回避するために、タイマーによって制御される非同期データプロバイダーを作成できます。

ファイルプロミスでの非同期データプロバイダーの使用

Adobe AIR 2 以降

非同期データプロバイダーオブジェクトを使用するときは、IFilePromise の isAsync プロパティを true に設定し、open() メソッドによって返されるオブジェクトに IEventDispatcher インターフェイスを実装する必要があります。様々なビルトインオブジェクトをデータプロバイダーとして使用できるように、ランタイムは複数の代替イベントを監視します。例えば、progress イベントは FileStream オブジェクトおよび URLStream オブジェクトによって送出され、socketData イベントは Socket オブジェクトによって送出されます。ランタイムは、これらすべてのオブジェクトから適切なイベントを監視します。

次の各イベントは、データプロバイダーオブジェクトからのデータの読み取りプロセスを制御します。

- Event.OPEN — データソースの準備が完了していることをランタイムに通知します。
- ProgressEvent.PROGRESS — データが使用可能であることをランタイムに通知します。ランタイムは、使用可能なデータの量をデータプロバイダーオブジェクトから読み取ります。
- ProgressEvent.SOCKET_DATA — データが使用可能であることをランタイムに通知します。socketData イベントはソケットベースのオブジェクトによって送出されます。その他のオブジェクトタイプについては、progress イベントを送出する必要があります（ランタイムは両方のイベントを監視してデータの読み取りが可能なタイミングを検出します）。
- Event.COMPLETE — データがすべて読み込まれたことをランタイムに通知します。
- Event.CLOSE — データがすべて読み込まれたことをランタイムに通知します（この場合、ランタイムは close と complete の両方を監視します）。
- IOErrorEvent.IOERROR — データの読み取りエラーが発生したことをランタイムに通知します。ランタイムはファイル作成を中止し、IFilePromise の close() メソッドを呼び出します。
- SecurityErrorEvent.SECURITY_ERROR — セキュリティエラーが発生したことをランタイムに通知します。ランタイムはファイル作成を中止し、IFilePromise の close() メソッドを呼び出します。
- HTTPStatusEvent.HTTP_STATUS — 使用可能なデータがエラーメッセージ（404 ページなど）ではなく必要なコンテンツを表していることを確認するために、httpResponseStatus と共にランタイムによって使用されます。このイベントは HTTP プロトコルに基づくオブジェクトによって送出されます。
- HTTPStatusEvent.HTTP_RESPONSE_STATUS — 使用可能なデータが必要なコンテンツを表していることを確認するために、httpStatus と共にランタイムによって使用されます。このイベントは HTTP プロトコルに基づくオブジェクトによって送出されます。

データプロバイダーでは、これらのイベントが次の順序で送出されます。

1. open イベント
2. progress イベントまたは socketData イベント
3. complete イベントまたは close イベント

注意：ビルトインオブジェクト、FileStream、Socket、および URLStream では、該当するイベントが自動的に送出されます。

次の例では、カスタムの非同期データプロバイダーを使用してファイルプロミスを作成します。データプロバイダークラスによって ByteArray が拡張され（IDataInput サポート用）、IEventDispatcher インターフェイスが実装されます。タイマーイベントごとに、オブジェクトではデータのグループを生成し、データが使用可能であることをランタイムに通知する progress イベントを送出します。十分なデータが生成されたら、オブジェクトは complete イベントを送出します。

```
package
{
import flash.events.Event;
import flash.events.EventDispatcher;
import flash.events.IEventDispatcher;
import flash.events.ProgressEvent;
import flash.events.TimerEvent;
import flash.utils.ByteArray;
import flash.utils.Timer;

[Event(name="open", type="flash.events.Event.OPEN")]
[Event(name="complete",  type="flash.events.Event.COMPLETE")]
[Event(name="progress", type="flash.events.ProgressEvent")]
[Event(name="ioError", type="flash.events.IOErrorEvent")]
[Event(name="securityError", type="flash.events.SecurityErrorEvent")]
public class AsyncDataProvider extends ByteArray implements IEventDispatcher
{
    private var dispatcher:EventDispatcher = new EventDispatcher();
    public var fileSize:int = 0; //The number of characters in the file
    private const chunkSize:int = 1000; //Amount of data written per event
    private var dispatchDataTimer:Timer = new Timer( 100 );
    private var opened:Boolean = false;

    public function AsyncDataProvider()
    {
        super();
        dispatchDataTimer.addEventListener( TimerEvent.TIMER, generateData );
    }

    public function begin():void{
        dispatchDataTimer.start();
    }

    public function end():void
    {
        dispatchDataTimer.stop();
    }
    private function generateData( event:Event ):void
    {
        if( !opened )
        {
            var open:Event = new Event( Event.OPEN );
            dispatchEvent( open );
            opened = true;
        }
        else if( position + chunkSize < fileSize )
        {
            for( var i:int = 0; i <= chunkSize; i++ )
            {
                writeUTFBytes( 'A' );
            }
            //Set position back to the start of the new data
            this.position -= chunkSize;
            var progress:ProgressEvent =
                new ProgressEvent( ProgressEvent.PROGRESS, false, false, bytesAvailable, bytesAvailable +
chunkSize);
            dispatchEvent( progress )
        }
        else
        {
            var complete:Event = new Event( Event.COMPLETE );
            dispatchEvent( complete );
```

```
        }
    }
     //IEventDispatcher implementation
    public function addEventListener(type:String, listener:Function, useCapture:Boolean=false,
priority:int=0, useWeakReference:Boolean=false):void
    {
        dispatcher.addEventListener( type, listener, useCapture, priority, useWeakReference );
    }

    public function removeEventListener(type:String, listener:Function, useCapture:Boolean=false):void
    {
        dispatcher.removeEventListener( type, listener, useCapture );
    }

    public function dispatchEvent(event:Event):Boolean
    {
        return dispatcher.dispatchEvent( event );
    }

    public function hasEventListener(type:String):Boolean
    {
        return dispatcher.hasEventListener( type );
    }

    public function willTrigger(type:String):Boolean
    {
        return dispatcher.willTrigger( type );
    }
}
}
```

注意：この例の AsyncDataProvider クラスでは、ByteArray が拡張されるため、EventDispatcher も同時に拡張することはできません。IEventDispatcher インターフェイスを実装するには、内部の EventDispatcher オブジェクトを使用して、その内部オブジェクトに対して IEventDispatcher メソッド呼び出しを転送します。また、EventDispatcher を拡張して IDataInput を実装（または両方のインターフェイスを実装）することもできます。

非同期の IFilePromise 実装は、同期の実装とほぼ同じです。主な違いは、isAsync が true を返し、open() メソッドが非同期データオブジェクトを返すことです。

```
package
{
    import flash.desktop.IFilePromise;
    import flash.events.ErrorEvent;
    import flash.events.EventDispatcher;
    import flash.utils.IDataInput;

    public class AsynchronousFilePromise extends EventDispatcher implements IFilePromise
    {
        private var fileGenerator:AsyncDataProvider;
        private const fileSize:int = 5000; //size of file data
        private var filePath:String = "AsynchronousFile.txt";

        public function get relativePath():String
        {
            return filePath;
        }

        public function get isAsync():Boolean
        {
            return true;
        }
```

```
        public function open():IDataInput
        {
            fileGenerator = new AsyncDataProvider();
            fileGenerator.fileSize = fileSize;
            fileGenerator.begin();
            return fileGenerator;
        }

        public function close():void
        {
            fileGenerator.end();
        }

        public function reportError(e:ErrorEvent):void
        {
            trace("Something went wrong: " + e.errorID + " - " + e.type + ", " + e.text );
        }
    }
}
```

第 13 章：コピー＆ペースト

Flash Player 10 以降、Adobe AIR 1.0 以降

システムクリップボードとの間で情報をコピーするには、クリップボード API のクラスを使用します。Adobe® Flash® Player または Adobe® AIR® で実行されているアプリケーションとの間で送受信できるデータ形式は次のとおりです。

- テキスト
- HTML 形式のテキスト
- リッチテキスト形式データ
- 直列化されたオブジェクト
- オブジェクト参照（元のアプリケーション内でのみ有効）
- ビットマップ（AIR のみ）
- ファイル（AIR のみ）
- URL ストリング（AIR のみ）

コピー＆ペーストの基礎

Flash Player 10 以降、Adobe AIR 1.0 以降

コピー＆ペースト API には、次のクラスが含まれます。

パッケージ	クラス
flash.desktop	• Clipboard • ClipboardFormats • ClipboardTransferMode

静的 Clipboard.generalClipboard プロパティは、オペレーティングシステムのクリップボードを表します。Clipboard クラスには、クリップボードオブジェクトのデータを読み書きするためのメソッドが用意されています。

HTMLLoader クラス（AIR 内）および TextField クラスは、通常のコピー & ペーストのキーボードショートカットのデフォルト動作を実装します。コピー＆ペーストのショートカット動作をカスタムコンポーネントに実装するには、これらのキーストロークを直接リッスンします。また、ネイティブメニューコマンドをキーボードショートカット使用し、キーストロークに間接的に応答することもできます。

同じ情報の異なる表現を単一の Clipboard オブジェクトで使用できるようにして、他のアプリケーションがデータを認識して使用する可能性を高めることができます。例えば、イメージをイメージデータ、直列化された Bitmap オブジェクト、ファイルとして含めることができます。ある形式でのデータのレンダリングを、その形式のデータが読み取られるまでは形式の実際の作成が行われないように遅延させることができます。

システムクリップボードとの間の読み書き

Flash Player 10 以降、Adobe AIR 1.0 以降

オペレーティングシステムのクリップボードを読み取るには、次のように Clipboard.generalClipbooard オブジェクトの getData() メソッドを呼び出して、読み取り形式で渡します。

```
import flash.desktop.Clipboard;
import flash.desktop.ClipboardFormats;

if(Clipboard.generalClipboard.hasFormat(ClipboardFormats.TEXT_FORMAT)){
    var text:String = Clipboard.generalClipboard.getData(ClipboardFormats.TEXT_FORMAT);
}
```

注意：Flash Player で実行されているコンテンツ、または AIR のアプリケーション以外のサンドボックスで実行されているコンテンツは、paste イベントのイベントハンドラーでのみ getData() メソッドを呼び出すことができます。つまり、AIR アプリケーションサンドボックスで実行されているコードのみが、paste イベントハンドラー外の getData() メソッドを呼び出すことができます。

クリップボードに書き込むには、1 つ以上の形式で Clipboard.generalClipboard オブジェクトにデータを追加します。同じ形式の既存のデータは、自動的に上書きされます。ただし、新しいデータを書き込む前にシステムクリップボードのクリアも行い、他の形式の無関係なデータも削除されるようにすることをお勧めします。

```
import flash.desktop.Clipboard;
import flash.desktop.ClipboardFormats;

var textToCopy:String = "Copy to clipboard.";
Clipboard.generalClipboard.clear();
Clipboard.generalClipboard.setData(ClipboardFormats.TEXT_FORMAT, textToCopy, false);
```

注意：Flash Player で実行されているコンテンツ、または AIR のアプリケーション以外のサンドボックスで実行されているコンテンツは、setData() メソッドを、キーボードイベント、マウスイベント、copy イベント、cut イベントなどのユーザーイベントのイベントハンドラー内でのみ呼び出すことができます。つまり、AIR アプリケーションサンドボックスで実行されているコードのみが、ユーザーイベントハンドラー外の setData() メソッドを呼び出すことができます。

AIR での HTML コピー＆ペースト

Adobe AIR 1.0 およびそれ以降

Adobe AIR の HTML 環境では、コピー＆ペーストに関して独自のイベントとデフォルト動作が提供されています。アプリケーションサンドボックス内で実行するコードのみが、AIR の Clipboard.generalClipboard オブジェクトを通して、システムクリップボードに直接アクセスできます。非アプリケーションのサンドボックス内の JavaScript コードは、HTML ドキュメントのエレメントから送出されたコピーイベントまたはペーストイベントのいずれかに応答して送出されるイベントオブジェクトを通して、クリップボードにアクセスできます。

コピー＆ペーストイベントには、copy、cut、paste などがあります。これらのイベントに対して送出されるオブジェクトでは、clipboardData プロパティを使用してクリップボードにアクセスできます。

デフォルト動作

Adobe AIR 1.0 およびそれ以降

デフォルトでは、AIR は、コピーコマンドに応答して選択されたアイテムをコピーします。コピーコマンドは、キーボードショートカットまたはコンテキストメニューによって生成されます。編集可能な領域では、AIR はカットコマンドに応答してテキストをカットするか、またはペーストコマンドに応答してカーソル位置または選択位置にテキストをペーストします。

デフォルトの動作を実行しないようにするには、送出されたイベントオブジェクトの preventDefault() メソッドを、イベントハンドラーで呼び出します。

イベントオブジェクトの clipboardData プロパティの使用

Adobe AIR 1.0 およびそれ以降

コピーイベントまたはペーストイベントのいずれかの結果として送出されるイベントオブジェクトの clipboardData プロパティを使用すると、クリップボードのデータを読み書きできます。

コピーイベントまたはカットイベントの処理時にクリップボードに書き込むには、clipboardData オブジェクトの setData() メソッドを使用し、コピーするデータと MIME タイプを渡します。

```
function customCopy(event){
    event.clipboardData.setData("text/plain", "A copied string.");
}
```

ペーストされるデータにアクセスするには、clipboardData オブジェクトの getData() メソッドを使用し、データ形式の MIME タイプを渡します。使用できる形式は、types プロパティによって通知されます。

```
function customPaste(event){
    var pastedData = event.clipboardData("text/plain");
}
```

getData() メソッドと types プロパティには、paste イベントから送出されるイベントオブジェクト内でのみアクセスできます。

次の例では、デフォルトのコピー&ペースト動作を HTML ページでオーバーライドする方法を示します。copy イベントハンドラーは、コピーされるテキストをイタリック体にして、HTML テキストとしてクリップボードにコピーします。cut イベントハンドラーは、選択されたデータをクリップボードにコピーし、ドキュメントからデータを削除します。paste ハンドラーは、クリップボードの内容を HTML として挿入し、さらに挿入部分のスタイルを太字にします。

```
 <html>
<head>
    <title>Copy and Paste</title>
    <script language="javascript" type="text/javascript">
        function onCopy(event){
            var selection = window.getSelection();
            event.clipboardData.setData("text/html","<i>" + selection + "</i>");
            event.preventDefault();
        }

        function onCut(event){
             var selection = window.getSelection();
             event.clipboardData.setData("text/html","<i>" + selection + "</i>");
             var range = selection.getRangeAt(0);
             range.extractContents();

            event.preventDefault();
        }

        function onPaste(event){
            var insertion = document.createElement("b");
            insertion.innerHTML = event.clipboardData.getData("text/html");
             var selection = window.getSelection();
             var range = selection.getRangeAt(0);
             range.insertNode(insertion);
            event.preventDefault();
        }
    </script>
</head>
<body onCopy="onCopy(event)"
     onPaste="onPaste(event)"
     onCut="onCut(event)">
<p>Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium
doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore
veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam
voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur
magni dolores eos qui ratione voluptatem sequi nesciunt.</p>
</body>
</html>
```

クリップボードのデータ形式

Flash Player 10 以降、Adobe AIR 1.0 以降

クリップボードの形式では、Clipboard オブジェクトに配置されるデータについて説明します。Flash Player または AIR は、ActionScript データ型の形式とシステムクリップボード形式間で標準データ形式を自動的に変換します。また、アプリケーションオブジェクトは、アプリケーション定義の形式を使用して ActionScript ベースのアプリケーション内およびアプリケーション間で転送することができます。

Clipboard オブジェクトは、同じ情報の異なる形式での表現を格納できます。例えば、Sprite を表すクリップボードオブジェクトには、ファイルシステムに Sprite の表現をコピーまたはドラッグするために、同一のアプリケーション内で使用する参照形式、Flash Player または AIR で実行される別のアプリケーションが使用する直列化形式、イメージ エディターが使用するビットマップ形式、ファイル リスト形式、さらに、PNG ファイルをエンコードするための遅延レンダリング形式が含まれている可能性があります。

標準データ形式

Flash Player 10 以降、Adobe AIR 1.0 以降

標準形式名を定義する定数は、ClipboardFormats クラスで提供されています。

定数	説明
TEXT_FORMAT	テキスト形式データは、ActionScript の String クラスとの間で変換されます。
HTML_FORMAT	HTML マークアップを含むテキストです。
RICH_TEXT_FORMAT	リッチテキスト形式データは、ActionScript の ByteArray クラスとの間で変換されます。RTF マークアップはどのような方法でも解釈または変換されません。
BITMAP_FORMAT	（AIR のみ）ビットマップ形式データは、ActionScript の BitmapData クラス間で変換されます。
FILE_LIST_FORMAT	（AIR のみ）ファイルリスト形式データは、ActionScript のファイルオブジェクトの配列間で変換されます。
URL_FORMAT	（AIR のみ）URL 形式データは、ActionScript の String クラス間で変換されます。

AIR アプリケーションでホストされる HTML コンテンツで copy、cut または paste イベントに応答してデータをコピーおよびペーストするときは、ClipboardFormat ストリングの代わりに MIME タイプを使用する必要があります。データの有効な MIME タイプは、次のとおりです。

MIME タイプ	説明
テキスト	"text/plain"
URL	"text/uri-list"
ビットマップ	"image/x-vnd.adobe.air.bitmap"
ファイルリスト	"application/x-vnd.adobe.air.file-list"

注意：リッチテキスト形式データは、HTML コンテンツ内の paste イベントの結果として送出されたイベント オブジェクトの clipboardData プロパティから利用することはできません。

カスタムデータ形式

Flash Player 10 以降、Adobe AIR 1.0 以降

アプリケーション定義のカスタム形式を使用し、参照として、または直列化されたコピーとして、オブジェクトを転送できます。参照は、同じアプリケーション内でのみ有効です。直列化されたオブジェクトは アプリケーション間で転送できますが、直列化および直列化解除しても有効なオブジェクトについてのみ使用できます。通常、プロパティが単純型または直列化可能なオブジェクトの場合は、オブジェクトを直列化できます。

直列化されたオブジェクトを Clipboard オブジェクトに追加するには、serializable パラメーターを true に設定して、Clipboard.setData() メソッドを呼び出します。形式名としては、いずれかの標準形式、またはアプリケーションで定義されている任意のストリングを使用できます。

転送モード

Flash Player 10 以降、Adobe AIR 1.0 以降

カスタムデータ形式でクリップボードにオブジェクトを書き込む場合は、元のオブジェクトへの参照またはその直列化コピーとしてオブジェクトデータをクリップボードから読み取ることができます。オブジェクトを参照として転送するか直列化されたコピーとして転送するかを決定する 4 つの転送モードがあります。

転送モード	説明
ClipboardTransferModes.ORIGINAL_ONLY	参照のみが返されます。参照を使用できない場合は、null 値が返されます。
ClipboardTransferModes.ORIGINAL_PREFFERED	使用できる場合は、参照が返されます。参照を使用できない場合は、直列化されたコピーが返されます。
ClipboardTransferModes.CLONE_ONLY	直列化されたコピーのみが返されます。直列化されたコピーを使用できない場合は、null 値が返されます。
ClipboardTransferModes.CLONE_PREFFERED	使用できる場合は、直列化されたコピーが返されます。直列化されたコピーを使用できない場合は、参照が返されます。

カスタムデータ形式の読み取りと書き込み

Flash Player 10 以降、Adobe AIR 1.0 以降

クリップボードにオブジェクトを書き込むときの形式パラメータとして、予約された接頭辞である air: または flash: で始まらない任意のストリングを使用できます。そのオブジェクトを読み取るには、形式として同じストリングを使用します。次の例では、クリップボードにオブジェクトを読み書きする方法を示します。

```
public function createClipboardObject(object:Object):Clipboard{
    var transfer:Clipboard = Clipboard.generalClipboard;
    transfer.setData("object", object, true);
}

function createClipboardObject(object){
    var transfer = new air.Clipboard();
    transfer.setData("object", object, true);
}
```

クリップボードオブジェクトから直列化オブジェクトを抽出する（ドロップまたはペースト操作後）には、同じ形式名および CLONE_ONLY または CLONE_PREFFERED 転送モードを使用します。

```
var transfer:Object = clipboard.getData("object", ClipboardTransferMode.CLONE_ONLY);

var transfer = clipboard.getData("object", air.ClipboardTransferMode.CLONE_ONLY);
```

参照は、Clipboard オブジェクトに常に追加されます。（ドロップ操作またはペースト操作の後で）直列化されたコピーの代わりに参照をクリップボードオブジェクトから抽出するには、ORIGINAL_ONLY または ORIGINAL_PREFFERED 転送モードを使用します。

```
var transferredObject:Object =
    clipboard.getData("object", ClipboardTransferMode.ORIGINAL_ONLY);

var transferredObject =
    clipboard.getData("object", air.ClipboardTransferMode.ORIGINAL_ONLY);
```

参照は、Clipboard オブジェクトが現在のアプリケーションから発生している場合にのみ有効です。参照が使用できるときには参照にアクセスし、参照が使用できないときには直列化されたクローンにアクセスするには、ORIGINAL_PREFFERED 転送モードを使用します。

遅延レンダリング

Flash Player 10 以降、Adobe AIR 1.0 以降

データ形式の作成に大量の処理が必要になる場合は、必要に応じてデータを提供する関数を用意することで、遅延レンダリングを使用できます。この関数は、ドロップ操作またはペースト操作の受け取り側が遅延された形式でデータを要求する場合にのみ呼び出されます。

レンダリング関数を Clipboard オブジェクトに追加するには、setDataHandler() メソッドを使用します。この関数は、適切な形式でデータを返す必要があります。例えば、setDataHandler(ClipboardFormat.TEXT_FORMAT, writeText) を呼び出した場合、writeText() 関数はストリングを返す必要があります。

同じ型のデータ形式が setData() メソッドで Clipboard オブジェクトに追加された場合は、そのデータが遅延バージョンより優先されます（レンダリング関数は呼び出されません）。同じクリップボードデータが 2 回目にアクセスされるときは、レンダリング関数が再び呼び出される場合と、呼び出されない場合があります。

注意：Mac OS X では、遅延レンダリングは、カスタムデータ形式でのみ機能します。標準データ形式では、レンダリング関数は直ちに呼び出されます。

遅延レンダリング関数を使用したテキストのペースト

Flash Player 10 以降、Adobe AIR 1.0 以降

次の例では、遅延レンダリング関数の実装方法を示します。

ユーザーが「コピー」ボタンを押すと、アプリケーションによってシステムクリップボードがクリアされ、前のクリップボード操作のデータが消去されます。次に、setDataHandler() メソッドがクリップボードレンダラーとして renderData() 関数を設定します。

目的のテキストフィールドのコンテキストメニューから「ペースト」コマンドを選択すると、アプリケーションはクリップボードにアクセスして、目的のテキストを設定します。クリップボードのテキストデータ形式はストリングではなく関数を使用して設定されるので、クリップボードは renderData() 関数を呼び出します。renderData() 関数はテキストをソーステキストで返し、それがペースト先のテキストに割り当てられます。

「ペースト」ボタンをクリックする前にソーステキストを編集した場合、「コピー」ボタンをクリックした後で編集を行ったとしても、ペーストされるテキストに編集が反映されます。これは、「ペースト」ボタンがクリックされてから、レンダリング関数がソーステキストをコピーするからです（実際のアプリケーションで遅延レンダリングを使用するときは、このような問題を防ぐため、ソースデータを何らかの方法で保存または保護することが必要になる場合があります）。

Flash の例

```
package {
    import flash.desktop.Clipboard;
    import flash.desktop.ClipboardFormats;
    import flash.desktop.ClipboardTransferMode;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.text.TextFormat;
    import flash.text.TextFieldType;
    import flash.events.MouseEvent;
    import flash.events.Event;
    public class DeferredRenderingExample extends Sprite
    {
        private var sourceTextField:TextField;
        private var destination:TextField;
        private var copyText:TextField;
        public function DeferredRenderingExample():void
        {
            sourceTextField = createTextField(10, 10, 380, 90);
            sourceTextField.text = "Neque porro quisquam est qui dolorem "
                + "ipsum quia dolor sit amet, consectetur, adipisci velit.";

            copyText = createTextField(10, 110, 35, 20);
            copyText.htmlText = "<a href='#'>Copy</a>";
            copyText.addEventListener(MouseEvent.CLICK, onCopy);

            destination = createTextField(10, 145, 380, 90);
```

```
            destination.addEventListener(Event.PASTE, onPaste);
        }
        private function createTextField(x:Number, y:Number, width:Number,
                        height:Number):TextField
        {
            var newTxt:TextField = new TextField();
            newTxt.x = x;
            newTxt.y = y;
            newTxt.height = height;
            newTxt.width = width;
            newTxt.border = true;
            newTxt.multiline = true;
            newTxt.wordWrap = true;
            newTxt.type = TextFieldType.INPUT;
            addChild(newTxt);
            return newTxt;
        }
        public function onCopy(event:MouseEvent):void
        {
            Clipboard.generalClipboard.clear();
            Clipboard.generalClipboard.setDataHandler(ClipboardFormats.TEXT_FORMAT,
                            renderData);
        }
        public function onPaste(event:Event):void
        {
            sourceTextField.text =
            Clipboard.generalClipboard.getData(ClipboardFormats.TEXT_FORMAT).toString;
        }
        public function renderData():String
        {
            trace("Rendering data");
            var sourceStr:String = sourceTextField.text;
            if (sourceTextField.selectionEndIndex >
                     sourceTextField.selectionBeginIndex)
            {
                return sourceStr.substring(sourceTextField.selectionBeginIndex,
                                    sourceTextField.selectionEndIndex);
            }
            else
            {
                return sourceStr;
            }
        }
    }
}
```

Flex の例

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute" width="326" height="330"
applicationComplete="init()">
    <mx:Script>
    <![CDATA[
    import flash.desktop.Clipboard;
    import flash.desktop.ClipboardFormats;

    public function init():void
    {
        destination.addEventListener("paste", doPaste);
    }

    public function doCopy():void
    {
        Clipboard.generalClipboard.clear();
        Clipboard.generalClipboard.setDataHandler(ClipboardFormats.TEXT_FORMAT, renderData);
    }
    public function doPaste(event:Event):void
    {
        destination.text = Clipboard.generalClipboard.getData(ClipboardFormats.TEXT_FORMAT).toString;
    }

    public function renderData():String{
        trace("Rendering data");
        return source.text;
    }
    ]]>
    </mx:Script>
    <mx:Label x="10" y="10" text="Source"/>
    <mx:TextArea id="source" x="10" y="36" width="300" height="100">
        <mx:text>Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci
velit.</mx:text>
    </mx:TextArea>
    <mx:Label x="10" y="181" text="Destination"/>
    <mx:TextArea id="destination"  x="12" y="207" width="300" height="100"/>
    <mx:Button click="doCopy();" x="91" y="156" label="Copy"/>
</mx:Application>
```

第 14 章：AIR を使用したローカル SQL データベースの操作

Adobe AIR 1.0 およびそれ以降

Adobe® AIR® にはローカル SQL データベースの作成と操作を行う機能が含まれています。Adobe AIR ランタイムには、オープンソースの SQLite データベースシステムを使用する、多くの標準 SQL 機能をサポートする SQL データベースエンジンが含まれています。ローカル SQL データベースは永続的なローカルデータの格納に使用できます。例えば、ローカル SQL データベースは、アプリケーションデータ、アプリケーションユーザー設定、ドキュメント、またはアプリケーションでローカルに保存する他の任意のタイプのデータに使用できます。

ローカル SQL データベースについて

Adobe AIR 1.0 およびそれ以降

SQL データベースの使用に関する簡単な説明およびコード例については、Adobe Developer Connection で次のクイックスタートの記事を参照してください。

- ローカル SQL データベースの非同期操作
- ローカル SQL データベースの同期操作
- 暗号化されたデータベースの使用

Adobe AIR には、ランタイム内で実行される SQL ベースのリレーショナルデータベースエンジンが含まれています。データベースのデータは、AIR アプリケーションが実行されているコンピューター上（コンピューターのハードディスク上など）のデータベースファイルにローカルで格納されます。データベースの実行やデータファイルの格納はローカルで行われるため、ネットワーク接続が利用可能かどうかに関係なく AIR アプリケーションでデータベースを使用できます。したがって、ランタイムのローカル SQL データベースエンジンは、永続的なローカルアプリケーションデータを格納するための便利なメカニズムとして利用できます。SQL やリレーショナルデータベースの使用経験がある場合には特に便利です。

ローカル SQL データベースの用途

Adobe AIR 1.0 およびそれ以降

AIR のローカル SQL データベース機能は、ユーザーのローカルコンピューターにアプリケーションデータを格納するあらゆる用途に使用できます。Adobe AIR には、データをローカルに格納するためのメカニズムが複数含まれており、それぞれに異なる利点があります。AIR アプリケーションでのローカル SQL データベースの用途としては、例えば次のようなものが考えられます。

- データ指向アプリケーション（アドレス帳など）では、データベースを使用してメインアプリケーションデータを格納できます。
- ユーザーがドキュメントを作成して保存したり共有したりするドキュメント指向アプリケーションでは、各ドキュメントをデータベースファイルとしてユーザーが指定した場所に保存できます（ただし、データベースが暗号化されていない限り、任意の AIR アプリケーションでデータベースファイルを開くことができます。潜在的に機密性の高いドキュメントについては、暗号化することをお勧めします）。

- ネットワーク対応アプリケーションでは、アプリケーションデータのローカルキャッシュを格納したり、ネットワーク接続を利用できない場合にデータを一時的に格納したりするためにデータベースを使用できます。ローカルデータベースをネットワークデータストアと同期させるためのメカニズムを作成することもできます。
- すべてのアプリケーションで、個々のユーザーのアプリケーション設定（ユーザーオプションなど）やアプリケーション情報（ウィンドウのサイズや位置など）を格納するためにデータベースを使用できます。

関連項目

Christophe Coenraets：Employee Directory on AIR for Android

Raymond Camden：jQuery and AIR - Moving from web page to application

AIR データベースとデータベースファイルについて

Adobe AIR 1.0 およびそれ以降

個々の Adobe AIR ローカル SQL データベースは、コンピューターのファイルシステム内の 1 つのファイルとして格納されます。ランタイムには、データベースファイルの作成および構成や、データベースファイルのデータの操作および取得を管理する SQL データベースエンジンが含まれています。データベースのデータをファイルシステムのどこにどのように格納するかをランタイムが指定することはありません。各データベースはそれぞれがすべて 1 つのファイルに格納されます。データベースファイルをファイルシステムのどこに格納するかはユーザーが指定します。1 つの AIR アプリケーションで、独立した 1 つ以上のデータベース（データベースファイル）にアクセスできます。各データベースはファイルシステム上の 1 つのファイルとして格納されるため、アプリケーションの設計やオペレーティングシステムのファイルアクセスの制約によって必要とされる場所にデータベースを配置できます。各ユーザーが自分のデータのためにそれぞれ異なるデータベースファイルを使用することも、1 つのコンピューターのすべてのアプリケーションユーザーが 1 つのデータベースファイルにアクセスしてデータを共有することもできます。データは 1 つのコンピューターでローカルに格納されるため、別のコンピューターのユーザーとは自動的に共有されません。ローカル SQL データベースエンジンには、リモートデータベースやサーバーベースのデータベースに対して SQL ステートメントを実行する機能は用意されていません。

リレーショナルデータベースについて

Adobe AIR 1.0 およびそれ以降

リレーショナルデータベースは、データをコンピューターに格納（および取得）するためのメカニズムです。リレーショナルデータベースのデータはテーブルにまとめられます。テーブルの行はレコードまたは項目を表し、テーブルの列（「フィールド」とも呼ばれます）は各レコードを個別の値に分割します。例えば、アドレス帳アプリケーションに「友人」テーブルが含まれていた場合、テーブルの各行はデータベースに格納されている 1 人の友人を表し、テーブルの列は名前、姓、誕生日などのデータを表します。テーブルの各友人の行に対して、各列の値が個別に格納されます。

リレーショナルデータベースは、項目が別の種類の項目に関連付けられる複雑なデータを格納できるように作られています。リレーショナルデータベースでは、一対多の関係を持つ（1 つのレコードを別の種類の複数のレコードに関連付けることができる）データをすべて別のテーブルに分割する必要があります。例えば、アドレス帳アプリケーションで各友人について複数の電話番号を格納する場合、それは一対多の関係になります。この場合は、「友人」テーブルに各友人のすべての個人情報を格納し、別の「電話番号」テーブルにすべての友人のすべての電話番号を格納します。

各テーブルでは、友人や電話番号のデータを格納するほかに、この 2 つのテーブルの関係を追跡する（個々の友人のレコードをその電話番号に対応させる）ためのデータも必要になります。このデータを主キーと呼びます。主キーとは、テーブルの各行をそのテーブルの他の行から区別する一意の識別子です。主キーには、「自然キー」を使用することができます。自然キーとは、テーブルの各レコードを自然に区別するデータ項目です。「友人」テーブルに同じ誕生日を持つ友人が含まれていないことがわかっている場合は、誕生日の列を「友人」テーブルの主キー（自然キー）として使用できます。自然キーがない場合は、「友人 ID」などの主キー列を別途作成します。これは、アプリケーションで行を区別するために使用する人工的な値です。

主キーを使用して複数のテーブルの間に関係を設定することができます。例えば、各行（各友人）の一意の番号を含む「友人 ID」列が「友人」テーブルにある場合、「友人」テーブルに関連付けられる「電話番号」テーブルは、電話番号が属する友人の「友人 ID」を含む列と、実際の電話番号を含む列の、2 つの列で構成できます。これにより、どんなにたくさんの電話番号を持つ友人がいたとしても、すべての電話番号を「電話番号」テーブルに格納し、「友人 ID」の主キーを使用してその友人に関連付けることができます。あるテーブルの主キーが、関連テーブルでレコード間の関連を指定するために使用されている場合、その関連テーブルの値を外部キーと呼びます。AIR のローカルデータベースエンジンでは、多くのデータベースのように外部キー制約（挿入されたり更新されたりした外部キーの値に対応する行が主キーテーブルにあるかどうかを自動的に確認する制約）を作成することはできませんが、外部キー関係はリレーショナルデータベースの構造の重要な部分です。データベースのテーブルの間に関係を作成するときには外部キーを使用する必要があります。

SQL について

Adobe AIR 1.0 およびそれ以降

構造化照会言語（SQL）は、リレーショナルデータベースでデータを操作したり取得したりするために使用されます。SQL は、手続き型言語ではなく記述言語です。SQL ステートメントでは、データの取得方法をコンピューターに指示するのではなく、必要なデータのセットを記述します。データの取得方法はデータベースエンジンによって決定されます。

SQL 言語は、米国規格協会（ANSI）によって標準化されています。Adobe AIR のローカル SQL データベースは、SQL-92 標準のほとんどをサポートしています。

Adobe AIR でサポートされる SQL 言語について詳しくは、334 ページの「ローカルデータベースでの SQL サポート」を参照してください。

SQL データベースクラスについて

Adobe AIR 1.0 およびそれ以降

JavaScript でローカル SQL データベースを操作するには、以下のクラスのインスタンスを使用します（これらのクラスに対して air.* エイリアスを使用するには、HTML ドキュメントに AIRAliases.js ファイルを読み込む必要があります）。

クラス	説明
air.SQLConnection	データベース（データベースファイル）を作成したり開いたりするための手段や、データベースレベルの操作を実行したりデータベーストランザクションを制御したりするためのメソッドを提供します。
air.SQLStatement	データベースに対して実行される 1 つの SQL ステートメント（1 つのクエリまたはコマンド）を表します。ステートメントテキストの定義とパラメーター値の設定が含まれます。
air.SQLResult	ステートメントの実行に関する情報や実行の結果（SELECT ステートメントの結果行、UPDATE ステートメントや DELETE ステートメントの影響を受ける行の数など）を取得するための方法を提供します。

データベースの構造を記述するスキーマの情報を取得するには、以下のクラスを使用します。

クラス	説明
air.SQLSchemaResult	SQLConnection.loadSchema() メソッドの呼び出しによって生成されるデータベーススキーマの結果のコンテナとして機能します。
air.SQLTableSchema	データベース内の 1 つのテーブルを記述する情報を提供します。
air.SQLViewSchema	データベース内の 1 つのビューを記述する情報を提供します。
air.SQLIndexSchema	データベース内のテーブルまたはビューの 1 つの列を記述する情報を提供します。
air.SQLTriggerSchema	データベース内の 1 つのトリガーを記述する情報を提供します。

以下のクラスは、SQLConnection クラスで使用される定数を提供します。

クラス	説明
air.SQLMode	SQLConnection.open() メソッドと SQLConnection.openAsync() メソッドの openMode パラメーターで使用できる値を表す一連の定数を定義します。
air.SQLColumnNameStyle	SQLConnection.columnNameStyle プロパティで使用できる値を表す一連の定数を定義します。
air.SQLTransactionLockType	SQLConnection.begin() メソッドのオプションパラメーターで使用できる値を表す一連の定数を定義します。
air.SQLCollationType	SQLColumnSchema.defaultCollationType プロパティと SQLColumnSchema() コンストラクターの defaultCollationType パラメーターで使用できる値を表す一連の定数を定義します。

また、以下のクラスは、使用するイベント（およびサポート定数）を表します。

クラス	説明
air.SQLEvent	SQLConnection または SQLStatement のインスタンスで操作が正常に実行された場合に送出されるイベントを定義します。SQLEvent クラスには、各操作に関連付けられたイベント型定数が定義されています。
air.SQLErrorEvent	SQLConnection または SQLStatement のインスタンスで操作がエラーになった場合に送出されるイベントを定義します。
air.SQLUpdateEvent	INSERT、UPDATE、DELETE のいずれかの SQL ステートメントが実行された結果、接続先データベースのテーブルデータが変更された場合に、SQLConnection インスタンスで送出されるイベントを定義します。

さらに、以下のクラスは、データベース操作のエラーに関する情報を提供します。

クラス	説明
air.SQLError	データベース操作のエラーに関する情報（実行された操作や失敗の原因など）を提供します。
air.SQLErrorOperation	SQLError クラスの operation プロパティで使用できる値を表す一連の定数を定義します。このプロパティは、エラーになったデータベース操作を示します。

同期実行モードと非同期実行モードについて

Adobe AIR 1.0 およびそれ以降

ローカル SQL データベースを操作するためのコードを記述する際には、非同期実行モードまたは同期実行モードのどちらの実行モードでデータベース操作を実行するかを指定します。コード例では通常、各操作の実行方法が両方の方法で示されているため、ニーズに合う方の例を使用できます。

非同期実行モードでは、ランタイムに命令を与えると、要求した操作が完了または失敗したときにイベントが送出されます。まず、データベースエンジンに操作を実行するように命令します。データベースエンジンの処理はバックグラウンドで行われ、アプリケーションは引き続き実行されます。操作が完了（または失敗）すると、データベースエンジンがイベントを送出します。そのイベントによってコードがトリガーされて、それに続く操作が実行されます。このアプローチには、データベース操作がバックグラウンドで実行されている間もメインのアプリケーションコードの実行が継続されるという大きな利点があります。データベースの操作に時間がかかったとしても、アプリケーションは引き続き実行されるため、最も重要な点として、画面が動かなくなったりしてユーザーの操作が妨げられることがありません。その一方で、非同期操作のコードは他のコードに比べて記述が難しくなることがあります。一般的には、依存する複数の操作を様々なイベントリスナーメソッドに分割しなければならない場合に記述が難しくなります。

概念的には、操作をステップの 1 つのシーケンス（一連の同期操作）として記述する方が、複数のイベントリスナーメソッドに分割された一連の操作として記述するより簡単です。Adobe AIR では、データベース操作を非同期に実行するほかに、同期的に実行することもできます。同期実行モードでは、操作はバックグラウンドではなく、他のすべてのアプリケーショ

ンコードと同じ実行シーケンスで実行されます。データベースエンジンに操作を実行するように命令すると、その時点でコードが一時停止して、その間にデータベースエンジンの処理が行われます。操作が完了すると、コードの次の行から実行が再開されます。

操作を非同期に実行するか同期的に実行するかは、SQLConnection のレベルで設定します。1 つのデータベース接続を使用して、操作やステートメントの一部を同期的に実行し、その他を非同期に実行することはできません。SQLConnection で同期実行モードと非同期実行モードのどちらを使用するかは、データベースを開くための SQLConnection メソッドを呼び出すことによって指定します。SQLConnection.open() を呼び出すとその接続で同期実行モードが使用され、SQLConnection.openAsync() を呼び出すと非同期実行モードが使用されます。open() または openAsync() を使用して SQLConnection インスタンスをデータベースに接続すると、実行モードが同期または非同期のいずれかに固定されます。実行モードを変更するには、データベースへの接続をいったん閉じてもう一度開く必要があります。

各実行モードにはそれぞれ利点があります。ほとんどの面は似ていますが、各モードを使用するときに頭に入れておく必要がある違いがいくつかあります。これらのトピックの詳細と、各モードを使用する際のヒントについては、226 ページの「同期および非同期のデータベース操作の使用」を参照してください。

データベースの作成と変更

Adobe AIR 1.0 およびそれ以降

アプリケーションでデータの追加や取得ができるようになる前に、アプリケーションがアクセスできるデータベースにテーブルが定義されている必要があります。ここで説明するのは、データベース作成タスクと、データベース内にデータ構造を作成するタスクです。これらのタスクはデータの挿入や取得よりも使用回数は少ないですが、ほとんどのアプリケーションに必須のものです。

関連項目

Mind the Flex：Updating an existing AIR database

データベースの作成

Adobe AIR 1.0 およびそれ以降

データベースファイルを作成するには、まず SQLConnection インスタンスを作成します。次に、SQLConnection インスタンスの open() メソッドを呼び出して接続を同期実行モードで開くか、openAsync() メソッドを呼び出して非同期実行モードで開きます。open() メソッドと openAsync() メソッドは、データベースへの接続を開くために使用されます。open() メソッドまたは openAsync() メソッドで、存在しないファイルの場所を参照する File インスタンスを reference パラメーター（最初のパラメーター）に渡すと、その場所にデータベースファイルが作成され、新たに作成されたデータベースへの接続が開かれます。

データベースを作成する際には、open() メソッドを呼び出すか openAsync() メソッドを呼び出すかに関係なく、任意のファイル名拡張子を持つ任意の有効なファイル名をデータベースファイル名として使用できます。reference パラメーターに null を渡して open() メソッドや openAsync() メソッドを呼び出すと、ディスク上のデータベースファイルではなく新しいインメモリデータベースが作成されます。

次のコードリストは、非同期実行モードを使用してデータベースファイル（新しいデータベース）を作成する手順を示しています。この例では、データベースファイルが「DBSample.db」というファイル名で 147 ページの「アプリケーション記憶領域ディレクトリの参照」に保存されます。

```
// Include AIRAliases.js to use air.* shortcuts

var conn = new air.SQLConnection();

conn.addEventListener(air.SQLEvent.OPEN, openHandler);
conn.addEventListener(air.SQLErrorEvent.ERROR, errorHandler);

// The database file is in the application storage directory
var folder = air.File.applicationStorageDirectory;
var dbFile = folder.resolvePath("DBSample.db");

conn.openAsync(dbFile);

function openHandler(event)
{
    air.trace("the database was created successfully");
}

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

注意：File クラスでは特定のネイティブファイルパスを参照できますが、その場合、アプリケーションが複数のプラットフォームにわたって機能しなくなる可能性があります。例えば、C:¥Documents and Settings¥joe¥test.db というパスは、Windows 上でのみ機能します。このような理由から、File.applicationStorageDirectory などの File クラスの静的プロパティおよび resolvePath() メソッド（前の例で示したように）を使用することをお勧めします。詳しくは、145 ページの「File オブジェクトのパス」を参照してください。

操作を同期的に実行する場合は、SQLConnection インスタンスでデータベース接続を開くときに open() メソッドを呼び出します。次の例は、操作を同期的に実行する SQLConnection インスタンスを作成して開く方法を示しています。

```
// Include AIRAliases.js to use air.* shortcuts

var conn = new air.SQLConnection();

// The database file is in the application storage directory
var folder = air.File.applicationStorageDirectory;
var dbFile = folder.resolvePath("DBSample.db");

try
{
    conn.open(dbFile);
    air.trace("the database was created successfully");
}
catch (error)
{
    air.trace("Error message:", error.message);
    air.trace("Details:", error.details);
}
```

データベーステーブルの作成

Adobe AIR 1.0 およびそれ以降

データベースのテーブルを作成する際には、SELECT や INSERT などの SQL ステートメントを実行するときと同じ手順を使用して、そのデータベースに対して SQL ステートメントを実行します。テーブルを作成するには、新しいテーブルの列と制約の定義を含む CREATE TABLE ステートメントを使用します。SQL ステートメントの実行について詳しくは、210 ページの「SQL ステートメントの操作」を参照してください。

次の例では、非同期実行モードを使用して既存のデータベースファイルに「employees」という名前のテーブルを作成しています。このコードでは、conn という名前の SQLConnection インスタンスが既に作成されてデータベースに接続されていることを前提としています。

```
// Include AIRAliases.js to use air.* shortcuts

// ... create and open the SQLConnection instance named conn ...

var createStmt = new air.SQLStatement();
createStmt.sqlConnection = conn;

var sql =
    "CREATE TABLE IF NOT EXISTS employees (" +
    "   empId INTEGER PRIMARY KEY AUTOINCREMENT, " +
    "   firstName TEXT, " +
    "   lastName TEXT, " +
    "   salary NUMERIC CHECK (salary > 0)" +
    ")";
createStmt.text = sql;

createStmt.addEventListener(air.SQLEvent.RESULT, createResult);
createStmt.addEventListener(air.SQLErrorEvent.ERROR, createError);

createStmt.execute();

function createResult(event)
{
    air.trace("Table created");
}

function createError(event)
{
    air.trace("Error message:", event.error.message);
    air.trace("Details:", event.error.details);
}
```

次の例では、同期実行モードを使用して既存のデータベースファイルに「employees」という名前のテーブルを作成しています。このコードでは、conn という名前の SQLConnection インスタンスが既に作成されてデータベースに接続されていることを前提としています。

```
// Include AIRAliases.js to use air.* shortcuts

// ... create and open the SQLConnection instance named conn ...

var createStmt = new air.SQLStatement();
createStmt.sqlConnection = conn;

var sql =
    "CREATE TABLE IF NOT EXISTS employees (" +
    "   empId INTEGER PRIMARY KEY AUTOINCREMENT, " +
    "   firstName TEXT, " +
    "   lastName TEXT, " +
    "   salary NUMERIC CHECK (salary > 0)" +
    ")";
createStmt.text = sql;

try
{
    createStmt.execute();
    air.trace("Table created");
}
catch (error)
{
    air.trace("Error message:", error.message);
    air.trace("Details:", error.details);
}
```

SQL データベースのデータの操作

Adobe AIR 1.0 およびそれ以降

ローカル SQL データベースを操作するときに実行する一般的なタスクがいくつか存在します。タスクの内容は、データベースへの接続、データベース内のテーブルへのデータ追加、テーブルからのデータ取得などです。これらのタスクを行うときには心がけておくべきいくつかの項目があります。それは例えば、データ型の操作やエラー処理などです。

一般的なタスクよりも実行頻度が少ないデータベースタスクもいくつかありますが、一般的なタスクよりも前に実行が必要なタスクがほとんどです。例えば、データベースに接続してテーブルからデータを取得できるようになる前には、データベースを作成し、そのデータベースの中にテーブル構造を作成する必要があります。これらの実行頻度の少ない初期設定タスクについては、205 ページの「データベースの作成と変更」を参照してください。

データベース操作を非同期で実行すると選択することもできます。非同期操作では、データベースエンジンがバックグラウンドで実行され、イベントの送出によって操作の成功または失敗が通知されます。また、これらの操作を同期的に実行することもできます。同期の場合、データベース操作は 1 つずつ実行され、アプリケーション全体（画面への更新を含む）はある操作が完了するのを待ってから別のコードを実行します。非同期実行モードおよび同期実行モードでの操作について詳しくは、226 ページの「同期および非同期のデータベース操作の使用」を参照してください。

データベースへの接続

Adobe AIR 1.0 およびそれ以降

データベース操作を実行するには、まずデータベースファイルへの接続を開く必要があります。SQLConnection インスタンスは 1 つ以上のデータベースへの接続を表すために使用されます。SQLConnection インスタンスを使用して接続した最初のデータベースを「メイン」データベースと呼びます。このデータベースには、open() メソッド（同期実行モードの場合）または openAsync() メソッド（非同期実行モードの場合）を使用して接続します。

非同期 openAsync() 操作を使用してデータベースを開く場合は、openAsync() 操作が完了したら通知されるように、SQLConnection インスタンスの open イベントに登録します。また、操作が失敗したかどうかを確認できるように、SQLConnection インスタンスの error イベントに登録します。

次の例は、既存のデータベースファイルを非同期実行用に開く方法を示しています。このデータベースファイルは「DBSample.db」という名前で、ユーザーの 147 ページの「アプリケーション記憶領域ディレクトリの参照」に保存されています。

```
// Include AIRAliases.js to use air.* shortcuts

var conn = new air.SQLConnection();

conn.addEventListener(air.SQLEvent.OPEN, openHandler);
conn.addEventListener(air.SQLErrorEvent.ERROR, errorHandler);

// The database file is in the application storage directory
var folder = air.File.applicationStorageDirectory;
var dbFile = folder.resolvePath("DBSample.db");

conn.openAsync(dbFile, air.SQLMode.UPDATE);

function openHandler(event)
{
    air.trace("the database opened successfully");
}

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

次の例は、既存のデータベースファイルを同期実行用に開く方法を示しています。このデータベースファイルは「DBSample.db」という名前で、ユーザーの 147 ページの「アプリケーション記憶領域ディレクトリの参照」に保存されています。

```
// Include AIRAliases.js to use air.* shortcuts

var conn = new air.SQLConnection();

// The database file is in the application storage directory
var folder = air.File.applicationStorageDirectory;
var dbFile = folder.resolvePath("DBSample.db");

try
{
    conn.open(dbFile, air.SQLMode.UPDATE);
    air.trace("the database opened successfully");
}
catch (error)
{
    air.trace("Error message:", error.message);
    air.trace("Details:", error.details);
}
```

非同期実行の例の openAsync() メソッドの呼び出しと、同期実行の例の open() メソッドの呼び出しで、2 番目の引数が定数 SQLMode.UPDATE になっていることに注意してください。2 番目のパラメーター（openMode）に SQLMode.UPDATE を指定すると、指定したファイルが存在しない場合にエラーが送出されます。openMode パラメーターに SQLMode.CREATE を渡すと（または openMode パラメーターをオフのままにすると）、指定したファイルが存在しない場合にデータベースファイルが作成されます。ただし、ファイルが存在する場合は、ファイルが開かれます。この動作は、SQLMode.Update を使用し

た場合と同じです。そのほか、openMode パラメーターに SQLMode.READ を指定して既存のデータベースを読み取り専用モードで開くこともできます。その場合は、データベースからデータを取得することはできますが、データを追加したり、削除したり、変更したりすることはできません。

SQL ステートメントの操作

Adobe AIR 1.0 およびそれ以降

個々の SQL ステートメント（クエリまたはコマンド）は、ランタイム内で SQLStatement オブジェクトとして表されます。SQL ステートメントを作成して実行するには次の手順に従ってください。

SQLStatement インスタンスを作成します。
SQLStatement オブジェクトは、アプリケーション内で SQL ステートメントを表します。

```
var selectData = new air.SQLStatement();
```

クエリを実行するデータベースを指定します。
そのためには、SQLStatement オブジェクトの sqlConnection プロパティを、目的のデータベースに接続されている SQLConnection インスタンスに設定します。

```
// A SQLConnection named "conn" has been created previously
selectData.sqlConnection = conn;
```

実際の SQL ステートメントを指定します。
ステートメントテキストを String として作成し、SQLStatement インスタンスの text プロパティに割り当てます。

```
selectData.text = "SELECT col1, col2 FROM my_table WHERE col1 = :param1";
```

実行操作の結果を処理するための関数を定義します（非同期実行モードのみ）。
addEventListener() メソッドを使用して、SQLStatement インスタンスの result イベントと error イベントのリスナーとして関数を登録します。

```
// using listener methods and addEventListener()

selectData.addEventListener(air.SQLEvent.RESULT, resultHandler);
selectData.addEventListener(air.SQLErrorEvent.ERROR, errorHandler);

function resultHandler(event)
{
    // do something after the statement execution succeeds
}

function errorHandler(event)
{
    // do something after the statement execution fails
}
```

または、Responder オブジェクトを使用してリスナーメソッドを指定することもできます。その場合は、Responder インスタンスを作成してリスナーメソッドを関連付けます。

```
// using a Responder

var selectResponder = new air.Responder(onResult, onError);

function onResult(result)
{
    // do something after the statement execution succeeds
}

function onError(error)
{
    // do something after the statement execution fails
}
```

ステートメントテキストにパラメーター定義が含まれている場合は、それらのパラメーターに値を割り当てます。
パラメーター値を割り当てるには、SQLStatement インスタンスの parameters 連想配列プロパティを使用します。

```
selectData.parameters[":param1"] = 25;
```

SQL ステートメントを実行します。
SQLStatement インスタンスの execute() メソッドを呼び出します。

```
// using synchronous execution mode
// or listener methods in asynchronous execution mode
selectData.execute();
```

さらに、非同期実行モードでイベントリスナーの代わりに Responder を使用している場合は、その Responder インスタンスを execute() メソッドに渡します。

```
// using a Responder in asynchronous execution mode
selectData.execute(-1, selectResponder);
```

これらの手順の具体例については、以下のトピックを参照してください。

214 ページの「データベースからのデータの取得」

220 ページの「データの挿入」

222 ページの「データの変更または削除」

ステートメント内でのパラメーターの使用

Adobe AIR 1.0 およびそれ以降

SQL ステートメントパラメーターを使用すると、再利用可能な SQL ステートメントを作成できます。ステートメントパラメーターを使用する場合、ステートメント内の値は変わっても（INSERT ステートメントの追加される値など）、基本的なステートメントテキストは変わりません。その結果、パラメーターを使用することでパフォーマンスが向上し、アプリケーションのコーディングも容易になります。

ステートメントパラメーターについて

Adobe AIR 1.0 およびそれ以降

アプリケーションで 1 つの SQL ステートメントが、小さな変更を加えながら何度も使用されることがよくあります。例えば、ユーザーが新しい在庫品目をデータベースに追加できる在庫追跡アプリケーションでは、在庫品目をデータベースに追加するアプリケーションコードで、データを実際にデータベースに追加する SQL INSERT ステートメントが実行されますが、このステートメントは実行のたびに多少変更されます。テーブルに挿入される実際の値は、追加される在庫品目に固有の値であるため、毎回異なるからです。

このように異なる値で何度も使用される SQL ステートメントがある場合は、SQL テキスト内にリテラル値の代わりにパラメーターを含む SQL ステートメントを使用することをお勧めします。パラメーターは、ステートメントが実行されるたびに実際の値に置き換えられるステートメントテキスト内のプレースホルダーです。SQL ステートメント内でパラメーターを使用するには、SQLStatement インスタンスを通常どおりに作成し、text プロパティに割り当てる実際の SQL ステートメントで、リテラル値の代わりにパラメータープレースホルダーを使用します。各パラメーターの値は、SQLStatement インスタンスの parameters プロパティの要素の値を設定して定義します。parameters プロパティは連想配列であるため、特定の値を設定するには次のシンタックスを使用します。

```
statement.parameters[parameter_identifier] = value;
```

parameter_identifier は、名前付きパラメーターを使用する場合は文字列で、名前のないパラメーターを使用する場合は整数インデックスです。

名前付きパラメーターの使用

Adobe AIR 1.0 およびそれ以降

パラメーターは、名前付きパラメーターにすることができます。名前付きパラメーターには、パラメーター値をステートメントテキスト内のそのプレースホルダーの場所に対応させるためにデータベースで使用される特定の名前があります。パラメーター名は、「:」または「@」の文字と、それに続く名前で構成されます。以下に例を示します。

```
:itemName
@firstName
```

次のコードリストは、名前付きパラメーターの使用方法を示しています。

```
var sql =
    "INSERT INTO inventoryItems (name, productCode)" +
    "VALUES (:name, :productCode)";

var addItemStmt = new air.SQLStatement();
addItemStmt.sqlConnection = conn;
addItemStmt.text = sql;

// set parameter values
addItemStmt.parameters[":name"] = "Item name";
addItemStmt.parameters[":productCode"] = "12345";

addItemStmt.execute();
```

名前のないパラメーターの使用

Adobe AIR 1.0 およびそれ以降

名前付きパラメーターの代わりに、名前のないパラメーターを使用することもできます。名前のないパラメーターを使用する場合は、SQL ステートメント内で「?」文字を使用してパラメーターを表します。ステートメント内の順番に従って、0 から始まる数値インデックスが各パラメーターに割り当てられます。次の例では、先ほどの例が、名前のないパラメーターを使用するように変更されています。

```
var sql =
    "INSERT INTO inventoryItems (name, productCode)" +
    "VALUES (?, ?)";

var addItemStmt = new air.SQLStatement();
addItemStmt.sqlConnection = conn;
addItemStmt.text = sql;

// set parameter values
addItemStmt.parameters[0] = "Item name";
addItemStmt.parameters[1] = "12345";

addItemStmt.execute();
```

パラメーターを使用するメリット

Adobe AIR 1.0 およびそれ以降

SQL ステートメントでパラメーターを使用すると、次のようなメリットがあります。

パフォーマンスの向上 パラメーターを使用する SQLStatement インスタンスは、実行のたびに SQL テキストを動的に作成する SQLStatement インスタンスに比べて効率的に実行できます。このようにパフォーマンスが向上するのは、1 回準備したステートメントを異なるパラメーター値で何度も実行できるため、SQL ステートメントを再コンパイルする必要がないからです。

明示的なデータ型の指定 パラメーターを使用すると、SQL ステートメントの作成時には不明な値の型指定された置き換えが可能になります。データベースに渡される値の格納クラスを保証するには、パラメーターを使用するしかありません。パラメーターを使用しない場合は、すべての値が、関連付けられている列の型の類似性に基づいてテキスト表現から格納クラスに変換されます。

格納クラスと列の親和型について詳しくは、355 ページの「データ型のサポート」を参照してください。

セキュリティの強化 パラメーターを使用すると、SQL インジェクション攻撃と呼ばれる悪質なテクニックの防止に役立ちます。SQL インジェクション攻撃では、ユーザーがアクセスできる場所（データ入力フィールドなど）に SQL コードが入力されます。この場合、ユーザー入力を直接 SQL テキストに連結して SQL ステートメントを作成するアプリケーションコードでは、ユーザーが入力した SQL コードがデータベースに対して実行されます。次のリストは、ユーザー入力が SQL テキストに連結される例を示しています。**この方法は使用しないでください**。

```
// assume the variables "username" and "password"
// contain user-entered data

var sql =
    "SELECT userId " +
    "FROM users " +
    "WHERE username = '" + username + "' " +
    "    AND password = '" + password + "'";

var statement = new air.SQLStatement();
statement.text = sql;
```

ユーザーが入力した値をステートメントのテキストに連結する代わりにステートメントパラメーターを使用すると、SQL インジェクション攻撃を防止できます。この場合に SQL インジェクションが発生しないのは、パラメーター値はリテラルステートメントテキストの一部にはならず、置き換えられる値として明示的に扱われるからです。以下は、前のリストの代わりとして推奨されるリストです。

```
// assume the variables "username" and "password"
// contain user-entered data

var sql =
    "SELECT userId " +
    "FROM users " +
    "WHERE username = :username " +
    "    AND password = :password";

var statement = new air.SQLStatement();
statement.text = sql;

// set parameter values
statement.parameters[":username"] = username;
statement.parameters[":password"] = password;
```

データベースからのデータの取得

Adobe AIR 1.0 およびそれ以降

データベースからデータを取得する操作には 2 つのステップが含まれます。まず、データベースから取得するデータのセットを記述する SQL SELECT ステートメントを実行します。次に、取得したデータにアクセスして、必要に応じてアプリケーションで表示または操作します。

SELECT ステートメントの実行

Adobe AIR 1.0 およびそれ以降

データベースから既存のデータを取得するには、SQLStatement インスタンスを使用します。適切な SQL SELECT ステートメントをインスタンスの text プロパティに割り当てて、インスタンスの execute() メソッドを呼び出します。

SELECT ステートメントのシンタックスについて詳しくは、334 ページの「ローカルデータベースでの SQL サポート」を参照してください。

次の例では、非同期実行モードを使用し、SELECT ステートメントを実行して「products」という名前のテーブルからデータを取得しています。

```
// Include AIRAliases.js to use air.* shortcuts

var selectStmt = new air.SQLStatement();

// A SQLConnection named "conn" has been created previously
selectStmt.sqlConnection = conn;

selectStmt.text = "SELECT itemId, itemName, price FROM products";

selectStmt.addEventListener(air.SQLEvent.RESULT, resultHandler);
selectStmt.addEventListener(air.SQLErrorEvent.ERROR, errorHandler);

selectStmt.execute();

function resultHandler(event)
{
    var result = selectStmt.getResult();

    var numResults = result.data.length;
    for (i = 0; i < numResults; i++)
    {
        var row = result.data[i];
        var output = "itemId: " + row.itemId;
        output += "; itemName: " + row.itemName;
        output += "; price: " + row.price;
        air.trace(output);
    }
}

function errorHandler(event)
{
    // Information about the error is available in the
    // event.error property, which is an instance of
    // the SQLError class.
}
```

次の例では、同期実行モードを使用し、SELECT ステートメントを実行して「products」という名前のテーブルからデータを取得します。

```
// Include AIRAliases.js to use air.* shortcuts

var selectStmt = new air.SQLStatement();

// A SQLConnection named "conn" has been created previously
selectStmt.sqlConnection = conn;

selectStmt.text = "SELECT itemId, itemName, price FROM products";

try
{
    selectStmt.execute();

    var result = selectStmt.getResult();

    var numResults = result.data.length;
    for (i = 0; i < numResults; i++)
    {
        var row = result.data[i];
        var output = "itemId: " + row.itemId;
        output += "; itemName: " + row.itemName;
        output += "; price: " + row.price;
        air.trace(output);
    }
}
catch (error)
{
    // Information about the error is available in the
    // error variable, which is an instance of
    // the SQLError class.
}
```

非同期実行モードでは、ステートメントの実行が完了すると SQLStatement インスタンスによって result イベント（SQLEvent.RESULT）が送出されて、ステートメントが正常に実行されたことが通知されます。または、Responder オブジェクトを引数として execute() メソッドに渡した場合は、Responder オブジェクトの結果ハンドラー関数が呼び出されます。同期実行モードでは、execute() 操作が完了するまで実行が一時停止され、その後、コードの次の行から再開されます。

SELECT ステートメントの結果データへのアクセス

Adobe AIR 1.0 およびそれ以降

SELECT ステートメントの実行が完了したら、次の手順として、取得したデータにアクセスします。SELECT ステートメントの実行結果のデータを取得するには、SQLStatement オブジェクトの getResult() メソッドを呼び出します。

```
var result = selectStatement.getResult();
```

getResult() メソッドは SQLResult オブジェクトを返します。SQLResult オブジェクトの data プロパティは SELECT ステートメントの結果が格納された配列です。

```
var numResults = result.data.length;
for (var i = 0; i < numResults; i++)
{
    // row is an Object representing one row of result data
    var row = result.data[i];
}
```

SELECT の結果セットの各データ行は、data 配列に格納される Object インスタンスになります。そのオブジェクトには、結果セットの列と同じ名前のプロパティがあります。そのプロパティに、結果セットの列の値が格納されます。例えば、「itemId」、「itemName」および「price」の 3 つの列を含む結果セットを SELECT ステートメントで指定した場合は、itemId、itemName および price の 3 つのプロパティを持つ Object インスタンスが結果セットの各行について作成されて、各プロパティにそれぞれの列の値が格納されます。

次のコードリストでは、text プロパティが SELECT ステートメントの SQLStatement インスタンスを定義しています。このステートメントは、employees というテーブルのすべての行の firstName 列と lastName 列の値を含む行を取得します。この例では非同期実行モードを使用しています。実行が完了したら、selectResult() メソッドを呼び出します。その後、SQLStatement.getResult() を使用して結果行にアクセスして、trace() メソッドを使用して表示します。このリストでは、conn という名前の SQLConnection インスタンスが既に作成されてデータベースに接続されていることと、「employees」テーブルが既に作成されてデータが設定されていることを前提としています。

```
// Include AIRAliases.js to use air.* shortcuts

// ... create and open the SQLConnection instance named conn ...

// create the SQL statement
var selectStmt = new air.SQLStatement();
selectStmt.sqlConnection = conn;

// define the SQL text
var sql =
    "SELECT firstName, lastName " +
    "FROM employees";
selectStmt.text = sql;

// register listeners for the result and error events
selectStmt.addEventListener(air.SQLEvent.RESULT, selectResult);
selectStmt.addEventListener(air.SQLErrorEvent.ERROR, selectError);

// execute the statement
selectStmt.execute();

function selectResult(event)
{
    // access the result data
    var result = selectStmt.getResult();

    var numRows = result.data.length;
    for (i = 0; i < numRows; i++)
    {
        var output = "";
        for (columnName in result.data[i])
        {
            output += columnName + ": " + result.data[i][columnName] + "; ";
        }
        air.trace("row[" + i.toString() + "]\t", output);
    }
}

function selectError(event)
{
    air.trace("Error message:", event.error.message);
    air.trace("Details:", event.error.details);
}
```

次のコードリストで使用されているのは、先ほどの例と同じテクニックです。ただし、今度は同期実行モードを使用しています。この例では、text プロパティが SELECT ステートメントの SQLStatement インスタンスを定義しています。このステートメントは、employees というテーブルのすべての行の firstName 列と lastName 列の値を含む行を取得します。SQLStatement.getResult() を使用して結果行にアクセスして、trace() メソッドを使用して表示します。このリストでは、conn という名前の SQLConnection インスタンスが既に作成されてデータベースに接続されていることと、「employees」テーブルが既に作成されてデータが設定されていることを前提としています。

```
// Include AIRAliases.js to use air.* shortcuts

// ... create and open the SQLConnection instance named conn ...

// create the SQL statement
var selectStmt = new air.SQLStatement();
selectStmt.sqlConnection = conn;

// define the SQL text
var sql =
    "SELECT firstName, lastName " +
    "FROM employees";
selectStmt.text = sql;

try
{
    // execute the statement
    selectStmt.execute();

    // access the result data
    var result = selectStmt.getResult();

    var numRows = result.data.length;
    for (i = 0; i < numRows; i++)
    {
        var output = "";
        for (columnName in result.data[i])
        {
            output += columnName + ": " + result.data[i][columnName] + "; ";
        }
        air.trace("row[" + i.toString() + "]\t", output);
    }
}
catch (error)
{
    air.trace("Error message:", error.message);
    air.trace("Details:", error.details);
}
```

SELECT の結果データのデータ型の定義

Adobe AIR 1.0 およびそれ以降

既定では、SELECT ステートメントによって返される各行は Object インスタンスとして作成されます。この Object インスタンスには結果セットの列名と同じ名前のプロパティがあり、各列の値がプロパティの値として格納されます。しかし、SQL SELECT ステートメントを実行する前に SQLStatement インスタンスの itemClass プロパティを設定してクラスを指定することもできます。itemClass プロパティを設定すると、SELECT ステートメントによって返される各行が、指定したクラスのインスタンスとして作成されます。結果の列の値は、SELECT の結果セットの列名を itemClass クラスのプロパティ名に対応付けることによってプロパティ値に割り当てられます。

itemClass プロパティとして割り当てるクラスには、パラメーターを必要としないコンストラクターと、SELECT ステートメントによって返される各列に対応するプロパティが必要です。SELECT のリストの列に対応するプロパティ名が itemClass クラスにない場合はエラーと見なされます。

SELECT の結果の分割

Adobe AIR 1.0 およびそれ以降

既定では、SELECT ステートメントを実行すると結果セットのすべての行が一度に取得されます。ステートメントの完了後は、取得したデータを何らかの形で処理するのが一般的ですが（オブジェクトを作成する、データを画面に表示するなど）、ステートメントによって返される行の数が多い場合にすべてのデータを一度に処理すると、コンピューターに負荷がかかり、ユーザーインターフェイスの再描画が行われなくなる可能性があります。

このような場合にアプリケーションの体感的なパフォーマンスを向上させるには、一度に取得する結果行の数を指定します。これにより、最初の結果データがすぐに返されるようになるほか、結果行を複数のセットに分割して、各セットが処理されるたびにユーザーインターフェイスが更新されるようにすることができます。ただし、このテクニックを使用できるのは非同期実行モードを使用している場合だけです。

SELECT の結果を分割して取得するには、SQLStatement.execute() メソッドの最初のパラメーター（prefetch パラメーター）の値を指定します。prefetch パラメーターは、ステートメントが初めて実行されたときに取得する行数を表します。SQLStatement インスタンスの execute() メソッドを呼び出すときに prefetch パラメーター値を指定すると、指定した行数のみが取得されます。

```
// Include AIRAliases.js to use air.* shortcuts
var stmt = new air.SQLStatement();
stmt.sqlConnection = conn;

stmt.text = "SELECT ...";

stmt.addEventListener(air.SQLEvent.RESULT, selectResult);

stmt.execute(20); // only the first 20 rows (or fewer) are returned
```

結果行の最初のセットが利用可能になると、result イベントが送出されます。結果となる SQLResult インスタンスの data プロパティにデータの行が格納され、取得する結果行がまだあるかどうかが complete プロパティによって示されます。さらに結果行を取得するには、SQLStatement インスタンスの next() メソッドを呼び出します。next() メソッドの最初のパラメーターは、execute() メソッドと同様に、次に result イベントが送出されたときに取得する行数を表します。

```
function selectResult(event)
{
    var result = stmt.getResult();
    if (result.data != null)
    {
        // ... loop through the rows or perform other processing ...

        if (!result.complete)
        {
            stmt.next(20); // retrieve the next 20 rows
        }
        else
        {
            stmt.removeEventListener(air.SQLEvent.RESULT, selectResult);
        }
    }
}
```

result イベントは、next() メソッドによって結果行の続きのセットが返されるたびに送出されます。したがって、すべての行が取得されるまで同じリスナー関数を使用して（next() の呼び出しの）結果の処理を続けることができます。

詳しくは、SQLStatement.execute() メソッド（の prefetch パラメーター）の説明と SQLStatement.next() メソッドの説明を参照してください。

データの挿入

Adobe AIR 1.0 およびそれ以降

データベースにデータを追加する際には SQL INSERT ステートメントを実行します。ステートメントの実行が完了したら、新たに挿入された行の主キーにアクセスできます（データベースによってキーが生成された場合）。

INSERT ステートメントの実行

Adobe AIR 1.0 およびそれ以降

データベースのテーブルにデータを追加するには、text プロパティが SQL INSERT ステートメントの SQLStatement インスタンスを作成して実行します。

次の例では、既に存在している employees テーブルに 1 行のデータを追加する SQLStatement インスタンスが使用されています。この例は、非同期実行モードを使用してデータを挿入する方法を示しています。このリストでは、conn という名前の SQLConnection インスタンスが既に作成されてデータベースに接続されていることと、「employees」テーブルが既に作成されていることを前提としています。

```
// Include AIRAliases.js to use air.* shortcuts

// ... create and open the SQLConnection instance named conn ...

// create the SQL statement
var insertStmt = new air.SQLStatement();
insertStmt.sqlConnection = conn;

// define the SQL text
var sql =
    "INSERT INTO employees (firstName, lastName, salary) " +
    "VALUES ('Bob', 'Smith', 8000)";
insertStmt.text = sql;

// register listeners for the result and failure (status) events
insertStmt.addEventListener(air.SQLEvent.RESULT, insertResult);
insertStmt.addEventListener(air.SQLErrorEvent.ERROR, insertError);

// execute the statement
insertStmt.execute();

function insertResult(event)
{
    air.trace("INSERT statement succeeded");
}

function insertError(event)
{
    air.trace("Error message:", event.error.message);
    air.trace("Details:", event.error.details);
}
```

次の例では、同期実行モードを使用して、既に存在する employees テーブルに 1 行のデータを追加しています。このリストでは、conn という名前の SQLConnection インスタンスが既に作成されてデータベースに接続されていることと、「employees」テーブルが既に作成されていることを前提としています。

```
// Include AIRAliases.js to use air.* shortcuts

// ... create and open the SQLConnection instance named conn ...

// create the SQL statement
var insertStmt = new air.SQLStatement();
insertStmt.sqlConnection = conn;

// define the SQL text
var sql =
    "INSERT INTO employees (firstName, lastName, salary) " +
    "VALUES ('Bob', 'Smith', 8000)";
insertStmt.text = sql;

try
{
    // execute the statement
    insertStmt.execute();

    air.trace("INSERT statement succeeded");
}
catch (error)
{
    air.trace("Error message:", error.message);
    air.trace("Details:", error.details);
}
```

挿入された行の（データベースによって生成される）主キーの取得

Adobe AIR 1.0 およびそれ以降

テーブルにデータの行を挿入した後に、新たに挿入された行の（データベースによって生成される）主キーや行識別子の値がコードで必要になることがよくあります。例えば、テーブルに行を挿入した後に関連テーブルの行を追加する場合は、主キー値を関連テーブルの外部キーとして挿入する必要があります。新たに挿入された行の主キーを取得するには、ステートメントの実行に関連した SQLResult オブジェクトを使用します。これは、SELECT ステートメントの実行後に結果データにアクセスするために使用するのと同じオブジェクトです。INSERT ステートメントの実行が完了すると、他の SQL ステートメントの場合と同様に SQLResult インスタンスが作成されます。その SQLResult インスタンスにアクセスするには、SQLStatement オブジェクトの getResult() メソッドを呼び出します（イベントリスナーを使用している場合と同期実行モードを使用している場合）。非同期実行モードを使用していて、execute() の呼び出しで Responder インスタンスを渡した場合は、その結果ハンドラー関数に SQLResult インスタンスが引数として渡されます。いずれの場合も、その SQLResult インスタンスの lastInsertRowID プロパティに、直前に挿入された行の行識別子が格納されています（実行された SQL ステートメントが INSERT ステートメントの場合）。

次の例は、挿入された行の主キーに非同期実行モードでアクセスする方法を示しています。

```
insertStmt.text = "INSERT INTO ...";

insertStmt.addEventListener(air.SQLEvent.RESULT, resultHandler);

insertStmt.execute();

function resultHandler(event)
{
    // get the primary key
    var result = insertStmt.getResult();

    var primaryKey = result.lastInsertRowID;
    // do something with the primary key
}
```

次の例は、挿入された行の主キーに同期実行モードでアクセスする方法を示しています。

```
insertStmt.text = "INSERT INTO ...";

try
{
    insertStmt.execute();

    // get the primary key
    var result = insertStmt.getResult();

    var primaryKey = result.lastInsertRowID;
    // do something with the primary key
}
catch (error)
{
    // respond to the error
}
```

行識別子は、テーブル定義で主キー列として指定されている列の値である場合もあれば、そうでない場合もあります。これは次のルールによって決まります。

- テーブルの定義で使用されている主キー列の類似性（列のデータ型）が INTEGER の場合は、その行に挿入された値（AUTOINCREMENT 列の場合にはランタイムによって生成された値）が lastInsertRowID プロパティに格納されます。
- テーブルの定義で複数の主キー列（複合キー）が使用されている場合や、主キー列が 1 つでも親和型 INTEGER ではない場合は、その行の行識別子の整数値が背後で生成されます。その生成された値が lastInsertRowID プロパティの値になります。
- この値は常に、直前に挿入された行の行識別子です。INSERT ステートメントがトリガーを発生させて、その結果として行が挿入された場合は、INSERT ステートメントによって作成された行ではなく、トリガーによって挿入された最後の行の行識別子が lastInsertRowID プロパティに格納されます。

これらのルールによって、INSERT コマンドの実行後に SQLResult.lastInsertRowID プロパティを通じて値を取得できる、明示的に定義された主キー列が必要な場合は、その列を INTEGER PRIMARY KEY 列として定義する必要があります。明示的な INTEGER PRIMARY KEY 列がテーブルに含まれていなくても、関連テーブルとの関係を定義するという意味では、データベースによって生成される行識別子をテーブルの主キーとして使用しても何ら問題はありません。SQL ステートメントで行識別子列の値を取得するには、特殊な列名のいずれか（ROWID、_ROWID_、または OID）を使用します。関連テーブルで外部キー列を作成して、行識別子の値を（明示的に宣言された INTEGER PRIMARY KEY 列の場合と同じように）その外部キー列の値として使用することもできます。その意味では、自然キーではなく恣意的な主キーを使用している場合に、主キー値がランタイムによって生成されても問題がなければ、2 つのテーブルの間の外部キー関係を定義するためのテーブルの主キーとして INTEGER PRIMARY KEY 列を使用しても、システム生成の行識別子を使用しても、ほとんど違いはありません。

主キーと生成された行識別子について詳しくは、334 ページの「ローカルデータベースでの SQL サポート」を参照してください。

データの変更または削除

Adobe AIR 1.0 およびそれ以降

その他のデータ操作を実行するための手順は、210 ページの「SQL ステートメントの操作」で説明している SQL SELECT ステートメントや INSERT ステートメントを実行するための手順と変わりません。SQLStatement インスタンスの text プロパティを別の SQL ステートメントに置き換えるだけです。

- テーブルの既存のデータを変更する場合は UPDATE ステートメントを使用します。
- テーブルの 1 つ以上のデータ行を削除する場合は DELETE ステートメントを使用します。

これらのステートメントについて詳しくは、334 ページの「ローカルデータベースでの SQL サポート」を参照してください。

複数のデータベースの操作

Adobe AIR 1.0 およびそれ以降

SQLConnection.attach() メソッドを使用すると、既に開いているデータベースがある SQLConnection インスタンスで追加のデータベースへの接続を開くことができます。attach() メソッドの呼び出しで name パラメーターを使用して、関連付けるデータベースに名前を付けます。その名前は、そのデータベースを操作するステートメントを記述する際に、SQL ステートメント内でテーブル名を修飾するための接頭辞として使用できます（データベース名 . テーブル名の形式を使用します）。これにより、そのテーブルがそのデータベースにあることをランタイムに対して示すことができます。

同じ SQLConnection インスタンスに接続されている複数のデータベースのテーブルを含む 1 つの SQL ステートメントを実行することができます。SQLConnection インスタンスでトランザクションを作成すると、そのトランザクションは、その SQLConnection インスタンスを使用して実行されるすべての SQL ステートメントに適用されます。関連付けられているどのデータベースに対してステートメントが実行されるかは関係ありません。

また、アプリケーションで複数の SQLConnection インスタンスを作成して、各インスタンスを 1 つ以上のデータベースに接続することもできます。ただし、同じデータベースに対して複数の接続を使用する場合は、データベーストランザクションは複数の SQLConnection インスタンスにまたがって共有されないことに注意する必要があります。その結果、複数の SQLConnection インスタンスを使用して同じデータベースファイルに接続すると、両方の接続のデータ変更が正しく適用されるという保証が得られなくなります。例えば、2 つの UPDATE ステートメントまたは DELETE ステートメントを別の SQLConnection インスタンスを通じて同じデータベースに対して実行した場合に、1 つ目の操作の実行後にアプリケーションエラーが発生すると、データベースのデータが元に戻せない中間状態のままになり、データベースの整合性（および、結果としてアプリケーション）に影響が及ぶ可能性があります。

データベースエラーの処理

Adobe AIR 1.0 およびそれ以降

一般に、データベースエラーの処理は他のランタイムエラーの処理と同様で、発生する可能性があるエラーに備えたコードを記述して、ランタイムではなくそのコードがエラーに応答するようにします。発生する可能性があるデータベースエラーは、一般的な意味において 3 つのカテゴリに分類できます。接続エラー、SQL シンタックスエラーおよび制約エラーの 3 つです。

接続エラー

Adobe AIR 1.0 およびそれ以降

ほとんどのデータベースエラーは接続エラーです。接続エラーはあらゆる操作で発生します。接続エラーを防止するための対策はありますが、データベースがアプリケーションの重要な部分である場合、接続エラーから速やかに回復するための簡単な方法はほとんどありません。

ほとんどの接続エラーは、ランタイムとオペレーティングシステム、ファイルシステムおよびデータベースファイルとのやり取りに関連しています。例えば、ファイルシステム上の特定の場所にデータベースファイルを作成する権限がユーザーにないと、接続エラーが発生します。接続エラーを防止するための対策を以下に示します。

ユーザー固有のデータベースファイルを使用する 1 つのコンピューターでアプリケーションを使用するすべてのユーザーに対して 1 つのデータベースファイルを使用する代わりに、各ユーザーに専用のデータベースファイルを割り当てます。そのファイルは、ユーザーのアカウントに関連付けられているディレクトリに配置します（アプリケーションの格納ディレクトリ、ユーザーのドキュメントフォルダー、ユーザーのデスクトップなど）。

様々なユーザーの種類を考慮に入れる 様々なオペレーティングシステムで様々な種類のユーザーアカウントを使用してアプリケーションをテストします。ユーザーがコンピューターの管理者権限を持っているとは限りませんし、アプリケーションをインストールしたユーザーがアプリケーションを実行するとも限りません。

様々なファイルの場所を考慮に入れる ユーザーがデータベースファイルの保存場所を指定したり、開くファイルを選択したりできるようにする場合は、ユーザーによって使用される可能性があるファイルの場所について検討します。さらに、ユーザーがデータベースファイルを格納できる場所（またはデータベースファイルを開くことができる場所）を制限することも検討します。例えば、自分のユーザーアカウントの格納場所にあるファイルのみを開けるようにすることができます。

接続エラーが発生する可能性が最も高いのは、データベースを最初に作成したり開いたりするときです。したがって、接続エラーが発生すると、ユーザーがアプリケーションでデータベース関連の操作を一切実行できなくなります。読み取り専用エラーや権限エラーなどの一部の種類のエラーに対しては、データベースファイルを別の場所にコピーするという方法でエラーから回復できます。アプリケーションで、ユーザーがファイルの作成や書き込みの権限を持っている別の場所にデータベースファイルをコピーして、その場所を代わりに使用します。

シンタックスエラー

Adobe AIR 1.0 およびそれ以降

シンタックスエラーは、形式が正しくない SQL ステートメントをアプリケーションが実行しようとすると発生します。ローカルデータベースの SQL ステートメントは文字列として作成されるため、コンパイル時に SQL シンタックスをチェックすることはできません。SQL ステートメントのシンタックスをチェックするには、そのステートメントを実行する必要があります。SQL シンタックスエラーを防止するための対策を以下に示します。

すべての SQL ステートメントを徹底的にテストする できれば、アプリケーションの開発時（SQL ステートメントをステートメントテキストとしてアプリケーションコードに含める前）に別途 SQL ステートメントのテストを行います。さらに、単体テストなどのコードテスト手法を使用して、コードのすべてのオプションやバリエーションを実行するテストセットを作成します。

SQL を連結する（動的に生成する）代わりにステートメントパラメーターを使用する SQL ステートメントを動的に作成する代わりにパラメーターを使用すると、毎回同じ SQL ステートメントテキストを使用してステートメントが実行されることになります。その結果、ステートメントのテストが格段に容易になり、コードのバリエーションも限定されます。SQL ステートメントを動的に生成する必要がある場合は、ステートメントの動的な部分を最小限に抑えます。また、シンタックスエラーを引き起こさないようにユーザー入力を慎重に検証します。

アプリケーションでシンタックスエラーから回復するには、SQL ステートメントを調べてシンタックスを修正するための複雑なロジックが必要になります。シンタックスエラーを防止するための上述のガイドラインに従うと、実行時に SQL シンタックスエラーを引き起こす可能性がある場所（ステートメント内で使用されるユーザー入力など）をコードで特定できます。シンタックスエラーから回復するには、ユーザーにアドバイスを提供して、ステートメントが正常に実行されるようにするにはどこを修正すればよいかを知らせます。

制約エラー

Adobe AIR 1.0 およびそれ以降

制約エラーは、INSERT ステートメントや UPDATE ステートメントでデータを列に追加するときに、そのテーブルまたは列に対して定義されている制約に新しいデータが違反していると発生します。制約には次のようなものがあります。

UNIQUE 制約 テーブルのすべての行で列の値が重複していてはいけないことを表します。複数の列が UNIQUE 制約で結合される場合は、それらの列の値の組み合わせが重複していてはいけないことになります。つまり、一意として指定された列では、各行がそれぞれ異なっている必要があります。

主キー制約 制約で許可されるデータと許可されないデータに関しては、主キー制約は UNIQUE 制約と同じです。

NOT NULL 制約 列に NULL 値を格納できないことを表します。したがって、すべての行でその列に値がなければならないことになります。

CHECK 制約 1 つ以上のテーブルに対して任意の制約を指定できます。CHECK 制約では、列の値が取りうる範囲を定義するルール（数値列の値が 0 より大きくなければならないなど）を指定したり、列の値の関係（一方の列の値がもう一方の列の同じ行の値と異なっていなければならないなど）を指定したりするのが一般的です。

データ型（列の類似性）の制約 列の値のデータ型が強制されて、正しくない型の値を列に格納しようとするとエラーが発生します。ただし、多くの場合は、宣言されている列のデータ型に合わせて値が変換されます。詳しくは、226 ページの「データベースのデータ型の操作」を参照してください。

外部キーの値に対する制約は適用されません。したがって、外部キーの値が既存の主キーの値と一致している必要はありません。

ランタイムの SQL エンジンでは、あらかじめ定義されている種類の制約に加えて、トリガーを使用することもできます。トリガーとは、イベントハンドラーのように、特定のアクションが発生すると実行される定義済みの命令のセットです。例えば、特定のテーブルのデータが挿入されたり削除されたりした場合に実行されるトリガーを定義することができます。トリガーを使用すると、データの変更を調べて、指定した条件が満たされていない場合にエラーを発生させることもできます。したがって、トリガーは制約と同じ目的で使用することができます。制約エラーの防止と回復のための対策は、トリガーによって生成されるエラーにも当てはまります。ただし、トリガーによって生成されるエラーのエラー ID は制約エラーのエラー ID とは異なります。

特定のテーブルに適用する制約のセットは、アプリケーションの設計時に決定します。制約を意識的に設計すると、アプリケーションの設計で制約エラーの防止や回復が容易になります。ただし、制約エラーは体系的な予測や防止が困難です。制約エラーの予測が困難なのは、アプリケーションのデータが追加されるまで明らかにならないからです。制約エラーは、データベースの作成後、そこに追加されたデータによって発生します。それらのエラーは多くの場合、新しいデータとデータベースの既存のデータとの関係に起因しています。多くの制約エラーの防止に役立つ対策を以下に示します。

データベースの構造と制約について入念に計画する 制約の目的は、アプリケーションのルールを適用してデータベースのデータの整合性を保護することです。アプリケーションの計画を立てる際には、アプリケーションをサポートするために必要なデータベースの構造について検討し、その一環としてデータのルールを特定します（特定の値が必要かどうか、値に既定値があるかどうか、重複する値を許可するかどうかなど）。それらのルールに基づいてデータベースの制約を定義できます。

列の名前を明示的に指定する INSERT ステートメントは、値を挿入する列を明示的に指定しなくても記述できますが、無駄にリスクを高めることになります。値を挿入する列の名前を明示的に指定すると、自動的に生成される値、既定値を持つ列、NULL 値を許容する列などを使用できるようになります。また、すべての NOT NULL 列に明示的な値が挿入されていることを確認することもできます。

既定値を使用する 列に対して NOT NULL 制約を指定する場合は、可能であれば列定義で既定値を指定します。アプリケーションコードで既定値を指定することもできます。例えば、String 変数を使用してステートメントのパラメーター値を設定する前に、その String 変数が null かどうかをコードでチェックして値を割り当てることができます。

ユーザーが入力したデータを検証する 特に NOT NULL 制約と CHECK 制約では、ユーザーが入力したデータを事前にチェックして、制約によって指定されている制限に従っているかどうかを確認します。UNIQUE 制約では、このチェックを行うには SELECT クエリを実行してデータが一意かどうかを確認する必要があるため、必然的に他の制約に比べてチェックが困難になります。

トリガーを使用する トリガーを記述すると、挿入されたデータを検証（および場合によっては置換）したり、正しくないデータを修正するためのその他のアクションを実行したりできます。この検証と修正によって制約エラーの発生を防止できます。

制約エラーは様々な点で他の種類のエラーに比べて防止が困難ですが、幸い、アプリケーションが不安定になったり使用できなくなったりしないような形で制約エラーから回復するための方法がいくつかあります。

競合アルゴリズムを使用する 列に対して制約を定義した場合は、INSERT ステートメントや UPDATE ステートメントを作成するときに競合アルゴリズムを指定することができます。競合アルゴリズムは、制約違反が発生した場合にデータベース

で実行されるアクションを定義します。データベースエンジンで実行できるアクションはいくつかあります。例えば、1 つのステートメントを終了する、トランザクション全体を終了する、エラーを無視するなどのほか、古いデータを削除して、格納しようとしているデータに置き換えることもできます。

詳しくは、334 ページの「ローカルデータベースでの SQL サポート」の「ON CONFLICT（競合アルゴリズム）」を参照してください。

修正のためのフィードバックを提供する 特定の SQL コマンドに影響を与える可能性がある制約のセットは事前に特定できるため、ステートメントで発生する可能性がある制約エラーを予測することができます。その知識に基づいて、制約エラーに応答するアプリケーションロジックを作成できます。例えば、新しい製品を入力するためのデータ入力フォームを含むアプリケーションで、データベースの製品名列の定義で UNIQUE 制約を使用すると、データベースに新しい製品行を挿入するアクションで制約エラーが発生する可能性があります。したがって、このアプリケーションは、制約エラーを予測する形で設計されます。エラーが発生したら、指定した製品名は既に使用されているという内容の警告メッセージをユーザーに表示して、別の名前を選択するように求めます。あるいは、同じ名前を持つ他の製品に関する情報をユーザーが表示できるようにすることもできます。

データベースのデータ型の操作

Adobe AIR 1.0 およびそれ以降

データベースにテーブルを作成する際には、テーブルを作成するための SQL ステートメントでテーブルの各列の類似性（データ型）を定義します。類似性の宣言は省略することもできますが、CREATE TABLE SQL ステートメントで列の類似性を明示的に宣言することをお勧めします。

原則として、INSERT ステートメントを使用してデータベースに格納したオブジェクトは、SELECT ステートメントを実行すると同じデータ型のインスタンスとして返されます。ただし、値が格納されるデータベースの列の類似性によっては、異なるデータ型の値が返される場合もあります。値を列に格納するときに、その値のデータ型が列の類似性と一致していないと、値が列の類似性に合わせて変換されます。例えばデータベースの列が NUMERIC 類似性を使用して宣言されていた場合、挿入したデータは、格納される前に数値格納クラス（INTEGER または REAL）に変換されます。データを変換できない場合はエラーがスローされます。このルールに従うと、「12345」という String を NUMERIC 列に挿入した場合は、データベースに格納される前に 12345 という整数値に自動的に変換されます。その値を SELECT ステートメントで取得すると、String ではなく数値データ型（Number など）のインスタンスとして返されます。

望ましくないデータ型の変換が行われないようにするには、2 つのルールに従うのが最善の策になります。まず、各列を定義する際に、その列に格納する予定のデータの型と一致する類似性を使用して定義します。次に、定義されている類似性と一致するデータ型の値のみを挿入するようにします。これらのルールに従うと、2 つのメリットがあります。まず、データの挿入時にデータが予期せず変換される（その結果、場合によってはデータの本来の意味が失われる）ことがなくなります。また、データの取得時にデータが元のデータ型で返されるようになります。

使用可能な列の親和型の種類と SQL ステートメントでのデータ型の使用について詳しくは、355 ページの「データ型のサポート」を参照してください。

同期および非同期のデータベース操作の使用

Adobe AIR 1.0 およびそれ以降

これまでの節では、データの取得、挿入、更新、削除などの一般的なデータベース操作と、データベースファイルの作成やデータベース内のテーブルなどのオブジェクトの作成について説明し、それらの操作を非同期に実行する方法と同期的に実行する方法の両方の例を紹介しました。

先にも述べたように、非同期実行モードでは、データベースエンジンに操作を実行するように命令すると、データベースエンジンの処理がバックグラウンドで行われ、アプリケーションは引き続き実行されます。操作が完了すると、そのことを通知するイベントが送出されます。非同期実行には、データベース操作がバックグラウンドで実行されている間もメインのアプリケーションコードの実行が継続されるという大きな利点があります。このことは、操作の実行に時間がかかる場合に特に重要になります。

一方、同期実行モードでは、操作はバックグラウンドで実行されません。データベースエンジンに操作を実行するように命令すると、その時点でコードが一時停止して、その間にデータベースエンジンの処理が行われます。操作が完了すると、コードの次の行から実行が再開されます。

1 つのデータベース接続で操作やステートメントの一部を同期的に実行し、その他を非同期に実行することはできません。SQLConnection で同期実行モードと非同期実行モードのどちらを使用するかは、データベースへの接続を開くときに指定します。SQLConnection.open() を呼び出すとその接続で同期実行モードが使用され、SQLConnection.openAsync() を呼び出すと非同期実行モードが使用されます。open() または openAsync() を使用して SQLConnection インスタンスをデータベースに接続すると、実行モードが同期または非同期のいずれかに固定されます。

同期データベース操作の使用

Adobe AIR 1.0 およびそれ以降

同期実行を使用する場合に操作の実行や応答のために使用する実際のコードは、非同期実行モードのコードとほとんど変わりません。この 2 つのアプローチの主な違いは 2 つの領域に分類できます。1 つは別の操作に依存する操作の実行（SELECT の結果行や、INSERT ステートメントによって追加された行の主キーなど）、もう 1 つはエラー処理です。

同期操作のためのコードの記述

Adobe AIR 1.0 およびそれ以降

同期実行と非同期実行の主な違いとして、同期モードではコードを一続きのステップとして記述するのに対し、非同期コードではイベントリスナーを登録し、たいていは操作を複数のリスナーメソッドに分割します。データベースが同期実行モードで接続されている場合は、一連のデータベース操作を 1 つのコードブロックで連続して実行できます。次の例は、この方法を示しています。

```
// Include AIRAliases.js to use air.* shortcuts

var conn = new air.SQLConnection();

// The database file is in the application storage directory
var folder = File.applicationStorageDirectory;
var dbFile = folder.resolvePath("DBSample.db");

// open the database
conn.open(dbFile, air.OpenMode.UPDATE);

// start a transaction
conn.begin();

// add the customer record to the database
var insertCustomer = new air.SQLStatement();
insertCustomer.sqlConnection = conn;
insertCustomer.text =
    "INSERT INTO customers (firstName, lastName) " +
    "VALUES ('Bob', 'Jones')";
insertCustomer.execute();

var customerId = insertCustomer.getResult().lastInsertRowID;

// add a related phone number record for the customer
var insertPhoneNumber = new air.SQLStatement();
insertPhoneNumber.sqlConnection = conn;
insertPhoneNumber.text =
    "INSERT INTO customerPhoneNumbers (customerId, number) " +
    "VALUES (:customerId, '800-555-1234')";
insertPhoneNumber.parameters[":customerId"] = customerId;
insertPhoneNumber.execute();

// commit the transaction
conn.commit();
```

この例からわかるように、同期実行と非同期実行のどちらを使用する場合も、データベース操作を実行するために呼び出すメソッドは同じです。この 2 つのアプローチの主な違いは、別の操作に依存する操作の実行とエラー処理にあります。

別の操作に依存する操作の実行

Adobe AIR 1.0 およびそれ以降

同期実行モードを使用する場合は、操作の完了を確認するためにイベントをリッスンするコードを記述する必要はなく、コードの前の行の操作が正常に完了するとコードの次の行が実行されるものと考えることができます。したがって、別の操作の成功に依存する操作を実行するには、その操作のコードを依存する操作の直後に記述するだけで済みます。例えば、アプリケーションでトランザクションを開始して、INSERT ステートメントを実行し、挿入した行の主キーを取得して、その主キーを別のテーブルの別の行に挿入した後、最後にトランザクションをコミットするコードを記述する場合、これらすべてのコードを一続きのステートメントとして記述できます。これらの操作を実行する例を以下に示します。

```
// Include AIRAliases.js to use air.* shortcuts

var conn = new air.SQLConnection();

// The database file is in the application storage directory
var folder = File.applicationStorageDirectory;
var dbFile = folder.resolvePath("DBSample.db");

// open the database
conn.open(dbFile, air.OpenMode.UPDATE);

// start a transaction
conn.begin();

// add the customer record to the database
var insertCustomer = new air.SQLStatement();
insertCustomer.sqlConnection = conn;
insertCustomer.text =
    "INSERT INTO customers (firstName, lastName) " +
    "VALUES ('Bob', 'Jones')";
insertCustomer.execute();

var customerId = insertCustomer.getResult().lastInsertRowID;

// add a related phone number record for the customer
var insertPhoneNumber = new air.SQLStatement();
insertPhoneNumber.sqlConnection = conn;
insertPhoneNumber.text =
    "INSERT INTO customerPhoneNumbers (customerId, number) " +
    "VALUES (:customerId, '800-555-1234')";
insertPhoneNumber.parameters[":customerId"] = customerId;
insertPhoneNumber.execute();

// commit the transaction
conn.commit();
```

同期実行でのエラー処理

Adobe AIR 1.0 およびそれ以降

同期実行モードでは、操作の失敗を確認するためにエラーイベントをリッスンする代わりに、エラーをトリガーする可能性があるコードを try..catch..finally コードブロックで囲みます。エラーをスローするコードを try ブロックでラップし、それぞれの種類のエラーに対して実行するアクションを個別の catch ブロックに記述して、操作が成功したか失敗したかに関係なく常に実行するコード（不要になったデータベース接続を閉じるコードなど）を finally ブロックに配置します。エラー処理のために try..catch..finally ブロックを使用する例を以下に示します。この例は、前の例にエラー処理のコードを追加したものです。

```
// Include AIRAliases.js to use air.* shortcuts

var conn = new air.SQLConnection();

// The database file is in the application storage directory
var folder = File.applicationStorageDirectory;
var dbFile = folder.resolvePath("DBSample.db");

// open the database
conn.open(dbFile, air.SQLMode.UPDATE);

// start a transaction
conn.begin();

try
{
    // add the customer record to the database
    var insertCustomer = new air.SQLStatement();
    insertCustomer.sqlConnection = conn;
    insertCustomer.text =
        "INSERT INTO customers (firstName, lastName)" +
        "VALUES ('Bob', 'Jones')";

    insertCustomer.execute();

    var customerId = insertCustomer.getResult().lastInsertRowID;

    // add a related phone number record for the customer
    var insertPhoneNumber = new air.SQLStatement();
    insertPhoneNumber.sqlConnection = conn;
    insertPhoneNumber.text =
        "INSERT INTO customerPhoneNumbers (customerId, number)" +
        "VALUES (:customerId, '800-555-1234')";
    insertPhoneNumber.parameters[":customerId"] = customerId;

    insertPhoneNumber.execute();

    // if we've gotten to this point without errors, commit the transaction
    conn.commit();
}
catch (error)
{
    // rollback the transaction
    conn.rollback();
}
```

非同期実行モデルについて

Adobe AIR 1.0 およびそれ以降

非同期実行モードについて、同じデータベース接続に対して別の SQLStatement が実行されている場合には SQLStatement インスタンスを開始できないと考えている人がよくいますが、それは正しくありません。SQLStatement インスタンスが実行されている間は、そのステートメントの text プロパティを変更することはできません。ただし、実行する SQL ステートメントごとに異なる SQLStatement インスタンスを使用すれば、まだ別の SQLStatement インスタンスが実行されている間に SQLStatement の execute() メソッドを呼び出すことができます。エラーが発生することもありません。

非同期実行モードを使用してデータベース操作を実行する場合、各データベース接続（各 SQLConnection インスタンス）に対して固有のキュー（実行するように命令された操作のリスト）が内部で割り当てられます。各操作はそのキューに追加された順に実行されます。SQLStatement インスタンスを作成してその execute() メソッドを呼び出すと、そのステートメン

トの実行操作が接続のキューに追加され、その SQLConnection インスタンスで現在実行されている操作がなければ、バックグラウンドで実行が開始されます。ここで、同じコードブロックの中で別の SQLStatement インスタンスを作成し、その execute() メソッドを呼び出したとすると、その 2 つ目のステートメントの実行操作はキューの 1 つ目のステートメントの後に追加されます。1 つ目のステートメントの実行が完了すると、すぐにキューの次の操作が実行されます。キューの次の操作の処理は、まだメインアプリケーションコードで 1 つ目の操作の result イベントが送出されている間にバックグラウンドで開始されます。このテクニックを説明するコードを次に示します。

```
// Using asynchronous execution mode
var stmt1 = new air.SQLStatement();
stmt1.sqlConnection = conn;

// ... Set statement text and parameters, and register event listeners ...

stmt1.execute();

// At this point stmt1's execute() operation is added to conn's execution queue.

var stmt2 = new air.SQLStatement();
stmt2.sqlConnection = conn;

// ... Set statement text and parameters, and register event listeners ...

stmt2.execute();

// At this point stmt2's execute() operation is added to conn's execution queue.
// When stmt1 finishes executing, stmt2 will immediately begin executing
// in the background.
```

このようにキューの次のステートメントを自動的に実行する際には、重要な副作用として、ステートメントが別の操作の結果に依存している場合はその 1 つ目の操作が完了するまでそのステートメントをキューに追加できません（つまり、そのステートメントの execute() メソッドを呼び出すことができません）。これは、その 2 つ目のステートメントの execute() メソッドを呼び出してしまうと、そのステートメントの text プロパティや parameters プロパティを変更できなくなるからです。この場合は、1 つ目の操作の完了を知らせるイベントを待ってから次の操作を開始する必要があります。例えば、トランザクションのコンテキストでステートメントを実行する場合は、そのステートメントの実行はトランザクションを開く操作に依存するため、SQLConnection.begin() メソッドを呼び出してトランザクションを開いた後、SQLConnection インスタンスの begin イベントが送出されるまで待機する必要があります。このイベントが送出されるまでは SQLStatement インスタンスの execute() メソッドを呼び出すことはできません。この例で、操作が正常に実行されるようにアプリケーションを構成するには、begin イベントのリスナーとして登録されたメソッドを作成するのが最も簡単です。そのリスナーメソッドの中に、SQLStatement.execute() メソッドを呼び出すコードを配置します。

SQL データベースでの暗号化の使用

Adobe AIR 1.5 およびそれ以降

すべての Adobe AIR アプリケーションは、同じローカルデータベースエンジンを共有します。したがって、すべての AIR アプリケーションは、暗号化されていないデータベースファイルに対して接続、読み取り、書き込みを行うことができます。Adobe AIR 1.5 以降の AIR には、暗号化されたデータベースファイルの作成と接続の機能が含まれています。暗号化されたデータベースを使用する場合、データベースに接続するには、アプリケーションは適切な暗号化キーを提供する必要があります。正しくない暗号化キーが提供された場合、または暗号化キーが提供されなかった場合、アプリケーションはデータベースに接続できません。この結果、アプリケーションはデータベースからデータを読み取ることができず、データベースへのデータの書き込みやデータベースのデータの変更も行うことができません。

暗号化されたデータベースを使用するには、暗号化されたデータベースとしてデータベースを作成する必要があります。暗号化された既存のデータベースがある場合は、そのデータベースへの接続を開くことができます。暗号化されたデータベースの暗号化キーを変更することもできます。暗号化されたデータベースの使用方法については、その作成方法と接続方法以外は、暗号化されていないデータベースの場合と同じです。例えば、SQL ステートメントの実行は、データベースが暗号化されている場合も暗号化されていない場合も同じです。

暗号化されたデータベースの使用

Adobe AIR 1.5 およびそれ以降

暗号化は、データベースに保存されている情報へのアクセスを制限する場合に役立ちます。Adobe AIR のデータベース暗号化機能は、いくつかの用途に使用できます。暗号化されたデータベースを使用する場合の例を、いくつか次に示します。

- サーバーからダウンロードした非公開のアプリケーションデータの読み取り専用キャッシュ。
- サーバーと同期している、非公開データのローカルアプリケーションストア（データはサーバーに送信され、サーバーから読み込まれます）。
- アプリケーションによって作成および編集されたドキュメントのファイル形式として使用される、暗号化されたファイル。ファイルは、特定のユーザー専用である場合と、アプリケーションのすべてのユーザーで共有するように設計されている場合があります。
- ローカルデータストアのその他の用途。例えば、201 ページの「ローカル SQL データベースの用途」に記載されているように、マシンまたはデータベースファイルにアクセスできる他のユーザーに対してデータを公開しない場合があります。

暗号化されたデータベースの使用が必要となる理由がわかると、アプリケーションの設計方法を決定しやすくなります。このことは、特に、アプリケーションによるデータベースの暗号化キーの作成、取得および保存の方法に影響します。これらの考慮事項について詳しくは、235 ページの「データベースで暗号化を使用する場合の考慮事項」を参照してください。

暗号化されたデータベース以外に、機密データを保護する方法としては、暗号化されたローカルストアがあります。暗号化されたローカルストアの場合、String 型のキーを使用して単一の ByteArray 値を格納します。この値に対するアクセスは、値を格納する AIR アプリケーション自体によって、値が格納されているコンピューター上から実行される場合に限り可能です。暗号化されたローカルストアの場合、独自に暗号化キーを作成する必要はありません。このため、暗号化されたローカルストアは、ByteArray に容易にエンコードできるストリング値または値セットを格納する場合に最も適しています。暗号化されたデータベースは、構造化されたデータの格納と照会が必要とされる、より大きなデータセットの場合に最も適しています。暗号化されたローカルストアの使用について詳しくは、250 ページの「暗号化されたローカルストア」を参照してください。

暗号化されたデータベースの作成

Adobe AIR 1.5 およびそれ以降

暗号化されたデータベースを使用するには、データベースファイルを作成時に暗号化する必要があります。暗号化せずに作成したデータベースを、後で暗号化することはできません。同様に、暗号化されたデータベースを後で非暗号化することもできません。必要な場合は、暗号化されたデータベースの暗号化キーを変更できます。詳しくは、235 ページの「データベースの暗号化キーの変更」を参照してください。暗号化されていない既存のデータベースがあり、データベース暗号化を使用する必要がある場合は、暗号化されたデータベースを新たに作成し、既存のテーブル構造とデータを新しいデータベースにコピーします。

暗号化されたデータベースを作成する方法は、暗号化されていないデータベースを作成する方法（205 ページの「データベースの作成」を参照）とほぼ同じです。最初に、データベースへの接続を表す SQLConnection インスタンスを作成します。SQLConnection オブジェクトの open() メソッドまたは openAsync() メソッドを呼び出して、データベースを作成しま

す。このとき、ファイルがまだ存在しないデータベースの場所を指定します。暗号化されたデータベースを作成する場合、唯一の違いは、encryptionKey パラメーター（open() メソッドの第 5 パラメーターであり、openAsync() メソッドの第 6 パラメーター）の値を指定することです。

有効な encryptionKey パラメーターの値は、16 バイトを格納している ByteArray オブジェクトです。

次の例では、暗号化されたデータベースの作成方法を示します。わかりやすくするために、これらの例では、暗号化キーをアプリケーションコード内にハードコードしています。ただし、この方法は安全ではないため、使用しないでください。

```
var conn = new air.SQLConnection();

var encryptionKey = new air.ByteArray();
encryptionKey.writeUTFBytes("Some16ByteString"); // This technique is not secure!

// Create an encrypted database in asynchronous mode
conn.openAsync(dbFile, air.SQLMode.CREATE, null, false, 1024, encryptionKey);

// Create an encrypted database in synchronous mode
conn.open(dbFile, air.SQLMode.CREATE, false, 1024, encryptionKey);
```

暗号化キーの生成の推奨される方法を示す例については、236 ページの「例：暗号化キーの生成と使用」を参照してください。

暗号化されたデータベースへの接続

Adobe AIR 1.5 およびそれ以降

暗号化されたデータベースの作成方法と同様に、暗号化されたデータベースへの接続を開く手順は、暗号化されていないデータベースへの接続の場合と似ています。この手順については、208 ページの「データベースへの接続」を参照してください。接続を同期実行モードで開く場合は open() メソッドを呼び出し、非同期実行モードで開く場合は openAsync() メソッドを呼び出します。唯一の違いは、暗号化されたデータベースを開くには、encryptionKey パラメーター（open() メソッドの第 5 パラメーターであり、openAsync() メソッドの第 6 パラメーター）の適切な値を指定することです。

指定された暗号化キーが正しくない場合はエラーが発生します。open() メソッドの場合は、SQLError 例外がスローされます。openAsync() メソッドの場合は、SQLConnection オブジェクトによって SQLErrorEvent が送出されます。これには、SQLError オブジェクトの error プロパティが格納されています。いずれの場合も、例外によって生成された SQLError オブジェクトの errorID プロパティの値は 3138 になります。このエラー ID は、「File opened is not a database file」というエラーメッセージに対応します。

暗号化されたデータベースを非同期実行モードで開く方法を次の例に示します。わかりやすくするために、この例では、暗号化キーをアプリケーションコードでハードコードしています。ただし、この方法は安全ではないため、使用しないでください。

```
// Include AIRAliases.js to use air.* shortcuts
var conn = new air.SQLConnection();
conn.addEventListener(air.SQLEvent.OPEN, openHandler);
conn.addEventListener(air.SQLErrorEvent.ERROR, errorHandler);
var dbFile = air.File.applicationStorageDirectory.resolvePath("DBSample.db");

var encryptionKey = new air.ByteArray();
encryptionKey.writeUTFBytes("Some16ByteString"); // This technique is not secure!

conn.openAsync(dbFile, air.SQLMode.UPDATE, null, false, 1024, encryptionKey);

function openHandler(event)
{
    air.trace("the database opened successfully");
}

function errorHandler(event)
{
    if (event.error.errorID == 3138)
    {
        air.trace("Incorrect encryption key");
    }
    else
    {
        air.trace("Error message:", event.error.message);
        air.trace("Details:", event.error.details);
    }
}
```

暗号化されたデータベースを同期実行モードで開く方法を次の例に示します。わかりやすくするために、この例では、暗号化キーをアプリケーションコードでハードコードしています。ただし、この方法は安全ではないため、使用しないでください。

```
// Include AIRAliases.js to use air.* shortcuts
var conn = new air.SQLConnection();
var dbFile = air.File.applicationStorageDirectory.resolvePath("DBSample.db");

var encryptionKey = new air.ByteArray();
encryptionKey.writeUTFBytes("Some16ByteString"); // This technique is not secure!

try
{
    conn.open(dbFile, air.SQLMode.UPDATE, false, 1024, encryptionKey);
    air.trace("the database was created successfully");
}
catch (error)
{
    if (error.errorID == 3138)
    {
        air.trace("Incorrect encryption key");
    }
    else
    {
        air.trace("Error message:", error.message);
        air.trace("Details:", error.details);
    }
}
```

暗号化キーの生成の推奨される方法を示す例については、236 ページの「例：暗号化キーの生成と使用」を参照してください。

データベースの暗号化キーの変更

Adobe AIR 1.5 およびそれ以降

データベースが暗号化されている場合、データベースの暗号化キーを後で変更できます。データベースの暗号化キーを変更するには、まず、データベースへの接続を開きます。そのためには、SQLConnection インスタンスを作成し、open() メソッドまたは openAsync() メソッドを呼び出します。データベースに接続したら、新しい暗号化キーを引数として渡して、reencrypt() メソッドを呼び出します。

他の多くのデータベース操作と同様に、reencrypt() メソッドの動作は、データベースの接続に使用するモードが同期実行モードか非同期実行モードかによって異なります。open() メソッドを使用してデータベースに接続した場合、reencrypt() 操作は同期的に実行されます。操作が完了すると、コードの次の行から実行が再開されます。

```
var newKey = new air.ByteArray();
// ... generate the new key and store it in newKey
conn.reencrypt(newKey);
```

これに対し、openAsync() メソッドを使用してデータベース接続を開いた場合、reencrypt() 操作は非同期で実行されます。reencrypt() を呼び出すと、再暗号化プロセスが開始されます。操作が完了すると、SQLConnection オブジェクトによって reencrypt イベントが送出されます。再暗号化が完了したことを判別するには、イベントリスナーを使用します。

```
var newKey = new air.ByteArray();
// ... generate the new key and store it in newKey

conn.addEventListener(air.SQLEvent.REENCRYPT, reencryptHandler);

conn.reencrypt(newKey);

function reencryptHandler(event)
{
    // save the fact that the key changed
}
```

reencrypt() 操作は、独自のトランザクション内で実行されます。操作が中断または失敗する（例えば、操作が完了する前にアプリケーションが閉じられる）と、トランザクションがロールバックされます。その場合は、データベースの暗号化キーは元の暗号化キーのままになります。

reencrypt() メソッドを使用して、データベースの暗号化を解除することはできません。null 値または 16 バイト以外の ByteArray を reencrypt() メソッドに渡すと、エラーが発生します。

データベースで暗号化を使用する場合の考慮事項

Adobe AIR 1.5 およびそれ以降

232 ページの「暗号化されたデータベースの使用」では、暗号化されたデータベースの使用が適している例をいくつか示します。機密性の要件は、アプリケーションの使用例（ここに示す例および他の例を含む）によって異なります。アプリケーションでの暗号化の使用をどのように設計するかが、データベースのデータの機密性を制御する上で重要な役割を担います。例えば、暗号化されたデータベースを使用して、同じマシンを使用している他のユーザーに対しても個人情報を公開しないようにするには、各ユーザーのデータベースで独自の暗号化キーが必要になります。セキュリティを最大限に高めるために、ユーザーが入力したパスワードに基づいてキーを生成することができます。パスワードに基づいて暗号化キーを生成すると、他のユーザーがマシンのユーザーのアカウントを偽装できたとしても、データにアクセスすることはできません。機密性について別の観点から考えてみましょう。特定のアプリケーションのすべてのユーザーがデータベースファイルを読み取ることができるようにする必要があるとします。ただし、他のアプリケーションのユーザーは、このデータベースファイルを読み取ることができないようにします。この場合、そのアプリケーションがインストールされている各環境から、共有の暗号化キーにアクセスできる必要があります。

アプリケーションデータに求められる機密性のレベルに応じて、アプリケーション（具体的には、暗号化キーの生成に使用する手法）を設計できます。様々なデータ機密性レベルにおけるデザイン上の推奨事項を次の一覧に示します。

- 任意のマシンにインストールされている特定のアプリケーションにアクセスできるすべてのユーザーがデータベースにアクセスできるようにするには、アプリケーションのすべてのインスタンスで利用できる単一のキーを使用します。例えば、アプリケーションの初回実行時に、SSL などのセキュリティ保護されたプロトコルを使用して、サーバーから共有の暗号化キーをダウンロードできます。このキーを、暗号化されたローカルストアに保存して、次回以降も使用できるようにします。別の方法としては、マシンのユーザーごとにデータを暗号化し、サーバーなどのリモートデータストアとデータを同期させることで、データをポータブルにします。
- 単一のユーザーがどのマシンからでもデータベースにアクセスできるようにするには、ユーザーの秘密（パスワードなど）から暗号化キーを生成します。特に、特定のコンピューターに関連付けられた値（例えば、暗号化されたローカルストアに保存されている値）は、キーの生成に使用しないでください。別の方法としては、マシンのユーザーごとにデータを暗号化し、サーバーなどのリモートデータストアとデータを同期させることで、データをポータブルにします。
- 単一のマシンの単一のユーザーのみがデータベースにアクセスできるようにするには、パスワードおよび生成されたソルトからキーを生成します。この方法の例については、236 ページの「例：暗号化キーの生成と使用」を参照してください。

暗号化されたデータベースを使用するアプリケーションを設計する場合に注意する必要がある、セキュリティ上の追加考慮事項を次に示します。

- システムの安全性は、その最も弱い部分で決まります。ユーザーが入力したパスワードを使用して暗号化キーを生成する場合は、パスワードの最小の長さと複雑さについての制約を設けることを検討してください。基本文字のみで構成される短いパスワードは、簡単に推測できます。
- AIR アプリケーションのソースコードは、プレーンテキスト（HTML コンテンツの場合）または簡単に逆コンパイルできるバイナリ形式（SWF コンテンツの場合）でユーザーのマシンに保存されます。ソースコードはアクセス可能なので、次の 2 つの点に注意する必要があります。
  - 暗号化キーをソースコード内にハードコードしないこと
  - 暗号化キーの生成に使用する技法（ランダム文字生成ツールや特定のハッシュアルゴリズムなど）は、攻撃者によって簡単に解読されることを前提とすること
- AIR のデータベース暗号化では、AES（Advanced Encryption Standard）を CCM（Counter with CBC-MAC）モードで使用します。この暗号化では、暗号の安全性を高めるために、ユーザーが入力したキーをソルト値と組み合わせることが必要になります。この方法の例については、236 ページの「例：暗号化キーの生成と使用」を参照してください。
- データベースを暗号化することにした場合、そのデータベースに関連してデータベースエンジンが使用するすべてのディスクファイルが暗号化されます。ただし、データベースエンジンでは、トランザクションにおける読み取りと書き込みのパフォーマンスを改善するために、一部のデータを一時的にインメモリキャッシュに保持します。メモリに常駐するデータは暗号化されません。攻撃者がデバッガーなどを使用して、AIR アプリケーションによって使用されているメモリにアクセスできてしまうと、現在開いている、暗号化されていないデータベースのデータが入手可能になります。

例：暗号化キーの生成と使用

Adobe AIR 1.5 およびそれ以降

このアプリケーション例では、暗号化キーを生成する方法を 1 つ示します。このアプリケーションは、ユーザーのデータについて、最も高いレベルの機密性と安全性を提供するように設計されています。非公開データのセキュリティ保護に関する重要な側面の 1 つは、アプリケーションがデータベースに接続する際、ユーザーにパスワードの入力を毎回要求することです。したがって、この例が示すように、このレベルの機密性を要求するアプリケーションがデータベースの暗号化キーを直接保存しておくことは絶対に避けるべきです。

このアプリケーションは、暗号化キーを生成する ActionScript クラス（EncryptionKeyGenerator クラス）、および、そのクラスの使用方法を示す基本的なユーザーインターフェイスという 2 つの部分から構成されます。ソースコード全体については、239 ページの「暗号化キーの生成と使用のコード例全体」を参照してください。

EncryptionKeyGenerator クラスによる安全な暗号化キーの取得

Adobe AIR 1.5 およびそれ以降

アプリケーションで EncryptionKeyGenerator クラスを使用する際にその動作の詳細を理解する必要はありません。データベースの暗号化キーがクラスによって生成される方法について詳しくは、245 ページの「EncryptionKeyGenerator クラスについて」を参照してください。

アプリケーションで EncryptionKeyGenerator クラスを使用するには、次の手順に従ってください。

1. EncryptionKeyGenerator ライブラリをダウンロードします。EncryptionKeyGenerator クラスは、オープンソースの ActionScript 3.0 コアライブラリ（as3corelib）プロジェクトに含まれるクラスです。as3corelib パッケージ（ソースコードおよびドキュメントを含む）をダウンロードできます。また、プロジェクトページから SWC またはソースコードのファイルをダウンロードすることもできます。

2. SWC から SWF ファイルを抽出します。SWF ファイルを抽出するには、SWC ファイルの名前を ".zip" ファイル名拡張子を付けて変更し、その ZIP ファイルを開きます。ZIP ファイルから SWF ファイルを抽出し、アプリケーションソースコードが見つけられる場所に置きます。例えば、アプリケーションのメイン HTML ファイルがあるフォルダーに置くこともできます。必要に応じて、SWF ファイルの名前を変更できます。この例では、SWF ファイルを "EncryptionKeyGenerator.swf." という名前にしています。

3. アプリケーションソースコードで、SWF ファイルにリンクされた <script> ブロックを追加して、SWF コードライブラリを読み込みます。この方法について詳しくは、33 ページの「HTML ページ内での ActionScript ライブラリの使用」を参照してください。次のコードは、SWF ファイルをコードライブラリとして使用できるようにします。

   ```
   <script type="application/x-shockwave-flash" src="EncryptionKeyGenerator.swf"/>
   ```

   デフォルトでは、コード window.runtime の後にフルパッケージ名およびクラス名を指定して、クラスを使用できます。EncryptionKeyGenerator の場合、フルネームは次のようになります。

   ```
   window.runtime.com.adobe.air.crypto.EncryptionKeyGenerator
   ```

   クラスのエイリアスを作成して、フルネームを入力しなくてもすむようにできます。次のコードは、EncryptionKeyGenerator クラスを表すエイリアス ekg.EncryptionKeyGenerator を作成します。

   ```
   var ekg;
   if (window.runtime)
   {
       if (!ekg) ekg = {};
       ekg.EncryptionKeyGenerator = window.runtime.com.adobe.air.crypto.EncryptionKeyGenerator;
   }
   ```

4. コード内の、データベースを作成またはデータベースへの接続を開くポイントの前に、EncryptionKeyGenerator() コンストラクターを呼び出して EncryptionKeyGenerator インスタンスを作成するコードを追加します。

   ```
   var keyGenerator = new ekg.EncryptionKeyGenerator();
   ```

5. ユーザーからパスワードを取得します。

   ```
   var password = passwordInput.value;

   if (!keyGenerator.validateStrongPassword(password))
   {
       // display an error message
       return;
   }
   ```

EncryptionKeyGenerator インスタンスは、暗号化キーのベースとしてこのパスワードを使用します（次の手順で示します）。EncryptionKeyGenerator インスタンスは、強力なパスワードの検証要件に照らし合わせてパスワードをテストします。検証に失敗すると、エラーが発生します。このサンプルコードのように、EncryptionKeyGenerator オブジェクトの validateStrongPassword() メソッドを呼び出すことで、パスワードを事前にチェックすることができます。このように、パスワードが強力なパスワードの最小限の要件を満たすかどうかをチェックして、エラーを防ぐことができます。

6 次のように、パスワードから暗号化キーを生成します。

```
var encryptionKey = keyGenerator.getEncryptionKey(password);
```

getEncryptionKey() メソッドは、暗号化キー（16 バイト ByteArray）を生成して返します。暗号化キーを使用して、暗号化された新しいデータベースを作成するか、既存のデータベースを開くことができます。

getEncryptionKey() メソッドは必要なパラメーターを 1 つ持ちます。これは手順 5 で取得されたパスワードです。

注意：アプリケーションでは、最も高いレベルのデータのセキュリティとプライバシーを維持するために、アプリケーションがデータベースに接続するたびにユーザーのパスワード入力を要求する必要があります。ユーザーのパスワードまたはデータベース暗号化キーを直接格納しないでください。セキュリティのリスクが高くなります。代わりに、この例に示すように、暗号化データベースの作成時とその接続時に、同じ方法を使用してパスワードから暗号化キーを抽出するようにします。

getEncryptionKey() メソッドは、2 番目（オプション）のパラメーターである overrideSaltELSKey パラメーターも受け入れます。EncryptionKeyGenerator は、**salt** と呼ばれるランダムな値を作成します。これは、暗号化キーの一部として使用されます。暗号化キーを再作成できるようにするために、salt の値は AIR アプリケーションの暗号化されたローカルストア（ELS）に格納されます。デフォルトでは、EncryptionKeyGenerator クラスは特定の String を ELS キーとして使用します。まれに、アプリケーションが使用している他のキーと、このキーが競合する可能性があります。デフォルトのキーを使用する代わりに、独自の ELS キーを指定することもできます。その場合は、次のように、2 番目の getEncryptionKey() パラメーターとしてカスタムキーを渡すことで、カスタムキーを指定します。

```
var customKey = "My custom ELS salt key";
var encryptionKey = keyGenerator.getEncryptionKey(password, customKey);
```

7 データベースの作成とオープン

getEncryptionKey() メソッドから返される暗号化キーにより、コードは新しい暗号化されたデータベースを作成するか、既存の暗号化されたデータベースを開こうとします。いずれの場合も、SQLConnection クラスの open() メソッドまたは openAsync() メソッドを使用します。詳しくは、232 ページの「暗号化されたデータベースの作成」および 233 ページの「暗号化されたデータベースへの接続」を参照してください。

この例では、アプリケーションは非同期実行モードでデータベースを開くように設計されています。該当するイベントリスナーを設定し、最後の引数として暗号化キーを渡して、openAsync() メソッドを呼び出します。

```
conn.addEventListener(air.SQLEvent.OPEN, openHandler);
conn.addEventListener(air.SQLErrorEvent.ERROR, openError);

conn.openAsync(dbFile, air.SQLMode.CREATE, null, false, 1024, encryptionKey);
```

リスナーのメソッドでは、イベントリスナーの登録を解除します。次に、データベースが作成または開かれたか、エラーが発生したかを示すステータスメッセージが表示されます。これらのイベントハンドラーの最も注目すべき点は、openError() メソッドです。このメソッドでは、if ステートメントが、データベースが存在するかどうか（コードが既存のデータベースに接続しようとしているかどうか）、および、エラー ID が定数 EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID に合致するかどうかをチェックします。この両方の条件が成立する場合、おそらく、ユーザーが入力したパスワードが正しくないことを意味します（指定されたファイルがデータベースファイルではないことを意味する可能性もあります）。次のコードは、エラー ID をチェックするコードです。

```
if (!createNewDB && event.error.errorID == ekg.EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID)
{
    statusMsg.innerHTML = "<p class='error'>Incorrect password!</p>";
}
else
{
    statusMsg.innerHTML = "<p class='error'>Error creating or opening database.</p>";
}
```

この例のイベントリスナーのコード全体については、239 ページの「暗号化キーの生成と使用のコード例全体」を参照してください。

暗号化キーの生成と使用のコード例全体

Adobe AIR 1.5 およびそれ以降

アプリケーション例「暗号化キーの生成と使用」のコード全体を次に示します。このコードは 2 つの部分から構成されます。

この例では、EncryptionKeyGenerator を使用してパスワードから暗号化キーを作成しています。EncryptionKeyGenerator クラスは、オープンソースの ActionScript 3.0 コアライブラリ（as3corelib）プロジェクトに含まれるクラスです。as3corelib パッケージ（ソースコードおよびドキュメントを含む）をダウンロードできます。また、プロジェクトページから SWC またはソースコードのファイルをダウンロードすることもできます。

Flex の例

アプリケーションの MXML ファイルには、暗号化されたデータベースに対する接続を作成する（開く）単純なアプリケーションのソースコードが含まれています。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
creationComplete="init();">
    <mx:Script>
        <![CDATA[
            import com.adobe.air.crypto.EncryptionKeyGenerator;

            private const dbFileName:String = "encryptedDatabase.db";

            private var dbFile:File;
            private var createNewDB:Boolean = true;
            private var conn:SQLConnection;

            // ------- Event handling -------

            private function init():void
            {
                conn = new SQLConnection();
                dbFile = File.applicationStorageDirectory.resolvePath(dbFileName);
                if (dbFile.exists)
                {
                    createNewDB = false;
                    instructions.text = "Enter your database password to open the encrypted database.";
                    openButton.label = "Open Database";
                }
            }

            private function openConnection():void
            {
                var password:String = passwordInput.text;

                var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();
```

```
                if (password == null || password.length <= 0)
                {
                    statusMsg.text = "Please specify a password.";
                    return;
                }

                if (!keyGenerator.validateStrongPassword(password))
                {
                    statusMsg.text = "The password must be 8-32 characters long. It must contain at least
one lowercase letter, at least one uppercase letter, and at least one number or symbol.";
                    return;
                }

                passwordInput.text = "";
                passwordInput.enabled = false;
                openButton.enabled = false;

                var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password);

                conn.addEventListener(SQLEvent.OPEN, openHandler);
                conn.addEventListener(SQLErrorEvent.ERROR, openError);

                conn.openAsync(dbFile, SQLMode.CREATE, null, false, 1024, encryptionKey);
            }

            private function openHandler(event:SQLEvent):void
            {
                conn.removeEventListener(SQLEvent.OPEN, openHandler);
                conn.removeEventListener(SQLErrorEvent.ERROR, openError);

                statusMsg.setStyle("color", 0x009900);
                if (createNewDB)
                {
                    statusMsg.text = "The encrypted database was created successfully.";
                }
                else
                {
                    statusMsg.text = "The encrypted database was opened successfully.";
                }
            }

            private function openError(event:SQLErrorEvent):void
            {
                conn.removeEventListener(SQLEvent.OPEN, openHandler);
                conn.removeEventListener(SQLErrorEvent.ERROR, openError);

                if (!createNewDB && event.error.errorID ==
```

```
EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID)
                {
                    statusMsg.text = "Incorrect password!";
                }
                else
                {
                    statusMsg.text = "Error creating or opening database.";
                }
            }
        ]]>
    </mx:Script>
    <mx:Text id="instructions" text="Enter a password to create an encrypted database. The next time you
open the application, you will need to re-enter the password to open the database again." width="75%"
height="65"/>
    <mx:HBox>
        <mx:TextInput id="passwordInput" displayAsPassword="true"/>
        <mx:Button id="openButton" label="Create Database" click="openConnection();"/>
    </mx:HBox>
    <mx:Text id="statusMsg" color="#990000" width="75%"/>
</mx:WindowedApplication>
```

Flash Professional の例

アプリケーションの FLA ファイルには、暗号化されたデータベースに対する接続を作成する（開く）単純なアプリケーションのソースコードが含まれています。FLA ファイルのステージ上には、次に示す 4 個のコンポーネントが配置されています。

インスタンス名	コンポーネントの種類	説明
instructions	Label	ユーザーに対する手順説明
passwordInput	TextInput	ユーザーがパスワードを入力するための入力フィールド
openButton	Button	ユーザーがパスワードを入力した後にクリックするボタン
statusMsg	Label	ステータス（成功、失敗）メッセージの表示

アプリケーションのコードは、メインタイムラインのフレーム 1 にあるキーフレーム上に定義されています。アプリケーションのコードを次に示します。

```
import com.adobe.air.crypto.EncryptionKeyGenerator;

const dbFileName:String = "encryptedDatabase.db";

var dbFile:File;
var createNewDB:Boolean = true;
var conn:SQLConnection;

init();

// ------- Event handling -------

function init():void
{
    passwordInput.displayAsPassword = true;
    openButton.addEventListener(MouseEvent.CLICK, openConnection);
    statusMsg.setStyle("textFormat", new TextFormat(null, null, 0x990000));

    conn = new SQLConnection();
    dbFile = File.applicationStorageDirectory.resolvePath(dbFileName);

    if (dbFile.exists)
```

```
    {
        createNewDB = false;
        instructions.text = "Enter your database password to open the encrypted database.";
        openButton.label = "Open Database";
    }
    else
    {
        instructions.text = "Enter a password to create an encrypted database. The next time you open the
application, you will need to re-enter the password to open the database again.";
        openButton.label = "Create Database";
    }
}

function openConnection(event:MouseEvent):void
{
    var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();

    var password:String = passwordInput.text;

    if (password == null || password.length <= 0)
    {
        statusMsg.text = "Please specify a password.";
        return;
    }

    if (!keyGenerator.validateStrongPassword(password))
    {
        statusMsg.text = "The password must be 8-32 characters long. It must contain at least one lowercase
letter, at least one uppercase letter, and at least one number or symbol.";
        return;
    }

    passwordInput.text = "";
    passwordInput.enabled = false;
    openButton.enabled = false;

    var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password);

    conn.addEventListener(SQLEvent.OPEN, openHandler);
    conn.addEventListener(SQLErrorEvent.ERROR, openError);

    conn.openAsync(dbFile, SQLMode.CREATE, null, false, 1024, encryptionKey);
}

function openHandler(event:SQLEvent):void
{
    conn.removeEventListener(SQLEvent.OPEN, openHandler);
    conn.removeEventListener(SQLErrorEvent.ERROR, openError);

    statusMsg.setStyle("textFormat", new TextFormat(null, null, 0x009900));
    if (createNewDB)
    {
        statusMsg.text = "The encrypted database was created successfully.";
    }
```

```
    else
    {
        statusMsg.text = "The encrypted database was opened successfully.";
    }
}

function openError(event:SQLErrorEvent):void
{
    conn.removeEventListener(SQLEvent.OPEN, openHandler);
    conn.removeEventListener(SQLErrorEvent.ERROR, openError);

    if (!createNewDB && event.error.errorID == EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID)
    {
        statusMsg.text = "Incorrect password!";
    }
    else
    {
        statusMsg.text = "Error creating or opening database.";
    }
}
```

アプリケーションの HTML ファイルには、暗号化されたデータベースに対する接続を作成する（開く）単純なアプリケーションのソースコードが含まれています。

```
<html>
    <head>
        <title>Encrypted Database Example (HTML)</title>
        <style type="text/css">
            body
            {
                padding-top: 25px;
                font-family: Verdana, Arial;
                font-size: 14px;
            }
            div
            {
                width: 85%;
                margin-left: auto;
                margin-right: auto;
            }
            .error {color: #990000}
            .success {color: #009900}
        </style>

        <script type="text/javascript" src="AIRAliases.js"></script>
        <script type="application/x-shockwave-flash" src="EncryptionKeyGenerator.swf"/>
        <script type="text/javascript">
            // set up the class shortcut
            var ekg;
            if (window.runtime)
            {
                if (!ekg) ekg = {};
                ekg.EncryptionKeyGenerator = window.runtime.com.adobe.air.crypto.EncryptionKeyGenerator;
            }

            // app globals
            var dbFileName = "encryptedDatabase.db";
            var dbFile;
            var createNewDB = true;
            var conn;

            // UI elements
            var instructions;
```

```
            var passwordInput;
            var openButton;
            var statusMsg;

            function init()
            {
                // UI elements
                instructions = document.getElementById("instructions");
                passwordInput = document.getElementById("passwordInput");
                openButton = document.getElementById("openButton");
                statusMsg = document.getElementById("statusMsg");

                conn = new air.SQLConnection();
                dbFile = air.File.applicationStorageDirectory.resolvePath(dbFileName);
                if (dbFile.exists)
                {
                    createNewDB = false;
                    instructions.innerHTML = "<p>Enter your database password to open the encrypted
database.</p>";
                    openButton.value = "Open Database";
                }
            }

            function openConnection()
            {
                var keyGenerator = new ekg.EncryptionKeyGenerator();

                var password = passwordInput.value;

                if (password == null || password.length <= 0)
                {
                    statusMsg.innerHTML = "<p class='error'>Please specify a password.</p>";
                    return;
                }

                if (!keyGenerator.validateStrongPassword(password))
                {
                    statusMsg.innerHTML = "<p class='error'>The password must be 8-32 characters long. It
must contain at least one lowercase letter, at least one uppercase letter, and at least one number or
symbol.</p>";
                    return;
                }

                passwordInput.value = "";
                passwordInput.disabled = true;
                openButton.disabled = true;
                statusMsg.innerHTML = "";

                var encryptionKey = keyGenerator.getEncryptionKey(password);

                conn.addEventListener(air.SQLEvent.OPEN, openHandler);
                conn.addEventListener(air.SQLErrorEvent.ERROR, openError);

                conn.openAsync(dbFile, air.SQLMode.CREATE, null, false, 1024, encryptionKey);
            }

            function openHandler(event)
            {
                conn.removeEventListener(air.SQLEvent.OPEN, openHandler);
                conn.removeEventListener(air.SQLErrorEvent.ERROR, openError);

                if (createNewDB)
```

```
                {
                    statusMsg.innerHTML = "<p class='success'>The encrypted database was created
successfully.</p>";
                }
                else
                {
                    statusMsg.innerHTML = "<p class='success'>The encrypted database was opened
successfully.</p>";
                }
            }

            function openError(event)
            {
                conn.removeEventListener(air.SQLEvent.OPEN, openHandler);
                conn.removeEventListener(air.SQLErrorEvent.ERROR, openError);

                if (!createNewDB && event.error.errorID ==
ekg.EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID)
                {
                    statusMsg.innerHTML = "<p class='error'>Incorrect password!</p>";
                }
                else
                {
                    statusMsg.innerHTML = "<p class='error'>Error creating or opening database.</p>";
                }
            }
        </script>
    </head>

    <body onload="init();">
        <div id="instructions"><p>Enter a password to create an encrypted database. The next time you open
the application, you will need to re-enter the password to open the database again.</p></div>
        <div><input id="passwordInput" type="password"/><input id="openButton" type="button" value="Create
Database" onclick="openConnection();"/></div>
        <div id="statusMsg"></div>
    </body>
</html>
```

EncryptionKeyGenerator クラスについて

Adobe AIR 1.5 およびそれ以降

EncryptionKeyGenerator クラスを使用してアプリケーションデータベースの安全な暗号化キーを作成するに当たり、このクラスの機能を理解することは必須ではありません。このクラスの使用方法については、237 ページの「EncryptionKeyGenerator クラスによる安全な暗号化キーの取得」で説明されています。ただし、このクラスが使用している技術を理解しておくと便利です。例えば、さまざまなレベルのデータプライバシーが要求される状況で、このクラスを適応させたり、その技術の一部を組み込むことができます。

EncryptionKeyGenerator クラスは、オープンソースの ActionScript 3.0 コアライブラリ（as3corelib）プロジェクトに含まれるクラスです。as3corelib パッケージ（ソースコードおよびドキュメントを含む）をダウンロードからダウンロードできます。また、プロジェクトサイトでソースコードを表示したり、説明に従ってダウンロードできます。

コードで EncryptionKeyGenerator インスタンスを作成し、その getEncryptionKey() メソッドを呼び出す場合、正当なユーザーのみがデータにアクセスできるようにするための手順がいくつか講じられています。この手順は、データベース作成前にユーザー入力パスワードから暗号化キーを生成したり、データベースを開くために暗号化キーを再作成する手順と同じです。

強力なパスワードの取得と検証
Adobe AIR 1.5 およびそれ以降

コードで getEncryptionKey() メソッドを呼び出す際には、パラメーターとしてパスワードを渡します。パスワードは、暗号化キーの土台として使用されます。ユーザーだけが知っている情報を使用することで、パスワードを知っているユーザーだけがデータベースのデータにアクセスできるように設計します。攻撃者がコンピューター上のユーザーのアカウントにアクセスできたとしても、パスワードを知らない限りはデータベースにアクセスできません。セキュリティを最大限に高めるために、アプリケーションではパスワードを保存しません。

アプリケーションのコードは、EncryptionKeyGenerator インスタンスを生成し、getEncryptionKey() メソッドを呼び出して、ユーザーによって入力されたパスワードを引数（この例の変数 password）として渡します。

```
var keyGenerator = new ekg.EncryptionKeyGenerator();
var encryptionKey = keyGenerator.getEncryptionKey(password);
```

EncryptionKeyGenerator クラスでは、getEncryptionKey() メソッドが呼び出されると、まずユーザーの入力したパスワードがパスワード強度の要件を満たしているかチェックします。EncryptionKeyGenerator クラスでは、パスワードの長さを 8 ～ 32 文字に設定する必要があります。パスワードは大文字と小文字を組み合わせたもので、数字または記号が少なくとも 1 文字含まれている必要があります。

内部的に、getEncryptionKey() メソッドは、EncryptionKeyGenerator クラスの validateStrongPassword() メソッドを呼び出し、パスワードが有効でない場合には例外をスローします。validateStrongPassword() メソッドはパブリックメソッドなので、getEncryptionKey() メソッドを呼び出さずにパスワードをチェックし、エラーの発生を回避できます。

パスワードを 256 ビットに拡張
Adobe AIR 1.5 およびそれ以降

プロセスの後半で、パスワードを 256 ビット長にする必要があります。次のコードでは、各ユーザーが 256 ビット（32 文字）長のパスワードを入力するよう要求する代わりに、パスワード文字を繰り返すことによって、長いパスワードを作成します。

concatenatePassword() メソッドのコードを次に示します。

パスワードの長さが 256 ビット未満である場合、パスワード自体を連結して 256 ビットにしています。ちょうどの長さにならない場合は、最後の繰り返しを短くして、ちょうど 256 ビットにします。

256 ビットのソルト値の生成と取得
Adobe AIR 1.5 およびそれ以降

次の手順では、256 ビットのソルト値を取得します。このソルト値は後の手順でパスワードに結合されます。ソルトは、ランダム値であり、ユーザーが入力した値に追加または結合されてパスワードを形成します。ソルトをパスワードと共に使用することにより、実際に存在する単語や一般的な言葉をユーザーがパスワードとして選択した場合でも、システムで使用されるパスワードとソルトの組み合わせはランダム値になります。このランダムな性質は、辞書攻撃（攻撃者が単語のリストを使用してパスワードを推測しようとする）からシステムを守るために役立ちます。また、ソルト値を生成し、暗号化されたローカルストアに保存することにより、データベースファイルが置かれているマシンのユーザーアカウントに関連付けられます。

アプリケーションが getEncryptionKey() メソッドを初めて呼び出す際には、このコードによってランダムな 256 ビットのソルト値が生成されます。それ以外の場合は、暗号化されたローカルストアからソルト値が読み込まれます。

XOR 演算子による 256 ビットのパスワードとソルトの結合

Adobe AIR 1.5 およびそれ以降

256 ビットのパスワードと 256 ビットのソルト値が用意できました。次は、ソルト値と連結パスワードとをビット XOR 演算で組み合わせ、1 つの値にします。これによって、文字データとして表現可能なあらゆる文字を含む可能性がある 256 ビットのパスワードを作成できます。ユーザーの入力するパスワードは主に英数字で構成されている場合がほとんどですが、そうであっても、ここで生成されるパスワードの性質は変わりません。ユーザーに長く複雑なパスワードの入力を要求しなくても、こうしてランダムな性質を付加すれば、パスワードの組み合わせの数を増やすことができます。

キーのハッシング

Adobe AIR 1.5 およびそれ以降

連結したパスワードとソルト値とを組み合わせた後は、さらにセキュリティを強化するために、SHA-256 ハッシュアルゴリズムを使用してこの値をハッシュします。値をハッシュすることにより、攻撃者がリバースエンジニアリングを行うのが困難になります。

ハッシュからの暗号化キーの抽出

Adobe AIR 1.5 およびそれ以降

暗号化キーは、16 バイト（128 ビット）ちょうどの長さの ByteArray でなければなりません。SHA-256 ハッシュアルゴリズムの結果は常に 256 ビット長になります。したがって、最後の手順では、実際の暗号化キーとして使用される 128 ビット分をハッシュ結果から選択します。

最初の 128 ビットは、暗号化キーとして使用する必要がありません。その他任意の場所から始まるビット範囲や、1 つおきに抽出したビット列、または何か別の方法で選択したビットを使用することもできます。重要なのは、128 個の別個のビットを選択すること、および毎回同じ 128 個のビットを使用することです。

SQL データベースの操作のための戦略

Adobe AIR 1.0 およびそれ以降

アプリケーションでローカル SQL データベースにアクセスして操作するには様々な方法があります。アプリケーションコードの構成や、操作を実行する順序やタイミングなどは、アプリケーションの設計によって様々です。選択する方法によって、アプリケーションの開発の容易さや、将来の更新でアプリケーションに変更を加えるときの容易さ、ユーザーから見たアプリケーションのパフォーマンスなどが違ってきます。

設定済みデータベースの配布

Adobe AIR 1.0 およびそれ以降

アプリケーションで AIR ローカル SQL データベースを使用する際には、テーブルや列などの特定の構造がデータベースに含まれている必要があります。さらに、アプリケーションによっては、データベースファイルに特定のデータがあらかじめ設定されている必要がある場合もあります。データベースに適切な構造が含まれるようにするには、まず、アプリケーションコードでデータベースを作成するという方法があります。データベースファイルが特定の場所にあるかどうかをアプリケーションの読み込み時にチェックして、ファイルが存在しない場合には一連のコマンドを実行してデータベースファイルを作成し、データベース構造を作成し、テーブルに初期データを設定します。

データベースとそのテーブルを作成するコードは複雑になることが多く、たいていはアプリケーションがインストールされている期間に 1 回しか使用されないにもかかわらず、アプリケーションのサイズと複雑さを増加させる要因になります。データベース、構造およびデータをプログラムで作成する代わりに、設定済みのデータベースをアプリケーションと一緒に配布することもできます。そのためには、データベースファイルをアプリケーションの AIR パッケージに含めます。

同梱されたデータベースファイルは、AIR パッケージに含まれているすべてのファイルと同様に、アプリケーションディレクトリ（File.applicationDirectory プロパティによって表されるディレクトリ）にインストールされます。ただし、このディレクトリのファイルは読み取り専用であるため、AIR パッケージのファイルを「テンプレート」データベースとして使用して、ユーザーが初めてアプリケーションを実行したときに元のデータベースファイルをユーザーの 147 ページの「アプリケーション記憶領域ディレクトリの参照」（または別の場所）にコピーし、そのデータベースをアプリケーション内で使用します。

ローカル SQL データベースの操作のベストプラクティス

Adobe AIR 1.0 およびそれ以降

ローカル SQL データベースを使用する際には、アプリケーションのパフォーマンス、セキュリティおよびメンテナンス性を向上させるために以下のテクニックを使用することをお勧めします。

データベース接続を事前に作成する

Adobe AIR 1.0 およびそれ以降

アプリケーションの最初の読み込み時に実行するステートメントがなくても、ステートメントの実行時に遅れが生じないように、事前に（アプリケーションの起動後など）SQLConnection オブジェクトをインスタンス化してその open() メソッドまたは openAsync() メソッドを呼び出します。208 ページの「データベースへの接続」を参照してください。

データベース接続を再利用する

Adobe AIR 1.0 およびそれ以降

アプリケーションの実行期間全体にわたってアクセスするデータベースがある場合は、接続を閉じたり開いたりする代わりに、SQLConnection インスタンスへの参照を保持してアプリケーション全体で再利用します。208 ページの「データベースへの接続」を参照してください。

非同期実行モードを優先的に使用する

Adobe AIR 1.0 およびそれ以降

データアクセスコードを記述する際には、同期操作を使用した方がコードが短く単純になることが多いため、操作を非同期ではなく同期的に実行したくなるかもしれませんが、226 ページの「同期および非同期のデータベース操作の使用」で説明したように、同期操作はユーザーの体感的なパフォーマンスに影響を与え、アプリケーションのユーザーエクスペリエンスを低下させる可能性があります。1 つの操作にかかる時間は操作（特に、操作に含まれるデータの量）によって変わります。例えば、データベースにデータを 1 行だけ追加する SQL INSERT ステートメントにかかる時間は、数千行のデータを取得する SELECT ステートメントより短くなります。しかし、同期実行を使用して複数の操作を実行すると、通常はそれらの操作が一続きで実行されます。1 つ 1 つの操作にかかる時間は非常に短くても、すべての同期操作が完了するまでアプリケーションが応答しなくなります。その結果、一続きで実行される複数の操作の時間が累積されて、アプリケーションの動作が停止する場合もあります。

非同期操作を標準のアプローチとして使用します。多数の行を含む操作では特にそうするようにしてください。SELECT ステートメントの結果が大きい場合にその処理を分割する方法もありますが（219 ページの「SELECT の結果の分割」を参照）、非同期実行モードでしか使用できません。同期操作を使用するのは、非同期プログラミングでは実現できない機能があ

る場合のみにしてください。その場合も、アプリケーションのユーザーへのパフォーマンスの影響を考慮し、アプリケーションをテストしてアプリケーションのパフォーマンスへの影響を確認する必要があります。非同期実行を使用するとコードが複雑になる場合もありますが、コードを記述するのは 1 回だけで済むのに対し、アプリケーションのユーザーはそれを（速くても遅くても）繰り返し使用しなければならないことを忘れないでください。

多くの場合は、実行する SQL ステートメントごとに個別の SQLStatement インスタンスを使用することで、複数の SQL 操作を一度にキューに登録できます。これにより、非同期コードを同期コードのように記述することができます。詳しくは、230 ページの「非同期実行モデルについて」を参照してください。

SQLStatement の text プロパティを変更せずに別の SQL ステートメントを使用する

Adobe AIR 1.0 およびそれ以降

アプリケーション内で複数回実行される SQL ステートメントがある場合は、SQL ステートメントごとに個別の SQLStatement インスタンスを作成し、その SQL コマンドを実行するたびにその SQLStatement インスタンスを使用します。例えば、作成するアプリケーションに何度も実行される 4 つの異なる SQL 操作が含まれている場合は、4 つの SQLStatement インスタンスを作成し、各ステートメントの execute() メソッドを呼び出してステートメントを実行します。すべての SQL ステートメントに対して 1 つの SQLStatement インスタンスを使用し、ステートメントを実行する前に毎回その text プロパティを再定義するという方法は避けてください。

ステートメントパラメーターを使用する

Adobe AIR 1.0 およびそれ以降

ユーザー入力をステートメントテキストに連結する代わりに SQLStatement のパラメーターを使用します。パラメーターを使用すると、SQL インジェクション攻撃が行われる可能性がなくなるため、アプリケーションのセキュリティが向上します。クエリで（SQL リテラル値だけでなく）オブジェクトを使用できるようにもなります。さらに、ステートメントを実行のたびに再コンパイルせずに再利用できるため、実行効率も向上します。詳しくは、211 ページの「ステートメント内でのパラメーターの使用」を参照してください。

第 15 章：暗号化されたローカルストア

Adobe® AIR® ランタイムは、ユーザーのコンピューターにインストールされた各 AIR アプリケーションに対して永続的に暗号化されたローカルストア（ELS）を提供します。これにより、他のユーザーによって簡単に解読されないように暗号化された形式で、ユーザーのローカルハードディスクに格納されたデータを保存および取得できるようになります。各 AIR アプリケーションに対して個別の暗号化されたローカルストアが使用され、各 AIR アプリケーションは各ユーザーに対して個別の暗号化されたローカルストアを使用します。

注意：暗号化されたローカルストアが用意されているだけでなく、AIR は、SQL データベースに格納されたコンテンツの暗号化も提供します。詳しくは、231 ページの「SQL データベースでの暗号化の使用」を参照してください。

Web サービスに対するログイン資格情報など、セキュリティで保護する必要がある情報をキャッシュするために暗号化されたローカルストアを使用する必要がある場合があります。他のユーザーに対して非公開にする必要がある情報を保存するには、ELS が適切です。ただし、ELS では同じユーザーアカウントで実行される他のプロセスからデータを保護することはできません。このため、DRM や暗号化キーなどの機密アプリケーションデータを保護するには適切でありません。

デスクトッププラットフォームにおいて、AIR では、Windows は DPAPI、Mac OS は KeyChain、Linux は KeyRing または KWallet を使用して、暗号化されたローカルストアを各アプリケーションおよびユーザーに関連付けます。暗号化されたローカルストアは AES-CBC 128 ビット暗号化を使用します。

Android では、EncryptedLocalStorage クラスによって保存されたデータは暗号化されません。代わりに、データはオペレーティングシステムに用意されているユーザーレベルのセキュリティによって保護されます。Android オペレーティングシステムはすべてのアプリケーションに個別のユーザー ID を割り当てます。アプリケーションがアクセスできるのは、自身のファイルと、パブリックロケーション（リムーバブルストレージカードなど）で作成されたファイルだけです。「ルート化された」Android デバイス上では、ルート権限で実行しているアプリケーションは、他のアプリケーションのファイルにアクセス可能です。したがって、ルート化されたデバイスでは、ルート化されていないデバイスで確保される暗号化されたローカルストアの高いデータ保護レベルが実現されません。

暗号化されたローカルストア内の情報は、アプリケーションセキュリティサンドボックス内の AIR アプリケーションコンテンツでのみ使用できます。

AIR アプリケーションを更新すると、更新されたバージョンには暗号化されたローカルストアの既存のデータへのアクセス権が保持されます。ただし、次の場合を除きます。

- stronglyBound パラメーターを true に設定してアイテムが追加された場合。
- 既存バージョンと更新バージョンが両方とも AIR 1.5.3 以前にパブリッシュされていて、更新が移行署名で署名されている場合。

暗号化されたローカルストアの制限事項

暗号化されたローカルストアのデータは、ユーザーのオペレーティングシステムアカウントの資格情報で保護されています。他のエンティティは、そのユーザーとしてログインできる場合を除き、ストア内のデータにアクセスできません。ただし、認証されたユーザーによって実行される他のアプリケーションからのアクセスに対しては、データのセキュリティは保護されません。

このような攻撃を機能させるにはユーザーが認証される必要があるので、ユーザーのプライベートデータは保護されます（ユーザーアカウント自体が偽装されない限り）。ただし、ライセンスやデジタル権限管理用のキーなど、アプリケーションでユーザーに公開しない可能性のあるデータは、保護されません。したがって、ELS はそのような情報を格納するのに適切な場所ではありません。ELS は、パスワードなどのユーザーのプライベートデータを保存する目的にのみ適しています。

ELS 内のデータは、様々な理由で失われる場合があります。例えば、ユーザーがアプリケーションをアンインストールしたり、暗号化されたデータを削除したりする場合があります。また、更新の結果、発行者 ID が変更される場合があります。このように、ELS は永続的なデータ記憶領域ではなく、プライベートキャッシュとして扱う必要があります。

stronglyBound パラメーターはサポートされなくなったため、true に設定しないでください。このパラメーターを true に設定しても、追加のデータ保護は提供されません。同時に、発行者 ID が同じでもアプリケーションが更新されるとデータへのアクセスが失われます。

格納されたデータが 10MB を超えると、暗号化されたローカルストアの処理が遅くなる場合があります。

AIR アプリケーションをアンインストールするときは、アンインストーラーは暗号化されたローカルストアに格納されたデータを削除しません。

ELS を使用するためのベストプラクティスには次のようなものがあります。

- パスワードなどのユーザーの機密データを保存する目的で ELS を使用します（stronglyBound を false に設定）。
- DRM キーやライセンスのトークンなどのアプリケーションの機密情報を保存する目的では ELS を使用しないでください。
- ELS データが失われた場合に、ELS に保存されたデータをアプリケーションで作成し直す方法を提供します。例えば、必要に応じて、ユーザーにアカウント資格情報の再入力を求めます。
- stronglyBound パラメーターは使用しないでください。
- stronglyBound を true に設定する場合は、更新中に保存されたアイテムを移行しないでください。代わりに、更新後にデータを再作成します。
- 比較的少量のデータのみ保存します。それより大きな量のデータに対しては、暗号化を適用した AIR SQL データベースを使用します。

関連項目

flash.data.EncryptedLocalStore

暗号化されたローカルストアへのデータの追加

EncryptedLocalStore クラスの setItem() 静的メソッドを使用して、データをローカルストアに格納します。データは、ストリングをキーとして使用し、バイト配列として格納されたデータと共に、ハッシュテーブルに格納されます。

例えば、次のコードは暗号化されたローカルストアにストリングを格納します。

```
var str = "Bob";
var bytes = new air.ByteArray();
bytes.writeUTFBytes(str);
air.EncryptedLocalStore.setItem("firstName", bytes);
```

setItem() メソッドの 3 番目のパラメーターである stronglyBound パラメーターはオプションです。このパラメーターを true に設定すると、暗号化されたローカルストアは、格納している AIR アプリケーションのデジタル署名とビット列に、格納されたアイテムをバインドします。

```
var str = "Bob";
var bytes = new air.ByteArray();
bytes.writeUTFBytes(str);
air.EncryptedLocalStore.setItem("firstName", bytes, false);
```

true に設定された stronglyBound を使用して格納されたアイテムの場合、getItem() に対する後続の呼び出しは、呼び出しを行っている AIR アプリケーションが格納を行っているアプリケーションと同一である場合（アプリケーションディレクトリ内のファイル内のデータが変更されていない場合）にのみ成功します。呼び出しを行っている AIR アプリケーションが格納を行っているアプリケーションと異なる場合、強力にバインドされたアイテムに対する getItem() を呼び出したときに、アプリケーションは Error 例外をスローします。アプリケーションをアップデートした場合、暗号化されたローカルストアに以前書き込まれた、強力にバインドされたデータを読み取ることはできません。モバイルデバイスでは stronglyBound を true に設定しても無視され、このパラメーターは常に false として扱われます。

stronglyBound パラメーターが false （既定）に設定されている場合、アプリケーションがデータを読み取るためには、発行者 ID のみ同じままである必要があります。アプリケーションのビット列は変化してもよく（同じ発行者による署名が必要です）、データを格納したアプリケーションのビット列とまったく同じである必要はありません。元のアプリケーションと同じ発行者 ID を持つアップデートされたアプリケーションは、データに引き続きアクセスできます。

注意：実際には、stronglyBound を true に設定すると、新たなデータ保護は追加されません。このため、悪意のあるユーザーがアプリケーションを改ざんして ELS に格納されているアイテムにアクセスする可能性が残ります。さらに、データは stronglyBound を true 設定するか false に設定するかにかかわらず、外部の、ユーザー以外の脅威から同様に強力に保護されます。こうした理由から、stronglyBound を true に設定することはお勧めしません。

暗号化されたローカルストアのデータへのアクセス

Adobe AIR 1.0 およびそれ以降

次の例のように EncryptedLocalStore.getItem() メソッドを使用すると、暗号化されたローカルストアから値を取得できます。

```
var storedValue = air.EncryptedLocalStore.getItem("firstName");
air.trace(storedValue.readUTFBytes(storedValue.length)); // "foo"
```

暗号化されたローカルストアからのデータの削除

Adobe AIR 1.0 およびそれ以降

次の例のように、EncryptedLocalStore.removeItem() メソッドを使用して、暗号化されたローカルストアから値を削除できます。

```
air.EncryptedLocalStore.removeItem("firstName");
```

次の例のように、EncryptedLocalStore.reset() メソッドを呼び出すことによって、暗号化されたローカルストアからすべてのデータを消去できます。

```
air.EncryptedLocalStore.reset();
```

第 16 章：バイト配列の操作

Flash Player 9 以降、Adobe AIR 1.0 以降

ByteArray クラスを使用すると、データのバイナリストリームを読み書きできます。バイナリストリームは基本的にバイトの配列です。このクラスは、最も基本的なレベルのデータにアクセスするための手段を提供します。コンピューターのデータはバイト（8 ビットのグループ）で構成されるので、バイト単位でデータを読み取れるということは、クラスとアクセスメソッドが存在しないデータにアクセスできることを意味します。ByteArray クラスを使用すると、ビットマップから、ネットワーク経由で伝送されるデータストリームに至るまで、任意のデータストリームをバイトレベルで解析できます。

writeObject() メソッドを使用すると、オブジェクトを直列化された Action Message Format（AMF）で ByteArray に書き込むことができます。一方、readObject() メソッドを使用すると、直列化されたオブジェクトを ByteArray から元のデータ型の変数に読み取ることができます。表示オブジェクトを除くすべてのオブジェクトを直列化できます。表示オブジェクトは、表示リストに配置できるオブジェクトです。カスタムクラスをランタイムに使用できる場合は、直列化されたオブジェクトをカスタムクラスインスタンスに割り当てて戻すこともできます。オブジェクトを AMF に変換した後は、ネットワーク接続を通して効率よく転送したり、ファイルに保存したりできます。

ここで説明する Adobe® AIR® のサンプルアプリケーションでは、バイトストリーム処理の例として .zip ファイルを読み取り、.zip ファイルに含まれるファイルのリストを抽出して、それをデスクトップに書き込みます。

関連項目

flash.utils.ByteArray

flash.utils.IExternalizable

Action Message Format 仕様

ByteArray の読み取りと書き込み

Flash Player 9 以降、Adobe AIR 1.0 以降

ByteArray クラスは flash.utils パッケージの一部です。コードに AIRAliases.js ファイルが含まれている場合は、エイリアス air.ByteArray を使用して ByteArray クラスを参照することもできます。ByteArray を作成するには、次の例に示すように ByteArray コンストラクターを呼び出します。

```
var stream = new air.ByteArray();
```

ByteArray のメソッド

Flash Player 9 以降、Adobe AIR 1.0 以降

意味のあるデータストリームはすべて、分析して目的の情報を発見できるような形式に編成されています。例えば、単純な従業員ファイルのレコードには通常、ID 番号、氏名、住所、電話番号などが含まれます。MP3 オーディオファイルには、ダウンロードされているファイルのタイトル、作成者、アルバム、発行日、およびジャンルを示す ID3 タグが含まれます。一定の形式を使用することで、データストリーム上のデータの予想される順序がわかります。これにより、データストリームをインテリジェントに読み取ることができます。

ByteArray クラスには、データストリームの読み書きを容易にするメソッドがいくつか用意されています。例えば、readBytes() と writeBytes()、readInt() と writeInt()、readFloat() と writeFloat()、readObject() と writeObject()、readUTFBytes() と writeUTFBytes() などがあります。これらのメソッドを使用することで、データストリームから特定のデータ型の変数にデータを読み取ったり、特定のデータ型からバイナリデータストリームに直接書き込んだりすることができます。

例えば、次のコードは、ストリングと浮動小数点数の単純な配列を読み取り、各エレメントを ByteArray に書き込みます。この配列は、コードから適切な ByteArray メソッド（writeUTFBytes() および writeFloat()）を呼び出してデータを書き込む処理に適した編成になっています。同じデータパターンの繰り返しなので、ループを使用して配列を読み取ることができます。

```
// The following example reads a simple Array (groceries), made up of strings
// and floating-point numbers, and writes it to a ByteArray.

// define the grocery list Array
var groceries = ["milk", 4.50, "soup", 1.79, "eggs", 3.19, "bread" , 2.35]
// define the ByteArray
var bytes = new air.ByteArray();
// for each item in the array
for (i = 0; i < groceries.length; i++) {
        bytes.writeUTFBytes(groceries[i++]); //write the string and position to the next item
        bytes.writeFloat(groceries[i]);// write the float
        air.trace("bytes.position is: " + bytes.position); //display the position in ByteArray
}
air.trace("bytes length is: " +  bytes.length);// display the length
```

position プロパティ

Flash Player 9 以降、Adobe AIR 1.0 以降

position プロパティには、読み取り中または書き込み中の ByteArray のインデックスを示すポインターの現在位置が格納されます。次のコードに示すように、position プロパティの初期値は 0（ゼロ）です。

```
var bytes = new air.ByteArray();
air.trace("bytes.position is initially: " + bytes.position); // 0
```

ByteArray の読み書きを行うと、使用したメソッドによって position プロパティが更新され、最後に読み書きしたバイトの直後の位置を指すようになります。例えば次のコードでは、ストリングを ByteArray に書き込んだ後、position プロパティは ByteArray 内のストリングの直後のバイトを指すようになります。

```
var bytes = new air.ByteArray();
air.trace("bytes.position is initially: " + bytes.position); // 0
bytes.writeUTFBytes("Hello World!");
air.trace("bytes.position is now: " + bytes.position);// 12
```

同様に、読み取り操作を行うと、読み取ったバイト数だけ position プロパティがインクリメントされます。

```
var bytes = new air.ByteArray();

air.trace("bytes.position is initially: " + bytes.position); // 0
bytes.writeUTFBytes("Hello World!");
air.trace("bytes.position is now: " + bytes.position);// 12
bytes.position = 0;
air.trace("The first 6 bytes are: " + (bytes.readUTFBytes(6)));//Hello
air.trace("And the next 6 bytes are: " + (bytes.readUTFBytes(6)));// World!
```

position プロパティを ByteArray の特定の位置に設定することで、そのオフセットを読み書きできます。

bytesAvailable プロパティと length プロパティ

Flash Player 9 以降、Adobe AIR 1.0 以降

length プロパティと bytesAvailable プロパティは、ByteArray の長さ、および現在位置から終端までの間に残っているバイト数を示します。次の例では、これらのプロパティの使用方法を示します。この例では、テキストのストリングを ByteArray に書き込んだ後、ByteArray を一度に 1 バイトずつ読み取り、文字「a」または終端（bytesAvailable <= 0）を検出すると終了します。

```
var bytes = new air.ByteArray();
var text = "Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Vivamus etc.";

bytes.writeUTFBytes(text); // write the text to the ByteArray
air.trace("The length of the ByteArray is: " + bytes.length);// 70
bytes.position = 0; // reset position
while (bytes.bytesAvailable > 0 && (bytes.readUTFBytes(1) != 'a')) {
    //read to letter a or end of bytes
}
if (bytes.position < bytes.bytesAvailable) {
    air.trace("Found the letter a; position is: " + bytes.position); // 23
    air.trace("and the number of bytes available is: " + bytes.bytesAvailable);// 47
}
```

endian プロパティ

Flash Player 9 以降、Adobe AIR 1.0 以降

マルチバイトの数値、つまり格納に必要なメモリが 1 バイトより多い数値の格納方法は、コンピューターによって異なる場合があります。例えば、整数の格納に 4 バイト（32 ビット）のメモリを使用するとします。あるコンピューターでは数値の最上位バイトが最初に（最も小さいメモリアドレスに）格納され、別のコンピューターでは最下位バイトが最初に格納されます。このようなコンピューターの属性、すなわちバイト順序の属性は、ビッグエンディアン（最上位バイトが最初）またはリトルエンディアン（最下位バイトが最初）と呼ばれます。例えば、0x31323334 という数値は、バイト順序がビッグエンディアンかリトルエンディアンかに応じて次のように格納されます。a0 は 4 バイトのうちの最小メモリアドレスを表し、a3 は最大メモリアドレスを表します。

ビッグエンディアン	ビッグエンディアン	ビッグエンディアン	ビッグエンディアン
a0	a1	a2	a3
31	32	33	34

リトルエンディアン	リトルエンディアン	リトルエンディアン	リトルエンディアン
a0	a1	a2	a3
34	33	32	31

ByteArray クラスの endian プロパティを使用すると、マルチバイト数値を処理する際のバイト順序を指定することができます。このプロパティに設定できる値は「bigEndian」または「littleEndian」で、Endian クラスでは endian プロパティを設定するための定数として BIG_ENDIAN と LITTLE_ENDIAN が定義されています。

compress() メソッドと uncompress() メソッド

Flash Player 9 以降、Adobe AIR 1.0 以降

compress() メソッドを使用すると、パラメーターとして指定する圧縮アルゴリズムに従って、ByteArray を圧縮できます。uncompress() メソッドを使用すると、圧縮アルゴリズムに従って、圧縮されている ByteArray を圧縮解除できます。compress() および uncompress() の呼び出し後、バイト配列の長さは新しい長さに設定され、position プロパティは終端に設定されます。

CompressionAlgorithm クラス（AIR）で定義されている定数を使用して、圧縮アルゴリズムを指定できます。ByteArray クラスでは、deflate アルゴリズム（AIR のみ）と zlib アルゴリズムの両方をサポートしています。deflate 圧縮アルゴリズムは、zlib、gzip および一部の zip 実装など、いくつかの圧縮形式で使用されています。zlib で圧縮されたデータの形式については、http://www.ietf.org/rfc/rfc1950.txt を参照してください。deflate 圧縮アルゴリズムについては、http://www.ietf.org/rfc/rfc1951.txt を参照してください。

次の例では、bytes という名前の ByteArray を deflate アルゴリズムで圧縮しています。

```
bytes.compress(air.CompressionAlgorithm.DEFLATE);
```

次の例では、deflate アルゴリズムで圧縮された ByteArray の圧縮を解除しています。

```
bytes.uncompress(CompressionAlgorithm.DEFLATE);
```

オブジェクトの読み取りと書き込み

Flash Player 9 以降、Adobe AIR 1.0 以降

readObject() メソッドと writeObject() メソッドは、直列化された Action Message Format （AMF）でエンコードされた ByteArray のオブジェクトを読み取ったり、書き込んだりします。AMF はアドビ システムズ社が開発した独自のメッセージプロトコルであり、Netstream、NetConnection、NetStream、LocalConnection、Shared Objects など、様々な ActionScript 3.0 クラスで使用されています。

1 バイト型のマーカーが、その後に続くエンコードされたデータの型を示します。AMF では、次の 13 のデータ型が使用されます。

```
value-type = undefined-marker | null-marker | false-marker | true-marker | integer-type |
    double-type | string-type | xml-doc-type | date-type | array-type | object-type |
    xml-type | byte-array-type
```

型マーカーの後には、エンコードされたデータが続きます。ただし、マーカーが null や true または false などの単一の可能な値を表す場合は例外です。このようなマーカーの場合、他には何もエンコードされません。

AMF には、AMF0 と AMF3 という 2 つのバージョンがあります。AMF 0 では複雑なオブジェクトの参照渡しがサポートされ、エンドポイントでオブジェクトの関係を復元することができます。AMF 3 は AMF 0 の改良版であり、オブジェクトの参照に加えてオブジェクトの特性とストリングを渡し、ActionScript 3.0 で導入された新しいデータ型をサポートします。オブジェクトデータのエンコードに使用する AMF のバージョンは、ByteArray.objectEcoding プロパティで指定します。flash.net.ObjectEncoding クラスでは、AMF のバージョンを指定するための定数として、ObjectEncoding.AMF0 と ObjectEncoding.AMF3 が定義されています。

次の例では、writeObject() を呼び出して XML オブジェクトを ByteArray に書き込んだ後、デスクトップの order ファイルに書き込んでいます。この例では、終了時に「Wrote order file to desktop!」というメッセージを AIR ウィンドウに表示します。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd"><html xmlns="http://www.w3.org/1999/xhtml">
<head>
<style type="text/css">
        #taFiles
        {
            border: 1px solid black;
            font-family: Courier, monospace;
            white-space: pre;
            width: 95%;
            height: 95%;
            overflow-y: scroll;
        }
</style>
<script type="text/javascript" src="AIRAliases.js" ></script>
<script type="text/javascript">

//define ByteArray
var inBytes = new air.ByteArray();
//add objectEncoding value and file heading to output text
var output = "Object encoding is: " + inBytes.objectEncoding + "\n\n" + "order file: \n\n";

function init() {

    readFile("order", inBytes);
    inBytes.position = 0;//reset position to beginning
    // read XML from ByteArray
    var orderXML = inBytes.readObject();
    // convert to XML Document object
    var myXML = (new DOMParser()).parseFromString(orderXML, "text/xml");
    document.write(myXML.getElementsByTagName("menuName")[0].childNodes[0].nodeValue + ": ");
       document.write(myXML.getElementsByTagName("price")[0].childNodes[0].nodeValue + "<br/>");      //
burger: 3.95
       document.write(myXML.getElementsByTagName("menuName")[1].childNodes[0].nodeValue + ": ");
       document.write(myXML.getElementsByTagName("price")[1].childNodes[0].nodeValue + "<br/>");       //
fries: 1.45
}   // end of init()

// read specified file into byte array
function readFile(fileName, data) {
    var inFile = air.File.desktopDirectory; // source folder is desktop
    inFile = inFile.resolvePath(fileName);  // name of file to read
    var inStream = new air.FileStream();
    inStream.open(inFile, air.FileMode.READ);
    inStream.readBytes(data, 0, data.length);
    inStream.close();
}
</script>
</head>

<body onload = "init();">
        <div id="taFiles"></div>
</body>
</html>
```

readObject() メソッドは、直列化された AMF のオブジェクトを ByteArray から読み取り、指定された型のオブジェクトに格納します。次の例では、order ファイルをデスクトップから ByteArray（inBytes）に読み取り、readObject() を呼び出して orderXML に格納した後、XML オブジェクトドキュメント myXML に変換して、2 つのアイテムエレメントと価格エレメントの値を表示しています。また、objectEncoding プロパティの値を、order ファイルのコンテンツのヘッダーと共に表示しています。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<style type="text/css">
        #taFiles
        {
            border: 1px solid black;
            font-family: Courier, monospace;
            white-space: pre;
            width: 95%;
            height: 95%;
            overflow-y: scroll;
        }
    </style>
<script type="text/javascript" src="AIRAliases.js" ></script>
<script type="text/javascript">

//define ByteArray
var inBytes = new air.ByteArray();
//add objectEncoding value and file heading to output text
var output = "Object encoding is: " + inBytes.objectEncoding + "<br/><br/>" + "order file items:" +
"<br/><br/>";

function init() {

    readFile("order", inBytes);
    inBytes.position = 0;//reset position to beginning
    // read XML from ByteArray
    var orderXML = inBytes.readObject();
    // convert to XML Document object
    var myXML = (new DOMParser()).parseFromString(orderXML, "text/xml");
    document.write(output);
    document.write(myXML.getElementsByTagName("menuName")[0].childNodes[0].nodeValue + ": ");
       document.write(myXML.getElementsByTagName("price")[0].childNodes[0].nodeValue + "<br/>");      //
burger: 3.95
       document.write(myXML.getElementsByTagName("menuName")[1].childNodes[0].nodeValue + ": ");
       document.write(myXML.getElementsByTagName("price")[1].childNodes[0].nodeValue + "<br/>");       //
fries: 1.45
}   // end of init()


// read specified file into byte array
function readFile(fileName, data) {
    var inFile = air.File.desktopDirectory; // source folder is desktop
    inFile = inFile.resolvePath(fileName);  // name of file to read
    var inStream = new air.FileStream();
    inStream.open(inFile, air.FileMode.READ);
    inStream.readBytes(data, 0, data.length);
    inStream.close();
}
</script>
</head>

<body onload = "init();">
        <div id="taFiles"></div>
</body>
</html>
```

ByteArray の例：.zip ファイルの読み取り

Adobe AIR 1.0 およびそれ以降

この例では、種類の異なる複数のファイルを含む単純な .zip ファイルを読み取る方法を示します。その手段として、ここでは各ファイルに関連するデータをメタデータから抽出し、ファイルごとに ByteArray に圧縮解除して、ファイルをデスクトップに書き込んでいます。

.zip ファイルの一般的な構造は、http://www.pkware.com/documents/casestudies/APPNOTE.TXT に記載されている PKWARE Inc. の仕様に基づきます。最初にあるのは .zip アーカイブ内の最初のファイルのファイルヘッダーとファイルデータで、その後に、他の各ファイルのファイルヘッダーとファイルデータのペアが続きます。ファイルヘッダーの構造については後で説明します。.zip ファイルには次に、必要に応じてデータ記述子レコードが含まれます（通常は、出力 zip ファイルがディスクに保存されずメモリに作成された場合がこれに当たります）。次に、アーカイブ復号化ヘッダー、アーカイブ追加データレコード、中央ディレクトリ構造、Zip64 中央ディレクトリ終了レコード、Zip64 中央ディレクトリ終了ロケーター、中央ディレクトリ終了レコードなど、いくつかの追加オプションエレメントがあります。

この例のコードは、フォルダーを含まない zip ファイルのみを解析するように作られており、データ記述子レコードは想定していません。最後のファイルデータの後にある情報はすべて無視します。

各ファイルのファイルヘッダーの形式は次のとおりです。

ファイルヘッダーシグネチャ	4 バイト
必要なバージョン	2 バイト
汎用ビットフラグ	2 バイト
圧縮方法	2 バイト（8 = DEFLATE、0 = UNCOMPRESSED）
最終ファイル更新時刻	2 バイト
最終ファイル更新日	2 バイト
crc-32	4 バイト
圧縮サイズ	4 バイト
非圧縮サイズ	4 バイト
ファイル名の長さ	2 バイト
追加フィールドの長さ	2 バイト
ファイル名	variable
追加フィールド	variable

ファイルヘッダーの後には実際のファイルデータがあり、圧縮方法フラグに応じて、圧縮されている場合とされていない場合があります。フラグの値は、ファイルデータが圧縮されていない場合は 0（ゼロ）、データが DEFLATE アルゴリズムで圧縮されている場合は 8、他の圧縮アルゴリズムの場合は別の値になります。

この例のユーザーインターフェイスは、ラベルとテキスト領域（taFiles）で構成されています。アプリケーションは、.zip ファイルで検出したファイルごとに、ファイル名、圧縮サイズおよび非圧縮サイズの情報をテキスト領域に書き込みます。次の MXML ドキュメントは、Flex バージョンのアプリケーションのユーザーインターフェイスの定義です。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
creationComplete="init();">
    <mx:Script>
        <![CDATA[
            // The application code goes here
        ]]>
    </mx:Script>
    <mx:Form>
        <mx:FormItem label="Output">
            <mx:TextArea id="taFiles" width="320" height="150"/>
        </mx:FormItem>
    </mx:Form>
</mx:WindowedApplication>
```

この例のユーザーインターフェイスは、ラベルとテキスト領域（taFiles）で構成されています。アプリケーションは、.zip ファイルで検出したファイルごとに、ファイル名、圧縮サイズおよび非圧縮サイズの情報をテキスト領域に書き込みます。次の HTML ページは、アプリケーションのユーザーインターフェイスの定義です。

```
<html>
    <head>
        <style type="text/css">
            #taFiles
            {
                border: 1px solid black;
                font-family: Courier, monospace;
                white-space: pre;
                width: 95%;
                height: 95%;
                overflow-y: scroll;
            }
        </style>
        <script type="text/javascript" src="AIRAliases.js"></script>
        <script type="text/javascript">
            // The application code goes here
        </script>
    </head>
    <body onload="init();">
        <div id="taFiles"></div>
    </body>
</html>
```

プログラムの最初で、次のタスクを実行します。

- ByteArray の bytes を定義します。

```
var bytes = new air.ByteArray();
```

- ファイルヘッダーのメタデータを格納する変数を定義します。

```
// variables for reading fixed portion of file header
var fileName = new String();
var flNameLength;
var xfldLength;
var offset;
var compSize;
var uncompSize;
var compMethod;
var signature;

var output;
```

- .zip ファイルを表す File（zfile）オブジェクトと FileStream（zStream）オブジェクトを定義し、展開するファイルが含まれている .zip ファイルの場所（デスクトップディレクトリの「HelloAIR.zip」という名前のファイル）を指定します。

```
// File variables for accessing .zip file
var zfile = air.File.desktopDirectory.resolvePath("HelloAIR.zip");
var zStream = new air.FileStream();
```

Flex の場合、プログラムのコードは init() メソッドで開始し、このメソッドは、ルート mx:WindowedApplication タグの creationComplete ハンドラーとして呼び出されます。

プログラムのコードは init() メソッドで開始し、このメソッドは、body タグの onload イベントハンドラーとして呼び出されます。

```
function init()
{
```

プログラムは、.zip ファイルを READ モードで開くことにより開始します。

```
    zStream.open(zfile, air.FileMode.READ);
```

次に、bytes の endian プロパティを LITTLE_ENDIAN に設定し、数値フィールドのバイト順序は最下位バイトが最初であることを示します。

```
    bytes.endian = air.Endian.LITTLE_ENDIAN;
```

次に、while() ステートメントでループを開始し、ファイルストリーム内の現在位置がファイルのサイズ以上になるまで続けます。

```
    while (zStream.position < zfile.size)
    {
```

ループ内部の最初のステートメントでは、ファイルストリームの最初の 30 バイトを ByteArray の bytes に読み取ります。最初の 30 バイトは、最初のファイルヘッダーの固定サイズ部分を構成します。

```
        // read fixed metadata portion of local file header
        zStream.readBytes(bytes, 0, 30);
```

次に、コードは、30 バイトのヘッダーの最初のバイトから整数（signature）を読み取ります。ZIP 形式の定義では、すべてのファイルヘッダーのシグネチャは 16 進数値 0x04034b50 と指定されています。シグネチャが異なる場合は、コードが .zip ファイルのファイル部分を超えており、展開するファイルがそれ以上ないことを意味します。その場合は、コードはバイト配列の終端を待つことなく、直ちに while ループを終了します。

```
        bytes.position = 0;
        signature = bytes.readInt();
        // if no longer reading data files, quit
        if (signature != 0x04034b50)
        {
            break;
        }
```

コードの次の部分では、オフセット位置 8 のヘッダーバイトを読み取って、値を変数 compMethod に格納します。このバイトには、そのファイルの圧縮時に使用された圧縮方法を示す値が含まれます。複数の圧縮方法を使用できますが、実際には、ほとんどすべての .zip ファイルで DEFLATE 圧縮アルゴリズムが使用されます。現在のファイルが DEFLATE 圧縮アルゴリズムで圧縮されている場合は、compMethod は 8 です。ファイルが圧縮されていない場合は、compMethod は 0 です。

```
        bytes.position = 8;
        compMethod = bytes.readByte();  // store compression method (8 == Deflate)
```

最初の 30 バイトの後にはヘッダーの可変長部分があり、ファイル名と、場合によっては追加フィールドが含まれます。変数 offset に、この部分のサイズが格納されています。サイズはファイル名の長さと追加フィールドの長さを加えたもので、これらの値はヘッダーのオフセット 26 および 28 から読み取ります。

```
offset = 0;// stores length of variable portion of metadata
bytes.position = 26;  // offset to file name length
flNameLength = bytes.readShort();// store file name
offset += flNameLength; // add length of file name
bytes.position = 28;// offset to extra field length
xfldLength = bytes.readShort();
offset += xfldLength;// add length of extra field
```

次に、プログラムは、offset 変数に格納されているバイト数だけ、ファイルヘッダーの可変長部分を読み取ります。

```
// read variable length bytes between fixed-length header and compressed file data
zStream.readBytes(bytes, 30, offset);
```

そして、ヘッダーの可変長部分からファイル名を読み取り、ファイルの圧縮サイズ（zip サイズ）と非圧縮サイズ（元のサイズ）と共にテキスト領域に表示します。

```
bytes.position = 30;
fileName = bytes.readUTFBytes(flNameLength); // read file name
output += fileName + "<br />"; // write file name to text area
bytes.position = 18;
compSize = bytes.readUnsignedInt();  // store size of compressed portion
output += "\tCompressed size is: " + compSize + '<br />';
bytes.position = 22;  // offset to uncompressed size
uncompSize = bytes.readUnsignedInt();  // store uncompressed size
output += "\tUncompressed size is: " + uncompSize + '<br />';
```

続いて、圧縮サイズで指定されている長さだけファイルストリームからファイルの残りを読み取って bytes に格納し、最初の 30 バイトのファイルヘッダーを上書きします。圧縮サイズは、ファイルが圧縮されていない場合でも正確です。その場合の圧縮サイズはファイルの非圧縮サイズと等しくなるからです。

```
// read compressed file to offset 0 of bytes; for uncompressed files
// the compressed and uncompressed size is the same
if (compSize == 0) continue;
zStream.readBytes(bytes, 0, compSize);
```

次に、圧縮ファイルの圧縮を解除し、outfile() 関数を呼び出して出力ファイルストリームに書き込みます。outfile() には、ファイル名と、ファイルデータを含むバイト配列を渡します。

```
if (compMethod == 8) // if file is compressed, uncompress
{
    bytes.uncompress(air.CompressionAlgorithm.DEFLATE);
}
outFile(fileName, bytes);   // call outFile() to write out the file
```

右中括弧は、while ループの終わり、および init() メソッドとアプリケーションコード（outFile() メソッド以外）の終わりを示します。実行は while ループの先頭に戻り、.zip ファイルの次のバイトの処理を続けます。つまり、別のファイルを展開するか、または最後のファイルの処理が済んだ場合は .zip ファイルの処理を終了します。すべてのファイルの処理が終了したら、output 変数の内容を div エレメントの taFiles に書き込み、ファイルの情報を画面に表示します。

```
    } // end of while loop

    document.getElementById("taFiles").innerHTML = output;
} // end of init() method
```

outfile() 関数は、デスクトップ上の出力ファイルを WRITE モードで開き、filename パラメーターで指定された名前を設定します。次に、ファイルデータを data パラメーターから出力ファイルストリーム（outStream）に書き込み、ファイルを閉じます。

```
function outFile(fileName, data)
{
    var outFile = air.File.desktopDirectory; // dest folder is desktop
    outFile = outFile.resolvePath(fileName);  // name of file to write
    var outStream = new air.FileStream();
    // open output file stream in WRITE mode
    outStream.open(outFile, air.FileMode.WRITE);
    // write out the file
    outStream.writeBytes(data, 0, data.length);
    // close it
    outStream.close();
}
```

第 17 章：AIR での PDF コンテンツの追加

Adobe AIR 1.0 およびそれ以降

Adobe® AIR® で実行しているアプリケーションでは、SWF コンテンツや HTML コンテンツだけでなく、PDF コンテンツもレンダリングできます。AIR アプリケーションは、HTMLLoader クラス、WebKit エンジンおよび Adobe® Reader® のブラウザープラグインを使用して PDF コンテンツをレンダリングします。AIR アプリケーションでは、PDF コンテンツをアプリケーションのウィンドウ全体の大きさまで拡大したり、インターフェイスの一部にしたりすることができます。
Adobe Reader ブラウザープラグインは、AIR アプリケーションでの PDF ファイルの表示を制御します。Reader のツールバーインターフェイス（位置、アンカー設定、可視性など）を変更すると、その変更は、後で AIR アプリケーションとブラウザーで PDF ファイルを表示する場合にも反映されます。

重要：AIR で PDF コンテンツをレンダリングするには、Adobe Reader または Adobe® Acrobat® のバージョン 8.1 以降がユーザーのコンピューターにインストールされている必要があります。

PDF 機能の検出

Adobe AIR 1.0 およびそれ以降

Adobe Reader または Adobe Acrobat 8.1 以降がインストールされていない場合、AIR アプリケーションに PDF コンテンツは表示されません。ユーザーが PDF コンテンツをレンダリングできるかどうかを調べるには、最初に HTMLLoader.pdfCapability プロパティを確認します。このプロパティは、HTMLPDFCapability クラスの次のいずれかの定数に設定されます。

定数	説明
HTMLPDFCapability.STATUS_OK	適切なバージョン（8.1 以降）の Adobe Reader が検出されたため、PDF コンテンツを HTMLLoader オブジェクトに読み込むことができます。
HTMLPDFCapability.ERROR_INSTALLED_READER_NOT_FOUND	Adobe Reader が検出されませんでした。HTMLLoader オブジェクトは PDF コンテンツを表示できません。
HTMLPDFCapability.ERROR_INSTALLED_READER_TOO_OLD	Adobe Reader が検出されましたが、バージョンが古すぎます。HTMLLoader オブジェクトは PDF コンテンツを表示できません。
HTMLPDFCapability.ERROR_PREFERRED_READER_TOO_OLD	適切なバージョン（8.1 以降）の Adobe Reader が検出されましたが、8.1 より前のバージョンの Adobe Reader で PDF コンテンツを処理するように設定されています。HTMLLoader オブジェクトは PDF コンテンツを表示できません。

Windows では、ユーザーのシステムで Adobe Acrobat または Adobe Reader のバージョン 7.x 以降が動作している場合、PDF の読み込みをサポートするより新しいバージョンがインストールされていたとしても、そのバージョンが使用されます。この場合、pdfCapability プロパティの値が HTMLPDFCapability.STATUS_OK であれば、AIR アプリケーションで PDF コンテンツを読み込もうとしたときに、古いバージョンの Acrobat または Reader に警告が表示されます（AIR アプリケーションで例外がスローされることはありません）。エンドユーザーがこのような状況になる可能性がある場合は、アプリケーションを実行しているときに Acrobat を開かないよう指示するメッセージをエンドユーザーに表示することを検討してください。PDF コンテンツが許容可能な時間内に読み込まれない場合は、このようなメッセージが表示されるようにすることをお勧めします。

Linux では、AIR はユーザーが書き出した PATH 内（acroread コマンドが含まれる場合）および /opt/Adobe/Reader ディレクトリ内の Adobe Reader を検索します。

次のコードでは、ユーザーが AIR アプリケーションで PDF を表示できるかどうかを調べます。PDF を表示できない場合は HTMLPDFCapability エラーオブジェクトに対応するエラーコードがトレースされます。

```
if(air.HTMLLoader.pdfCapability == air.HTMLPDFCapability.STATUS_OK)
{
    air.trace("PDF content can be displayed");
}
else
{
    air.trace("PDF cannot be displayed. Error code:", HTMLLoader.pdfCapability);
}
```

PDF コンテンツの読み込み

Adobe AIR 1.0 およびそれ以降

AIR アプリケーションに PDF を追加するには、HTMLLoader インスタンスを作成し、そのサイズを設定して、PDF のパスを読み込みます。

AIR アプリケーションでは、ブラウザーと同様の方法で PDF を追加できます。例えば、ウィンドウの最上位の HTML、オブジェクトタグ、フレームまたは iframe に PDF を読み込むことができます。

次の例では、外部サイトから PDF を読み込んでいます。iframe の src プロパティを値を使用可能な外部 PDF へのパスに置き換えます。

```
<html>
    <body>
        <h1>PDF test</h1>
        <iframe id="pdfFrame"
            width="100%"
            height="100%"
            src="http://www.example.com/test.pdf"/>
    </body>
</html>
```

また、file URL スキームや AIR 固有の URL スキーム（app や app-storage など）からコンテンツを読み込むこともできます。例えば、次のコードでは、アプリケーションディレクトリの PDFs サブディレクトリにある test.pdf を読み込んでいます。

app:/js_api_reference.pdf

AIR URL スキームについて詳しくは、311 ページの「URI スキーム」を参照してください。

PDF コンテンツのスクリプト作成

Adobe AIR 1.0 およびそれ以降

ブラウザー内の Web ページと同様に、JavaScript を使用して PDF コンテンツを制御することができます。

Acrobat の JavaScript 拡張機能には、次のような機能があります。

- ページのナビゲーションおよび表示倍率の制御

- ドキュメント内のフォームの処理
- マルチメディアイベントの制御

Adobe Acrobat の JavaScript 拡張機能について詳しくは、Adobe Acrobat Developer Connection（http://www.adobe.com/devnet/acrobat/javascript.html）を参照してください。

HTML と PDF 間の通信の基礎

Adobe AIR 1.0 およびそれ以降

HTML ページ内の JavaScript では、PDF コンテンツを表す DOM オブジェクトの postMessage() メソッドを呼び出すことで、PDF コンテンツ内の JavaScript にメッセージを送信することができます。例えば、次のような埋め込み PDF コンテンツがあるとします。

```
<object id="PDFObj" data="test.pdf" type="application/pdf" width="100%" height="100%"/>
```

コンテナとなっている HTML コンテンツ内にある次の JavaScript は、PDF ファイル内の JavaScript にメッセージを送信します。

```
pdfObject = document.getElementById("PDFObj");
pdfObject.postMessage(["testMsg", "hello"]);
```

PDF ファイルには、このメッセージを受信するための JavaScript を含めることができます。PDF ファイルには、ドキュメントレベル、フォルダーレベル、ページレベル、フィールドレベル、バッチレベルなどのコンテキストで JavaScript を追加できます。ここでは、PDF ドキュメントが開いたときに評価されるスクリプトを定義するドキュメントレベルコンテキストについてのみ説明します。

PDF ファイルでは、hostContainer オブジェクトに messageHandler プロパティを追加できます。messageHandler プロパティは、メッセージに応答するハンドラー関数を定義するオブジェクトです。例えば、次のコードでは、PDF ファイルがホストコンテナ（PDF ファイルが埋め込まれた HTML コンテンツ）から受信したメッセージを処理する関数を定義しています。

```
this.hostContainer.messageHandler = {onMessage: myOnMessage};

function myOnMessage(aMessage)
{
    if(aMessage[0] == "testMsg")
    {
        app.alert("Test message: " + aMessage[1]);
    }
    else
    {
        app.alert("Error");
    }
}
```

HTML ページ内の JavaScript では、ページに含まれている PDF オブジェクトの postMessage() メソッドを呼び出すことができます。このメソッドを呼び出すと、PDF ファイル内のドキュメントレベルの JavaScript にメッセージ（"Hello from HTML"）が送信されます。

```
<html>
    <head>
    <title>PDF Test</title>
    <script>
        function init()
        {
            pdfObject = document.getElementById("PDFObj");
            try {
                 pdfObject.postMessage(["alert", "Hello from HTML"]);
            }
            catch (e)
            {
                alert( "Error: \n name = " + e.name + "\n message = " + e.message );
            }
        }
    </script>
    </head>
    <body onload='init()'>
        <object
            id="PDFObj"
            data="test.pdf"
            type="application/pdf"
            width="100%" height="100%"/>
    </body>
</html>
```

より高度な例と、Acrobat 8 を使用して PDF ファイルに JavaScript を追加する方法については、「Adobe AIR アプリケーションの PDF コンテンツのクロススクリプト」を参照してください。

AIR 内の PDF コンテンツに関する既知の制限

Adobe AIR 1.0 およびそれ以降

Adobe AIR 内の PDF コンテンツには、次のような制限があります。

- PDF コンテンツは、透明な（transparent プロパティが true に設定されている）ウィンドウ（NativeWindow オブジェクト）には表示されません。
- PDF ファイルの表示順序は、AIR アプリケーションの他の表示オブジェクトの表示順序とは動作が異なります。PDF コンテンツは、HTML の表示順序に従って適切に配置されますが、AIR アプリケーションの表示順序では常にコンテンツの最上部に配置されます。
- PDF ドキュメントを含む HTMLLoader オブジェクトの特定のビジュアルプロパティが変更されると、PDF ドキュメントは表示されなくなります。このようなプロパティには、filters、alpha、rotation および scaling があります。これらのプロパティを変更すると、プロパティがリセットされるまで PDF コンテンツは表示されません。HTMLLoader オブジェクトを含む表示オブジェクトコンテナで同じプロパティを変更した場合も PDF コンテンツは表示されません。
- PDF コンテンツが表示されるのは、PDF コンテンツを含む NativeWindow オブジェクトの Stage オブジェクトの scaleMode プロパティ（window.nativeWindow.stage プロパティ）が air.StageScaleMode.NO_SCALE に設定されている場合のみです。他の値に設定されている場合、PDF コンテンツは表示されません。
- PDF ファイル内のコンテンツへのリンクをクリックすると、PDF コンテンツのスクロール位置が更新されます。PDF ファイルの外部にあるコンテンツへのリンクをクリックすると、（リンクのターゲットが新規ウィンドウである場合でも）PDF ファイルを含む HTMLLoader オブジェクトがリダイレクトされます。
- PDF コメントワークフローは、AIR では機能しません。

第 18 章：サウンドの操作

Adobe® AIR® クラスには、サウンドコンテンツを読み込んで再生する機能など、ブラウザーで実行する HTML コンテンツには使用できない機能が多く含まれます。

関連項目

flash.media.Sound

flash.media.Microphone

flash.events.SampleDataEvent

サウンドの操作の基礎

サウンドを制御するには、先にサウンドを Adobe AIR アプリケーションに読み込む必要があります。オーディオデータを AIR に読み込むには 5 つの方法があります。

- MP3 ファイルなどの外部サウンドファイルを、アプリケーションに読み込むことができます。
- サウンド情報を SWF ファイルに埋め込み、それを読み込んで（<script src="[swfFile].swf" type="application/x-shockwave-flash"/> を使用して）、再生できます。
- ユーザーのコンピューターに接続されたマイクロフォンを使用して、オーディオ入力を取得できます。
- サーバーからストリーミングされるサウンドデータにアクセスできます。
- サウンドデータを動的に生成できます。

サウンドデータを外部サウンドファイルから読み込むときは、サウンドデータの読み込みがすべて終わらないうちに、サウンドファイルの先頭部分の再生を開始できます。

デジタルオーディオのエンコードには様々なサウンドファイル形式が使用されますが、AIR でサポートされるのは MP3 形式で保存されたサウンドファイルです。WAV や AIFF などの他の形式のサウンドファイルを、直接読み込んだり再生したりすることはできません。

AIR でサウンドを操作するときは、runtime.flash.media パッケージの複数のクラスを使用する場合があります。Sound クラスは、サウンドファイルをロードすることにより、またはサウンドデータをサンプリングして再生を開始するイベントに関数を割り当てることにより、オーディオ情報にアクセスするために使用するクラスです。サウンドの再生を開始すると、SoundChannel オブジェクトにアクセスできるようになります。読み込んだオーディオファイルは、アプリケーションが同時に再生する複数のサウンドのうちの 1 つにすぎない場合があります。再生している個々のサウンドはそれぞれが専用の SoundChannel オブジェクトを使用します。スピーカを通して実際に再生されるサウンドでは、すべての SoundChannel オブジェクトが混合された状態で一緒に出力されます。サウンドのプロパティを制御したり、再生を停止したりするには、この SoundChannel インスタンスを使用します。最後に、混合されたオーディオを制御する必要がある場合は、SoundMixer クラスを使用して混合出力を制御できます。

また、他の複数のランタイムクラスを使用して、AIR でサウンドを操作するときにさらに細かいタスクを実行することもできます。サウンド関連のすべてのクラスについて詳しくは、269 ページの「サウンドアーキテクチャについて」を参照してください。

Adobe AIR デベロッパーセンターには、サンプルアプリケーションが用意されています（HTML ベースのアプリケーションにおけるサウンドの再生（http://www.adobe.com/go/learn_air_qs_sound_html_jp））。

サウンドアーキテクチャについて

アプリケーションでは、次の 5 つのメインソースからサウンドデータをロードすることができます。

- 実行時に読み込む外部サウンドファイル
- SWF ファイルに埋め込まれているサウンドリソース
- ユーザーのシステムに接続されたマイクロフォンから入力されるサウンドデータ
- Flash Media Server などのリモートメディアサーバーからストリーミングされるサウンドデータ
- sampleData イベントハンドラーを使用して動的に生成されるサウンドデータ

サウンドデータは、完全に読み込んでから再生することも、ストリーミングすることも、つまりまだ読み込みの途中で再生することもできます。

Adobe AIR は、MP3 形式で保存されたサウンドファイルをサポートします。WAV や AIFF などの他の形式のサウンドファイルを、直接読み込んだり再生したりすることはできません。ただし、NetStream クラスを使用すると、AAC オーディオファイルを読み込んで再生することもできます。

AIR サウンドアーキテクチャには、次のクラスが含まれます。

クラス	説明
Sound	Sound クラスは、サウンドの読み込みを処理し、サウンドの基本的なプロパティを管理し、サウンドの再生を開始します。
SoundChannel	アプリケーションが Sound オブジェクトを再生するときは、再生を制御するために新しい SoundChannel オブジェクトが作成されます。SoundChannel オブジェクトは、サウンドの左右の再生チャンネルの音量を制御します。再生するサウンドはそれぞれ、専用の SoundChannel オブジェクトを使用します。
SoundLoaderContext	SoundLoaderContext クラスは、サウンドを読み込むときに使用するバッファリングの秒数、およびファイルを読み込むときにランタイムがサーバーでクロスドメインポリシーファイルを検索するかどうかを指定します。SoundLoaderContext オブジェクトは、Sound.load() メソッドに対するパラメーターとして使用されます。
SoundMixer	SoundMixer クラスは、アプリケーション内のすべてのサウンドに関係する再生とセキュリティのプロパティを制御します。実際には、複数のサウンドチャンネルが 1 つの共通 SoundMixer オブジェクトを介して混合されます。SoundMixer オブジェクトのプロパティ値は、現在再生中のすべての SoundChannel オブジェクトに適用されます。
SoundTransform	SoundTransform クラスには、サウンドの音量とバランスを制御する値が含まれます。SoundTransform オブジェクトは、数あるオブジェクトの中で、個々の SoundChannel オブジェクト、グローバル SoundMixer オブジェクト、または Microphone オブジェクトに適用できます。
ID3Info	ID3Info オブジェクトには、MP3 サウンドファイルに格納されることの多い ID3 メタデータ情報を表すプロパティが含まれます。
Microphone	Microphone クラスは、ユーザーのコンピューターに接続されたマイクロフォンまたはその他のサウンド入力デバイスを表します。マイクロフォンからのオーディオ入力は、ローカルスピーカに渡したり、リモートサーバーに送信したりできます。Microphone オブジェクトは、ゲイン、サンプリングレート、および固有のサウンドストリームの他の特性を制御します。

読み込まれて再生される各サウンドには、Sound クラスと SoundChannel クラスについて、専用のインスタンスが必要です。再生の間に、SoundMixer クラスは複数の SoundChannel インスタンスからの出力を混合します。

Sound、SoundChannel、および SoundMixer クラスは、マイクロフォンから取得したサウンドデータ、または Flash Media Server などのストリーミングメディアサーバーからのサウンドデータには使用されません。

外部のサウンドファイルの読み込み

Sound クラスの各インスタンスは、特定のサウンドリソースを読み込んで再生をトリガーするために存在します。アプリケーションは、1 つの Sound オブジェクトを再利用して複数のサウンドを読み込むことはできません。新しいサウンドリソースを読み込むには、アプリケーションで別の Sound オブジェクトを作成する必要があります。

サウンドオブジェクトの作成

ボタンに関連付けるクリック音のような小さいサウンドファイルを読み込む場合、アプリケーションでは、次の例のように Sound オブジェクトを作成して、サウンドファイルを自動的に読み込ませることができます。

```
var req = new air.URLRequest("click.mp3");
var s = new air.Sound(req);
```

Sound() コンストラクターは、最初のパラメーターとして URLRequest オブジェクトを受け取ります。URLRequest パラメーターに値が渡ると、新しい Sound オブジェクトは指定されたサウンドリソースの読み込みを自動的に開始します。

最も単純なケースを除いて、アプリケーションではサウンドの読み込みの進行状況に注意し、読み込み中のエラーを監視する必要があります。例えば、クリック音がかなり大きい場合、そのサウンドをトリガーするボタンをユーザーがクリックするまでに、読み込みが完了してない可能性があります。読み込まれていないサウンドを再生しようとすると、ランタイムエラーが発生する場合があります。サウンドが完全に読み込まれるのを待ってから、サウンドの再生を開始するアクションをユーザーが実行できるようにする方が安全です。

サウンドのイベントについて

Sound オブジェクトは、サウンド読み込みプロセスの間に、複数の種類のイベントを送出します。アプリケーションは、これらのイベントをリッスンして読み込みの進行状況を追跡し、サウンドが完全に読み込まれたのを確認してから再生できます。次の表は、Sound オブジェクトから送出できるイベントをまとめたものです。

イベント	説明
open (air.Event.OPEN)	サウンド読み込み操作の開始直前に送出されます。
progress (air.ProgressEvent.PROGRESS)	データをファイルまたはストリームから受け取るとき、サウンド読み込み処理の間に定期的に送出されます。
id3 (air.Event.ID3)	ID3 データが MP3 サウンドで使用できるようになると送出されます。
complete (air.Event.COMPLETE)	サウンドリソースの全データが読み込まれると送出されます。
ioError (air.IOErrorEvent.IO_ERROR)	サウンドファイルを読み込むことができない場合、または全サウンドデータを受け取る前に読み込み処理が中断された場合に送出されます。

次のコードは、読み込みが終了した後でサウンドを再生する方法を示しています。

```
var s = new air.Sound();
s.addEventListener(air.Event.COMPLETE, onSoundLoaded);
var req = new air.URLRequest("bigSound.mp3");
s.load(req);

function onSoundLoaded(event)
{
    var localSound = event.target;
    localSound.play();
}
```

まず、URLRequest パラメーターの初期値を指定せずに新しい Sound オブジェクトを作成します。次に、Sound オブジェクトからの complete イベントをリッスンし、すべてのサウンドデータが読み込まれると onSoundLoaded() メソッドを実行します。次に、サウンドファイルの新しい URLRequest 値を指定して Sound.load() メソッドを呼び出します。

サウンドの読み込みが完了すると、onSoundLoaded() メソッドを実行します。Event オブジェクトの target プロパティは、Sound オブジェクトへの参照です。Sound オブジェクトの play() メソッドを呼び出すと、サウンドの再生が開始されます。

サウンド読み込み処理の監視

サウンドファイルは、大きくて読み込みに長い時間がかかる場合があります。インターネットから読み込む場合は特にその可能性があります。アプリケーションでは読み込みが完了する前にサウンドを再生できます。読み込まれたサウンドの量、および既に再生されたサウンドの量を、ユーザーに示したい場合があります。

Sound クラスが送出する 2 つのイベント、progress と complete を使用すると、サウンドの読み込みの進行状況を比較的簡単に表示できます。次の例では、これらのイベントを使用して読み込み中のサウンドに関する進行状況の情報を表示する方法を示します。

```
var s = new Sound();
s.addEventListener(air.ProgressEvent.PROGRESS,
    onLoadProgress);
s.addEventListener(air.Event.COMPLETE,
    onLoadComplete);
s.addEventListener(air.IOErrorEvent.IO_ERROR,
    onIOError);

var req = new air.URLRequest("bigSound.mp3");
s.load(req);

function onLoadProgress(event)
{
    var loadedPct = Math.round(100 * (event.bytesLoaded / event.bytesTotal));
    air.trace("The sound is " + loadedPct + "% loaded.");
}

function onLoadComplete(event)
{
    var localSound = event.target;
    localSound.play();
}
function onIOError(event)
{
    air.trace("The sound could not be loaded: " + event.text);
}
```

このコードでは最初に Sound オブジェクトを作成し、progress イベントと complete イベントのリスナーをこのオブジェクトに追加します。Sound.load() メソッドを呼び出して、サウンドファイルから最初のデータを受け取ると、progress イベントが発生し、onSoundLoadProgress() メソッドがトリガーされます。

読み込みが完了したサウンドデータの割合は、ProgressEvent オブジェクトの bytesLoaded プロパティの値を bytesTotal プロパティの値で割って得られます。同じ bytesLoaded プロパティと bytesTotal プロパティは、Sound オブジェクトでも使用できます。

この例では、アプリケーションでサウンドファイル読み込み中のエラーを検出して対応する方法も示されています。例えば、指定ファイル名のサウンドファイルが見つからない場合、Sound オブジェクトは ioError イベントを送出します。前述のコードでは、エラーが発生すると onIOError() メソッドを実行して簡単なエラーメッセージを表示します。

埋め込みサウンドの操作

AIR では、JavaScript を使用して、SWF ファイルに埋め込まれているサウンドにアクセスできます。このような SWF ファイルは、次のいずれかの方法を使用してアプリケーションに読み込むことができます。

- HTML ページの <script> タグを使用する
- runtime.flash.display.Loader クラスを使用する

  アプリケーションの SWF ファイルにサウンドファイルを埋め込む厳密な方法は、SWF コンテンツの開発環境によって異なります。SWF ファイルにメディアを埋め込む方法については、使用している SWF コンテンツ開発環境のドキュメントを参照してください。

  埋め込まれているサウンドを使用するには、そのサウンドのクラス名を ActionScript で参照します。例えば、次のコードは自動的に生成される DrumSound クラスのインスタンスを作成することで開始します。

```
var drum = new DrumSound();
var channel = drum.play();
```

  DrumSound は flash.media.Sound クラスのサブクラスなので、Sound クラスのメソッドとプロパティを継承します。前の例に示すように、play() メソッドが含まれます。

ストリーミングサウンドファイルの操作

サウンドファイルまたはビデオファイルをまだ読み込んでいる間にそのデータを再生することを、ストリーミングと呼びます。リモートサーバーから読み込まれるサウンドは、すべてのサウンドデータが読み込まれるのを待たなくてもユーザーがサウンドを聴くことができるよう、ストリーミングされることがよくあります。

SoundMixer.bufferTime プロパティは、サウンドを再生する前にアプリケーションが収集するサウンドデータのミリ秒数を表します。つまり、bufferTime プロパティが 5000 に設定されている場合、アプリケーションはサウンドファイルから少なくとも 5000 ミリ秒のデータを読み込んでからサウンドの再生を開始します。SoundMixer.bufferTime のデフォルト値は 1000 です。

アプリケーションでは、サウンドを読み込むときに bufferTime の新しい値を明示的に指定することで、個別のサウンドについてグローバルな SoundMixer.bufferTime の値をオーバーライドできます。デフォルトのバッファー時間をオーバーライドするには、まず SoundLoaderContext クラスのインスタンスを作成してその bufferTime プロパティを設定し、次にパラメーターとして Sound.load() メソッドに渡します。次に、この例を示します。

```
var s = new air.Sound();
var url = "http://www.example.com/sounds/bigSound.mp3";
var req = new air.URLRequest(url);
var context = new air.SoundLoaderContext(8000, true);
s.load(req, context);
s.play();
```

再生が続いている間、AIR はサウンドバッファーを指定されたサイズ以上の大きさで保持しようとします。サウンドデータの読み込みが再生速度より速い場合、再生は中断せずに継続します。しかし、ネットワークの制限によりデータの読み込み速度が低下すると、再生ヘッドがサウンドバッファーの最後に達する場合があります。その場合、再生は一時停止しますが、サウンドデータが読み込まれると自動的に再開します。

AIR がデータの読み込みを待っているために再生が中断しているのかどうかを判定するには、Sound.isBuffering プロパティを使用します。

動的に生成されたオーディオの操作

既存のサウンドを読み込んだりストリーミングしたりする代わりに、オーディオデータを動的に生成することもできます。オーディオデータを生成するには、Sound オブジェクトの sampleData イベントのイベントリスナーを割り当てる必要があります（sampleData イベントは、SampleDataEvent クラスで定義されています）。この環境では、Sound オブジェクトはファイルからサウンドデータを読み込みません。代わりに、このイベントに割り当てた関数を使用して自身にストリーミングされるサウンドデータのソケットとして機能します。

Sound オブジェクトに sampleData イベントリスナーを追加すると、サウンドバッファーに追加するデータが定期的に要求されます。このバッファーには、Sound オブジェクトで再生するデータが格納されます。Sound オブジェクトの play() メソッドを呼び出すと、新しいサウンドデータの要求時に sampleData イベントが送出されます（Sound オブジェクトがファイルから mp3 データを読み込んでいない場合のみ）。

SampleDataEvent オブジェクトには、data プロパティがあります。イベントリスナーでは、この data オブジェクトに ByteArray オブジェクトを書き込みます。このオブジェクトに書き込んだバイト配列は、バッファーに格納されたサウンドデータに追加され、Sound オブジェクトで再生されます。バッファー内のバイト配列は、-1 ～ 1 の浮動小数点数値のストリームです。各浮動小数点数値は、サウンドサンプルの 1 つのチャンネル（左または右）の振幅を表します。サウンドは、1 秒間に 44,100 回サンプリングされます。各サンプルには、バイト配列の浮動小数点数値としてインターリーブされた、左または右のチャンネルが格納されます。

ハンドラー関数では、ByteArray.writeFloat() メソッドを使用して、data プロパティ（sampleData イベント）に書き込みます。例えば、次のコードでは正弦波が生成されます。

```
var mySound = new air.Sound();
mySound.addEventListener(air.SampleDataEvent.SAMPLE_DATA, sineWaveGenerator);
mySound.play();
function sineWaveGenerator(event)
{
    for (i = 0; i < 8192; i++)
    {
        var n = Math.sin((i + event.position) / Math.PI / 4);
        event.data.writeFloat(n);
        event.data.writeFloat(n);
    }
}
```

Sound.play() を呼び出すと、アプリケーションがイベントハンドラーの呼び出しを開始し、サウンドサンプルデータを要求します。データの供給を中止するか、SoundChannel.stop() を呼び出すまでは、サウンドが再生される間、アプリケーションはイベントを送信し続けます。

イベントの待ち時間はプラットフォームによって異なりますが、AIR の将来のバージョンでは変更される可能性があります。特定の待ち時間に依存するのではなく、計算して求めてください。待ち時間を計算するには、次の式を使用します。

```
(SampleDataEvent.position / 44.1) - SoundChannelObject.position
```

イベントリスナーの各呼び出しでは、SampleDataEvent オブジェクトの data プロパティに 2048 ～ 8192 サンプルを指定します。最適なパフォーマンスを得るためには、できるだけ多くのサンプル数を指定します（最大 8192）。用意するサンプルが少なければ少ないほど、再生中にクリック音およびポップ音が発生する可能性が高くなります。この動作は、プラット

フォームによって異なります。また、ブラウザーのサイズを変更したときなど、様々な状況で発生します。2048 サンプルだけを指定した場合、あるプラットフォームで機能するコードが、別のプラットフォームで実行した場合は機能しなくなる可能性があります。できるだけ待ち時間を小さくする必要がある場合は、データの量をユーザーが選択できるようにすることを検討してください。

sampleData イベントリスナーの 1 回の呼び出しごとのサンプル数を 2048 未満にすると、残りのサンプルの再生後にアプリケーションが停止します。その後、SoundComplete イベントが送出されます。

MP3 データのサウンドの変更

Sound オブジェクトからデータを抽出するには、Sound.extract() メソッドを使用します。そのデータを使用（および変更）して、再生用に別の Sound オブジェクトの動的ストリームに書き込むことができます。たとえば、次のコードでは、読み込んだ MP3 ファイルのバイトを、フィルター関数 upOctave() を使用して渡しています。

```
var mySound = new air.Sound();
var sourceSnd = new air.Sound();
var urlReq = new air.URLRequest("test.mp3");
sourceSnd.load(urlReq);
sourceSnd.addEventListener(air.Event.COMPLETE, loaded);
function loaded(event)
{
    mySound.addEventListener(SampleDataEvent.SAMPLE_DATA, processSound);
    mySound.play();
}
function processSound(event)
{
        var bytes = new air.ByteArray();
        sourceSnd.extract(bytes, 8192);
        event.data.writeBytes(upOctave(bytes));
}
function upOctave(bytes)
{
    var returnBytes = new air.ByteArray();
    bytes.position = 0;
    while(bytes.bytesAvailable > 0)
    {
        returnBytes.writeFloat(bytes.readFloat());
        returnBytes.writeFloat(bytes.readFloat());
        if (bytes.bytesAvailable > 0)
        {
            bytes.position += 8;
        }
    }
    return returnBytes;
}
```

生成されたサウンドに関する制限事項

Sound オブジェクトと共に sampleData イベントリスナーを使用する場合、Sound の他のメソッドのうち、使用できるのは Sound.extract() と Sound.play() だけです。その他のメソッドまたはプロパティを呼び出すと、例外が発生します。SoundChannel オブジェクトのすべてのメソッドおよびプロパティは、引き続き使用できます。

サウンドの再生

次に示すように、読み込んだサウンドは Sound オブジェクトの Sound.play() メソッドを呼び出すだけで再生できます。

```
var req = new air.URLRequest("smallSound.mp3");
var snd = new air.Sound(req);
snd.play();
```

サウンドの再生操作

サウンドを再生するときは、次の操作を実行できます。

- 特定の開始位置からサウンドを再生する
- サウンドを一時停止し、後で同じ位置から再生を再開する
- サウンドの再生が終了する正確な時間を知る
- サウンドの再生の進行状況を追跡する
- サウンドの再生中に音量またはバランスを変更する

再生中にこれらの操作を実行するには、SoundChannel、SoundMixer、SoundTransform の各クラスを使用します。

SoundChannel クラスは、単一のサウンドの再生を制御します。SoundChannel.position プロパティは再生ヘッドと考えることができ、サウンドデータ内で再生中の現在位置を示します。

アプリケーションが Sound.play() メソッドを呼び出すと、再生を制御するために SoundChannel クラスの新しいインスタンスが作成されます。

アプリケーションは Sound.play() メソッドの startTime パラメーターとして、ミリ秒の単位で表した特定の開始位置を渡すことで、その位置からサウンドを再生できます。また、Sound.play() メソッドの loops パラメーターで数値を渡すことで、間を置かずにサウンドを繰り返して再生する固定回数を指定できます。

Sound.play() メソッドを呼び出すときに startTime パラメーターと loops パラメーターの両方を指定すると、サウンドが毎回同じ開始位置から繰り返して再生されます。次のコードはこの例を示しています。

```
var req = new air.URLRequest("repeatingSound.mp3");
var snd = new air.Sound();
snd.play(1000, 3);
```

この例では、サウンドの先頭から 1 秒後の位置から連続して 3 回、サウンドを再生します。

サウンドの一時停止と再開

歌やポッドキャストのような長いサウンドをアプリケーションで再生する場合、通常は、ユーザーがサウンドの再生を一時停止および再開できるようにする必要があります。サウンドは文字通り一時的に休止させておくことはできず、停止させることしかできません。ただし、サウンドの再生はどこからでも開始できます。サウンドを停止した位置を記録しておき、後でその位置からサウンドの再生を開始できます。

例えば、次のようなサウンドファイルを読み込んで再生するコードがあるとします。

```
var req = new air.URLRequest("bigSound.mp3");
var snd = new air.Sound(req);
var channel = snd.play();
```

サウンドを再生している間、channel オブジェクトの position プロパティは、サウンドファイル内で現在再生されている位置を示します。アプリケーションでは次のようにして、サウンドの再生を停止する前に位置の値を保存できます。

```
var pausePosition = channel.position;
channel.stop();
```

サウンドの再生を再開するには、保存しておいた位置の値を渡します。サウンドは停止したときと同じ位置から再開されます。

```
channel = snd.play(pausePosition);
```

再生の監視

サウンドの再生が停止したことをアプリケーションに認識させたい場合があります。その後で別のサウンドの再生を開始したり、前回の再生で使用したリソースをクリーンアップしたりできます。SoundChannel クラスは、サウンドの再生が終了すると soundComplete イベントを送出します。次の例で示すように、アプリケーションはこのイベントをリッスンして、適切なアクションを実行できます。

```
var snd = new air.Sound("smallSound.mp3");
var channel = snd.play();
s.addEventListener(air.Event.SOUND_COMPLETE, onPlaybackComplete);

public function onPlaybackComplete(event)
{
    air.trace("The sound has finished playing.");
}
```

SoundChannel クラスは、再生中には progress イベントを送出しません。再生の進行状況を報告するために、アプリケーションで独自のタイミングメカニズムを設定してサウンド再生ヘッドの位置を追跡できます。

サウンドの再生済みの割合を計算するには、SoundChannel.position プロパティの値を、再生しているサウンドデータの長さで割ります。

```
var playbackPercent = 100 * (channel.position / snd.length);
```

ただし、このコードが正確な再生済みの割合を報告するのは、再生を開始する前にサウンドデータが完全に読み込まれていた場合のみです。Sound.length プロパティが示すのは、現時点で読み込まれているサウンドデータのサイズであり、サウンドファイル全体の最終的なサイズではありません。まだ読み込みが完了していないストリーミングサウンドの再生の進行を追跡するには、サウンドファイル全体の最終的なサイズを推定し、その値を使用して計算する必要があります。サウンドデータの最終的な長さは、次のように Sound オブジェクトの bytesLoaded プロパティと bytesTotal プロパティを使用して推定できます。

```
var estimatedLength = Math.ceil(snd.length / (snd.bytesLoaded / snd.bytesTotal));
var playbackPercent = 100 * (channel.position / estimatedLength);
```

次のコードは、大きいサウンドファイルを読み込み、setInterval() 関数をタイミングメカニズムとして使用して、再生の進行状況を表示します。このコードが定期的に報告する再生済みの割合は、現在位置の値をサウンドデータの全長で割って得られたものです。

```
var snd = new air.Sound();
var url = "http://www.example.com/sounds/test.mp3";
var req = new air.URLRequest(url);
snd.load(req);

var channel = snd.play();
var timer = setInterval(monitorProgress, 100);
snd.addEventListener(air.Event.SOUND_COMPLETE, onPlaybackComplete);

function monitorProgress(event)
{
    var estimatedLength = Math.ceil(snd.length / (snd.bytesLoaded / snd.bytesTotal));
    var playbackPercent = Math.round(100 * (channel.position / estimatedLength));
    air.trace("Sound playback is " + playbackPercent + "% complete.");
}

function onPlaybackComplete(event)
{
    air.trace("The sound has finished playing.");
    clearInterval(timer);
}
```

サウンドデータの読み込みを開始した後、このコードは snd.play() メソッドを呼び出して、結果の SoundChannel オブジェクトを channel 変数に格納します。その後で monitorProgress() メソッドを追加し、setInterval() 関数で繰り返し呼び出します。このコードでは、イベントリスナーを使用して、再生が完了すると発生する SoundChannel オブジェクトの soundComplete イベントをリッスンします。

monitorProgress() メソッドは、読み込み済みのデータの量に基づいてサウンドファイルの全体の長さを推定します。その後、現在の再生済みの割合を計算して表示します。

サウンド全体の再生が終了すると、onPlaybackComplete() 関数を実行します。この関数は、setInterval() 関数のコールバックメソッドを削除して、再生の終了後に進行状況の更新が表示されないようにします。

ストリーミングサウンドの停止

ストリーミング中のサウンド、つまり再生中にまだ読み込まれているサウンドの再生処理では、他の場合とは異なる動作が発生します。ストリーミングサウンドを再生している SoundChannel インスタンスで stop() メソッドを呼び出すと、サウンドの再生が停止した後、サウンドの先頭から再生が再開します。このようなことが発生するのは、サウンドの読み込み処理が進行中であるためです。ストリーミングサウンドの読み込みと再生の両方を停止するには、Sound.close() メソッドを呼び出します。

サウンドの音量とバランスの制御

個々の SoundChannel オブジェクトは、サウンドの左右のステレオチャンネルを両方とも制御します。MP3 サウンドがモノラルサウンドの場合、SoundChannel オブジェクトの左右のステレオチャンネルに含まれる波形は同じです。

再生中のサウンドの各ステレオチャンネルの振幅を検出するには、SoundChannel オブジェクトの leftPeak プロパティと rightPeak プロパティを使用します。これらのプロパティは、サウンド波形自体のピーク振幅を示します。実際の再生音量を表してはいません。実際の再生音量は、サウンド波形の振幅と、SoundChannel オブジェクトおよび SoundMixer クラスで設定されている音量値の相関結果です。

SoundChannel オブジェクトの pan プロパティを使用すると、再生中の左右のチャンネルに異なる音量レベルを指定できます。pan プロパティに指定できる値は、-1 ～ 1 の範囲です。-1 という値は、左のチャンネルを最大音量で再生し、右のチャンネルを消音することを意味します。1 という値は、右のチャンネルを最大音量で再生し、左のチャンネルを消音することを意味します。-1 と 1 の間の値は、左と右のチャンネルの値を指定した値に比例して設定します。値 0 は、両方のチャンネルを均等に中程度の音量レベルで再生することを意味します。

次のコード例では、SoundTransform オブジェクトを作成し、音量の値を 0.6 に、バランスの値を -1（左上チャンネルは最大音量、右チャンネルは消音）に設定しています。その後、SoundTransform オブジェクトを play() メソッドにパラメーターとして渡します。play() メソッドは、渡された SoundTransform オブジェクトを、再生を制御するために作成される新しい SoundChannel オブジェクトに適用します。

```
var req = new air.URLRequest("bigSound.mp3");
var snd = new air.Sound(req);
var trans = new air.SoundTransform(0.6, -1);
var channel = snd.play(0, 1, trans);
```

サウンドの再生中に、音量とバランスを変更できます。そのためには、SoundTransform オブジェクトの pan または volume プロパティを設定し、そのオブジェクトを SoundChannel オブジェクトの soundTransform プロパティとして適用します。

また、SoundMixer クラスの soundTransform プロパティを使用すると、すべてのサウンドに対してグローバルな音量とバランスの値を一度に設定することもできます。次に、この例を示します。

```
SoundMixer.soundTransform = new air.SoundTransform(1, -1);
```

また、SoundTransform オブジェクトを使用すると、Microphone オブジェクトの音量とバランスも設定できます（282 ページの「サウンド入力のキャプチャ」を参照）。

次の例では、サウンドの再生中に、サウンドのバランスを左チャンネルから右チャンネルに変更してから元に戻しています。

```
var snd = new air.Sound();
var req = new air.URLRequest("bigSound.mp3");
snd.load(req);

var panCounter = 0;

var trans = new air.SoundTransform(1, 0);
var channel = snd.play(0, 1, trans);
channel.addEventListener(air.Event.SOUND_COMPLETE,
                         onPlaybackComplete);

var timer = setInterval(panner, 100);

function panner()
{
    trans.pan = Math.sin(panCounter);
    channel.soundTransform = trans; // or SoundMixer.soundTransform = trans;
    panCounter += 0.05;
}

function onPlaybackComplete(event)
{
    clearInterval(timer);
}
```

このコードでは、最初にサウンドファイルを読み込んだ後、SoundTransform オブジェクトを作成し、volume を 1（最大音量）に、バランスを 0（左右均等）に設定します。次に、snd.play() メソッドを呼び出し、SoundTransform オブジェクトをパラメーターとして渡します。

サウンドの再生中には、panner() メソッドを繰り返して実行します。panner() メソッドは、Math.sin() 関数を使用して、-1 ～ 1 の範囲の値を生成します。この範囲は、SoundTransform.pan プロパティに設定できる値に対応しています。SoundTransform オブジェクトの pan プロパティを新しい値に設定した後、チャンネルの soundTransform プロパティを、変更後の SoundTransform オブジェクトを使用するように設定します。

このコード例を実行する場合は、ファイル名 bigSound.mp3 をローカルの MP3 ファイルの名前に変更します。その後で例を実行します。まず、左チャンネルの音量が大きくなって右チャンネルの音量が小さくなり、次にはその逆に音量が変化します。

この例では、SoundMixer クラスの soundTransform プロパティを設定することで同じ効果を実現することもできます。ただし、その場合は、この SoundChannel オブジェクトが再生している単一のサウンドだけではなく、その時点で再生しているすべてのサウンドのバランスが影響を受けます。

サウンドメタデータの操作

MP3 形式を使用するサウンドファイルには、サウンドに関する追加データが ID3 タグの形式で含まれることがあります。

すべての MP3 ファイルに ID3 メタデータが含まれるわけではありません。Sound オブジェクトが MP3 サウンドファイルをロードすると、サウンドファイルに ID3 メタデータが含まれる場合でも、Event.ID3 イベントが送出されます。ランタイムエラーを防ぐため、アプリケーションでは、Event.ID3 イベントを受信するのを待ってから、読み込まれたサウンドの Sound.id3 プロパティにアクセスする必要があります。

次のコードでは、サウンドファイルの ID3 メタデータが読み込まれたことを認識する方法を示します。

```
var s = new air.Sound();
s.addEventListener(air.Event.ID3, onID3InfoReceived);
var urlReq = new air.URLRequest("mySound.mp3");
s.load(urlReq);

function onID3InfoReceived(event)
{
    var id3 = event.target.id3;

    air.trace("Received ID3 Info:");
    for (propName in id3)
    {
        air.trace(propName + " = " + id3[propName]);
    }
}
```

このコードでは、最初に Sound オブジェクトを作成し、id3 イベントをリッスンするように指示しています。サウンドファイルの ID3 メタデータが読み込まれると、onID3InfoReceived() メソッドが呼び出されます。onID3InfoReceived() メソッドに渡される Event オブジェクトの対象は、元の Sound オブジェクトです。その後、このメソッドは Sound オブジェクトの id3 プロパティを取得し、指定されているプロパティを順番に参照して値を追跡します。

生のサウンドデータへのアクセス

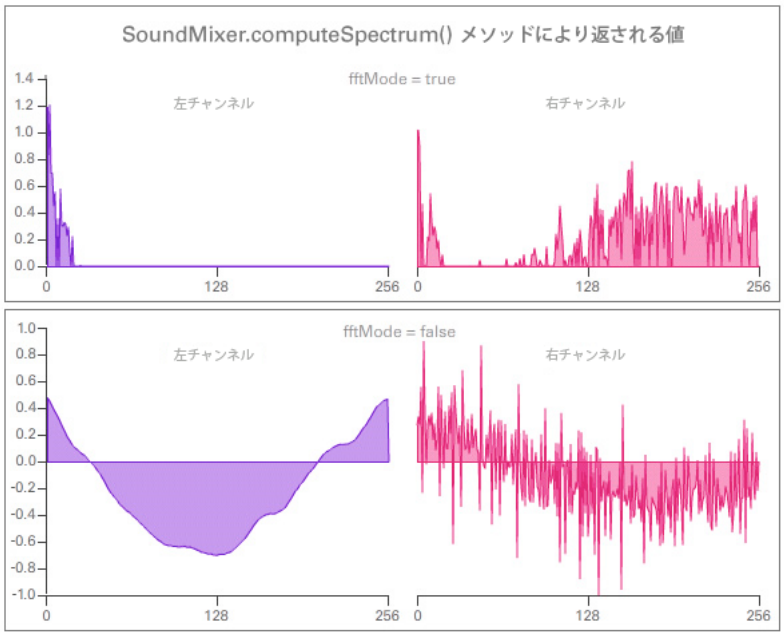
SoundMixer.computeSpectrum() メソッドを使用すると、現在再生されている波形の生のサウンドデータを、アプリケーションで読み取ることができます。複数の SoundChannel オブジェクトが再生中である場合、SoundMixer.computeSpectrum() メソッドでは、すべての SoundChannel オブジェクトが混合されたサウンドデータが示されます。

サウンドデータが返される方法

サウンドデータは、ByteArray オブジェクトとして返されます。このオブジェクトは 512 個の 4 バイトデータのセットを含み、各データは -1 ～ 1 の範囲の浮動小数点数値を表します。これらの値は、再生されているサウンド波形内のポイントにおける振幅を表します。値はそれぞれが 256 個の値から成る 2 つのグループとして提供され、第 1 のグループは左ステレオチャンネルを表し、第 2 のグループは右ステレオチャンネルを表します。

SoundMixer.computeSpectrum() メソッドは、FFTMode パラメーターが true に設定されている場合は、波形データではなく周波数スペクトルデータを返します。周波数スペクトルは、最低周波数から最高周波数へと、サウンドの周波数順に並べられた振幅を示します。波形データを周波数スペクトルデータに変換するには、高速フーリエ変換（FFT）が使用されます。結果の周波数スペクトルの値は、0 から約 1.414（2 の平方根）の範囲になります。

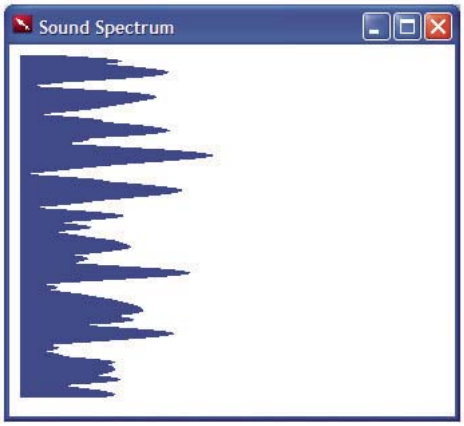
次の図は、computeSpectrum() メソッドから返されるデータを、FFTMode パラメーターが true の場合と false の場合で比較したものです。この図に使用したサウンドは、左チャンネルが大きいベース音で、右チャンネルがドラムを叩く音です。



computeSpectrum() メソッドは、より低いビットレートで再サンプリングされたデータを返すこともできます。一般にはこのようにすると、音の細部は損なわれますが、より滑らかな波形データまたは周波数データが得られます。computeSpectrum() メソッドのデータをサンプリングするレートは、stretchFactor パラメーターで制御します。stretchFactor パラメーターを 0（デフォルト値）に設定すると、サウンドデータは 44.1 kHz のレートでサンプリングされます。stretchFactor パラメーターの値が 1 大きくなるごとに、レートは半分になります。したがって、値に 1 を指定するとレートは 22.05 kHz、値に 2 を指定するとレートは 11.025 kHz、となります。stretchFactor の値を高くしても、computeSpectrum() メソッドはステレオチャンネルごとに 256 個の浮動小数点数値を返します。

簡易なサウンドビジュアライザの作成

次の例では、SoundMixer.computeSpectrum() メソッドを使用して、周期的にアニメーション動作するサウンド波形のグラフを表示します。



```
<html>
    <title>Sound Spectrum</title>
    <script src="AIRAliases.js" />
    <script>
        const PLOT_WIDTH = 600;
        const CHANNEL_LENGTH = 256;

        var snd = new air.Sound();
        var req = new air.URLRequest("test.mp3");
        var bytes = new air.ByteArray();
        var divStyles = new Array;

        /**
            * Initializes the application. It draws 256 DIV elements to the document body,
            * and sets up a divStyles array that contains references to the style objects of
            * each DIV element. It then calls the playSound() function.
            */
        function init()
        {
            var div;
            for (i = 0; i < CHANNEL_LENGTH; i++)
            {
                div = document.createElement("div");
                div.style.height = "1px";
                div.style.width = "0px";
                div.style.backgroundColor = "blue";
                document.body.appendChild(div);
                divStyles[i] = div.style;
            }
            playSound();
        }
        /**
            * Plays a sound, and calls setInterval() to call the setMeter() function
            * periodically, to display the sound spectrum data.
            */
        function playSound()
        {
            if (snd.url != null)
            {
                snd.close();
            }
            snd.load(req);
            var channel = snd.play();
            timer = setInterval(setMeter, 100);
            snd.addEventListener(air.Event.SOUND_COMPLETE, onPlaybackComplete);
        }

        /**
            * Computes the width of each of the 256 colored DIV tags in the document,
            * based on data returned by the call to SoundMixer.computeSpectrum(). The
            * first 256 floating point numbers in the byte array represent the data from
            * the left channel, and then next 256 floating point numbers represent the
            * data from the right channel.
            */
        function setMeter()
        {
            air.SoundMixer.computeSpectrum(bytes, false, 0);
            var n;
            for (var i = 0; i < CHANNEL_LENGTH; i++)
```

```
            {
                bytes.position = i * 4;
                n = Math.abs(bytes.readFloat());
                bytes.position = 256*4 + i * 4;
                n += Math.abs(bytes.readFloat());
                divStyles[i].width = n * PLOT_WIDTH;
            }
        }
        /**
         * When the sound is done playing, remove the intermediate process
         * started by setInterval().
         */
        function onPlaybackComplete(event)
        {
            clearInterval(interval);
        }
    </script>
    <body onload="init()">
    </body>
</html>
```

この例では、まずサウンドファイルを読み込んで再生し、次に setInterval() 関数を使用して SoundMixer.computeSpectrum() メソッドを監視し、サウンド波形データを bytes ByteArray オブジェクトに格納します。

サウンド波形は、棒グラフを表す div エレメントの幅を設定して描画します。

サウンド入力のキャプチャ

Microphone クラスを使用すると、ユーザーのシステムのマイクロフォンまたは他のサウンド入力デバイスにアプリケーションを接続できます。アプリケーションでは、入力したオーディオをシステムのスピーカにブロードキャストしたり、Flash Media Server などのリモートサーバーにオーディオデータを送信したりできます。マイクロフォンから入力される生のオーディオデータにはアクセスできません。システムのスピーカへのオーディオの送信、またはリモートサーバーへの圧縮オーディオデータの送信のみが可能です。リモートサーバーに送信するデータには、Speex または Nellymoser コーデックを使用できます（Speex コーデックは AIR 1.5 で使用できます）。

マイクロフォンへのアクセス

Microphone クラスにはコンストラクターメソッドはありません。代わりに、次の例で示すように、静的な Microphone.getMicrophone() メソッドを使用して新しい Microphone インスタンスを取得します。

```
var mic = air.Microphone.getMicrophone();
```

パラメーターを指定せずに Microphone.getMicrophone() メソッドを呼び出すと、ユーザーのシステムで検出された最初のサウンド入力デバイスが返されます。

システムには複数のサウンド入力デバイスを接続できます。アプリケーションでは、Microphone.names プロパティを使用して、使用可能なすべてのサウンド入力デバイスの名前の配列を取得できます。次に、Microphone.getMicrophone() メソッドを、配列にあるデバイスの名前のインデックス値と一致する index パラメーターを指定して呼び出します。

システムには、マイクロフォンまたは他のサウンド入力デバイスが接続されていない場合があります。Microphone.names プロパティまたは Microphone.getMicrophone() メソッドを使用すると、サウンド入力デバイスがインストールされているかどうかを確認できます。サウンド入力デバイスがインストールされていない場合は names 配列の長さがゼロになり、getMicrophone() メソッドは null 値を返します。

マイクロフォンのオーディオからローカルスピーカへのルーティング

Microphone.setLoopback() メソッドを呼び出してパラメーター値に true を指定することで、マイクロフォンからのオーディオ入力をローカルシステムのスピーカにルーティングできます。

ローカルマイクロフォンからのサウンドをローカルスピーカにルーティングすると、オーディオフィードバックループが形成されるリスクが伴います。それによってキーンという大きい音が発生し、サウンドハードウェアが破損する可能性があります。Microphone.setUseEchoSuppression() メソッドを呼び出してパラメーターに true を指定すると、オーディオフィードバックが発生するリスクは低下しますが、完全に除去することはできません。ユーザーがサウンドの再生にヘッドフォンなどスピーカ以外の装置を使用することが確実でない限り、Microphone.setLoopback(true) を呼び出す前に必ず Microphone.setUseEchoSuppression(true) を呼び出すことをお勧めします。

次のコードでは、ローカルマイクロフォンからローカルシステムのスピーカにオーディオをルーティングする方法を示します。

```
var mic = air.Microphone.getMicrophone();
mic.setUseEchoSuppression(true);
mic.setLoopBack(true);
```

マイクロフォンのオーディオの変更

アプリケーションでは、マイクロフォンから入力されたオーディオデータを 2 つの方法で変更できます。最初の方法では、入力サウンドのゲインを変更でき、実際には、指定した値で入力値が乗算されます。これにより、サウンドの音量を大きくしたり小さくしたりできます。Microphone.gain プロパティは、0 ～ 100 の範囲の数値を受け付けます。50 という値は 1 を掛けるのと同じで、通常の音量を指定します。値 0 は 0 を掛けるのと同じで、結果として入力オーディオは消音されます。50 より大きい値を指定すると、通常の音量より大きくなります。

アプリケーションでは、入力オーディオのサンプリングレートを変更することもできます。サンプリングレートを高くするほどサウンドの品質は向上しますが、データストリームの密度も高くなり、送信と格納に必要なリソースの量が増加します。Microphone.rate プロパティは、キロヘルツ（kHz）単位で測定されたオーディオサンプリングレートを表します。デフォルトのサンプリングレートは 8 kHz です。マイクロフォンがこれより高いレートをサポートしている場合は、Microphone.rate プロパティに 8 kHz より高い値を設定できます。例えば、Microphone.rate プロパティを 11 に設定するとサンプリングレートは 11 kHz になり、22 に設定するとサンプリングレートは 22 kHz になります。使用可能なサンプリングレートは、選択したコーデックによって異なります。Nellymoser コーデックを使用する場合は、サンプリングレートとして 5、8、11、16、22 および 44 kHz を指定できます。Speex コーデック（AIR 1.5 で利用可能）を使用する場合は、16 kHz のみ使用できます。

マイクロフォンの動作の検出

帯域幅と処理リソースを節約するため、ランタイムはマイクロフォンがサウンドを送信していないときにそれを検出しようとします。マイクロフォンの動作レベルが、ある一定の期間、無音レベルのしきい値を下回ったままであるとき、ランタイムはオーディオ入力の送信を停止して activity イベントを送出します。Speex コーデック（AIR 1.5 で利用可能）を使用する場合は、オーディオデータがアプリケーションから継続的に送信されるように、無音レベルを 0 に設定します。Speex の音声動作検出によって、帯域幅が自動的に軽減されます。

Microphone クラスの 3 つのプロパティを使用して、動作の検出を監視および制御します。

- 読み取り専用の activityLevel プロパティは、マイクロフォンが検出しているサウンドの大きさを、0 ～ 100 のスケールで示します。
- silenceLevel プロパティは、マイクロフォンをアクティブにして activity イベントを送出するために必要なサウンドの大きさを指定します。silenceLevel プロパティも 0 ～ 100 のスケールを使用し、デフォルト値は 10 です。

- silenceTimeout プロパティは、activity イベントを送出するしきい値となる、動作レベルが無音レベルを下回る時間の長さをミリ秒単位の数値で示します。silenceTimeout のデフォルト値は 2000 です。

Microphone.silenceLevel プロパティと Microphone.silenceTimeout プロパティはどちらも読み取り専用ですが、Microphone.setSilenceLevel() メソッドを使用して値を変更できます。

場合によっては、新しい動作が検出されたときにマイクロフォンをアクティブにする処理過程で短い遅延が発生する可能性があります。常にマイクロフォンをアクティブにしておくことで、アクティブ化に伴うこのような遅延を除去できます。そのためには、Microphone.setSilenceLevel() メソッドを、silenceLevel パラメーターに 0 を指定して呼び出します。これにより、サウンドが検出されなくても、マイクロフォンはアクティブな状態でオーディオデータの収集を継続します。逆に、silenceLevel パラメーターを 100 に設定すると、マイクロフォンがアクティブにならなくなります。

次の例では、マイクロフォンに関する情報を表示し、Microphone オブジェクトが送出する activity イベントと status イベントを報告します。

```
var deviceArray = air.Microphone.names;
air.trace("Available sound input devices:");
for (i = 0; i < deviceArray.length; i++)
{
    air.trace("   " + deviceArray[i]);
}

var mic = air.Microphone.getMicrophone();
mic.gain = 60;
mic.rate = 11;
mic.setUseEchoSuppression(true);
mic.setLoopBack(true);
mic.setSilenceLevel(5, 1000);

mic.addEventListener(air.ActivityEvent.ACTIVITY, this.onMicActivity);

var micDetails = "Sound input device name: " + mic.name + '\n';
micDetails += "Gain: " + mic.gain + '\n';
micDetails += "Rate: " + mic.rate + " kHz" + '\n';
micDetails += "Muted: " + mic.muted + '\n';
micDetails += "Silence level: " + mic.silenceLevel + '\n';
micDetails += "Silence timeout: " + mic.silenceTimeout + '\n';
micDetails += "Echo suppression: " + mic.useEchoSuppression + '\n';
air.trace(micDetails);

function onMicActivity(event)
{
    air.trace("activating=" + event.activating + ", activityLevel=" +
        mic.activityLevel);
}
```

前の例を実行し、システムのマイクロフォンに向かって話すか音を立ててみて、コンソールに表示されるトレースステートメントを確認してください。

メディアサーバーとのオーディオの送受信

Flash Media Server などのストリーミングメディアサーバーを使用すると、他のオーディオ機能を利用できます。

特に、Microphone オブジェクトを runtime.flash.net.NetStream オブジェクトに接続して、ユーザーのマイクロフォンからサーバーにデータを直接送信できる機能が挙げられます。また、サーバーから AIR アプリケーションにオーディオデータをストリーミングすることもできます。

AIR 1.5 では、Speex コーデックがサポートされるようになりました。メディアサーバーに送信される圧縮オーディオに使用するコーデックを設定するには、Microphone オブジェクトの codec プロパティを使用します。このプロパティには、SoundCodec クラスで列挙されている 2 つの値を設定できます。codec プロパティを SoundCodec.SPEEX に設定すると、オーディオの圧縮時に Speex コーデックスが選択されます。このプロパティを SoundCodec.NELLYMOSER（デフォルト）に設定すると、オーディオの圧縮に Nellymoser コーデックが選択されます。

詳しくは、http://www.adobe.com/jp/support/documentation からオンラインで提供されている Flash Media Server のドキュメントを参照してください。

第 19 章：クライアントのシステム環境

Flash Player 9 以降、Adobe AIR 1.0 以降

ここでは、ユーザーのシステムを操作する方法について説明します。ここでは、サポートされている機能を判別する方法と、ユーザーが入力方式エディター（IME）をインストールしてある場合に、IME を使用して複数言語のアプリケーションを構築する方法を示します。また、アプリケーションドメインの一般的な使用方法も示します。

関連項目

flash.system.System

flash.system.Capabilities

クライアントのシステム環境の基礎

Flash Player 9 以降、Adobe AIR 1.0 以降

高度な アプリケーションを構築していると、ユーザーのオペレーティングシステムの詳細およびアクセス機能を知る必要がある場合があります。flash.system パッケージには、次のようなシステムレベルの機能にアクセスできるクラスのコレクションが含まれています。

- 実行しているアプリケーションとセキュリティドメインコードの判別
- 画面サイズ（解像度）などのユーザーの Flash ランタイム（Flash® Player、Adobe® AIR™ など）インスタンスの機能、および MP3 などの特定の機能が使用可能かどうかの判別
- IME を使用した複数言語サイトの構築
- Flash ランタイムのコンテナ（HTML ページの場合もコンテナアプリケーションの場合もあります）または AIR のコンテナの操作
- ユーザーのクリップボードへの情報の保存

また、flash.system パッケージには IMEConversionMode および SecurityPanel クラスも含まれています。これらでは、それぞれ IME クラスと Security クラスで使用する静的定数が定義されています。

重要な概念と用語

次の参照リストに、重要な用語を示します。

オペレーティングシステム Microsoft Windows、Mac OS X、Linux® など、コンピューターで実行されるメインプログラム。他のすべてのアプリケーションはこのプログラムの内部で実行されます。

クリップボード [くりっぷぼーど] コピーまたはカットしたテキストやアイテムを保持するためのオペレーティングシステムのコンテナ。コンテナ内のアイテムはアプリケーションにペーストできます。

アプリケーションドメイン 異なる SWF ファイルに使用されているクラスを分離するメカニズム。これにより、複数の SWF ファイルで同じ名前の別のクラスが使用されていても、そのクラスが他のクラスを上書きすることはありません。

IME（入力方式エディター） 標準キーボードを使用して複雑な文字や記号を入力するために使用するプログラム（またはオペレーティングシステムツール）。

クライアントシステム プログラミング用語では、クライアントは個々のコンピューターで実行されるアプリケーションの一部（またはアプリケーション全体）で、単一のユーザーによって使用されます。クライアントシステムはユーザーのコンピューター上の基になるオペレーティングシステムです。

System クラスの使用

Flash Player 9 以降、Adobe AIR 1.0 以降

System クラスには、ユーザーのオペレーティングシステムを操作したり、ランタイムの現在のメモリ使用状況を取得できるメソッドとプロパティが格納されています。System クラスのメソッドとプロパティを使用すると、imeComposition イベントの待ち受け、ユーザーの現行コードページを使用した外部テキストファイルのロードまたは Unicode としてのロードを行うためのランタイムに対する指示、ユーザーのクリップボードの内容の設定も行うことができます。

実行時のユーザーのシステムに関するデータの取得

Flash Player 9 以降、Adobe AIR 1.0 以降

System.totalMemory プロパティを調べると、ランタイムが現在使用しているメモリの量をバイト単位で判別できます。このプロパティを使用してメモリ使用量を監視すれば、空きメモリの状況の変化に応じてアプリケーションの動作を最適に調整できます。例えば、特定のビジュアル効果を使用するとメモリ使用量が大幅に増加する場合、状況によってその効果を変化または完全に無効にすることが考えられます。

System.ime プロパティは、現在インストールされている IME（入力メソッドエディター）への参照です。このプロパティを使用すると、addEventListener() メソッドで imeComposition イベント（flash.events.IMEEvent.IME_COMPOSITION）を待ち受けることができます。

System クラスの 3 番目のプロパティは useCodePage です。useCodePage を true に設定した場合、ランタイムは、オペレーティングシステムの通常のコードページを使用して外部テキストファイルを読み込みます。このプロパティを false に設定すると、ランタイムは外部ファイルを Unicode として解釈します。

System.useCodepage を true に設定した場合、外部テキストファイル内で使用されている文字がオペレーティングシステムの通常のコードページに含まれていないと、そのテキストは表示されないことに注意してください。例えば、中国語を含んだ外部テキストファイルをロードする場合、English Windows コードページを使用しているシステム上では文字が表示されません。これは、English Windows コードページには中国語の文字が含まれないためです。

アプリケーション内で使用される外部テキストファイルをすべてのプラットフォームのユーザーが表示できるようにするには、すべての外部テキストファイルを Unicode で保存し、System.useCodePage をデフォルトの false の設定のままにします。このように、ランタイムは、テキストを Unicode として解釈します。

Capabilities クラスの使用

Flash Player 9 以降、Adobe AIR 1.0 以降

Capabilities クラスを使用すると、開発者はアプリケーションの実行環境に関する情報を取得できます。Capabilities クラスの様々なプロパティにより、ユーザーのシステムにおける画面解像度、アクセシビリティソフトウェアに対するサポートの有無、ユーザーのオペレーティングシステムの言語、および、現在インストールされている Flash ランタイムのバージョンを知ることができます。

Capabilities クラスのプロパティを確認すれば、実際のユーザー環境に応じてアプリケーションの動作を最適に調整できます。例えば、Capabilities.screenResolutionX および Capabilities.screenResolutionY プロパティを調べることにより、ユーザーのシステムで使用されているディスプレイ解像度を判別し、どのサイズのビデオが最適かを判断できます。また、Capabilities.hasMP3 プロパティを調べれば、外部 MP3 ファイルをロードする前にユーザーのシステムにおける MP3 再生のサポート状況を知ることができます。

次のコードでは、クライアント環境にインストールされている Flash ランタイムのバージョン情報を正規表現で解析しています。

```
var versionString = air.Capabilities.version;
var pattern = /^(\w*) (\d*),(\d*),(\d*),(\d*)$/;
var result = pattern.exec(versionString);
if (result != null)
{
    air.trace("input: " + result.input);
    air.trace("platform: " + result[1]);
    air.trace("majorVersion: " + result[2]);
    air.trace("minorVersion: " + result[3]);
    air.trace("buildNumber: " + result[4]);
    air.trace("internalBuildNumber: " + result[5]);
}
else
{
    air.trace("Unable to match RegExp.");
}
```

第 20 章：AIR アプリケーションの呼び出しと終了

Adobe AIR 1.0 およびそれ以降

この節では、インストールした Adobe® AIR® アプリケーションを呼び出す方法、および実行中のアプリケーションの終了に使用するオプションと考慮事項について説明します。

注意：NativeApplication、InvokeEvent および BrowserInvokeEvent オブジェクトは、AIR アプリケーションサンドボックスで実行されている SWF コンテンツでのみ使用できます。Flash Player ランタイムで実行されている SWF コンテンツ、ブラウザーまたはスタンドアローンプレーヤー（プロジェクター）内で実行されている SWF コンテンツおよびアプリケーションサンドボックスの外部の AIR アプリケーションで実行されている SWF コンテンツは、これらのクラスにアクセスできません。

AIR アプリケーションの呼び出しと終了に関する簡単な説明およびコード例については、Adobe Developer Connection で次のクイックスタートの記事を参照してください。

- 起動オプション
- 起動オプション

関連項目

air.NativeApplication

flash.events.InvokeEvent

flash.events.BrowserInvokeEvent

アプリケーションの呼び出し

Adobe AIR 1.0 およびそれ以降

AIR アプリケーションは、ユーザー（またはオペレーティングシステム）が次の動作を行った場合に呼び出されます。

- アプリケーションをデスクトップシェルから起動する。
- アプリケーションをコマンドラインシェルのコマンドとして使用する。
- アプリケーションがデフォルトで関連付けられている種類のファイルを開く。
- （Mac OS X）ドックタスクバーでアプリケーションアイコンをクリックする（アプリケーションが現在実行中であるかどうかは問いません）。
- インストーラーでアプリケーションの起動を選択する（新規インストール作業の最後、またはインストール済みアプリケーションの AIR ファイルをダブルクリックした後）。
- インストールされたバージョンがアプリケーションのアップデートを自動処理していることを表示したときに、AIR アプリケーションのアップデートを開始する（アプリケーション記述ファイルに <customUpdateUI>true</customUpdateUI> 宣言を含めている場合）。

- AIR アプリケーション向けの識別情報を指定した com.adobe.air.AIR launchApplication() メソッドを呼び出す Flash バッジまたはアプリケーションをホストしている Web ページを表示する（ブラウザーによる呼び出しを可能にするには、アプリケーション記述子に <allowBrowserInvocation>true</allowBrowserInvocation> 宣言も含まれている必要があります）。

AIR アプリケーションが呼び出されるたびに、AIR はシングルトンの NativeApplication オブジェクトを通じて、型 invoke の InvokeEvent オブジェクトを送出します。アプリケーション時間が自分自身を初期化し、イベントリスナーを登録できるようにするには、invoke イベントを破棄せず、キューに入れる必要があります。リスナーが登録されると、キューに入れられているすべてのイベントが直ちに送出されます。

注意：ブラウザー呼び出し機能を使用してアプリケーションが呼び出されると、NativeApplication オブジェクトは、そのアプリケーションが既に実行中でなければ、invoke イベントのみを送出します。

invoke イベントを受け取るには、NativeApplication オブジェクト（NativeApplication.nativeApplication）の addEventListener() メソッドを呼び出します。イベントリスナーは、invoke イベントについて登録されると、登録前に発生したすべての invoke イベントも受け取ります。キューに入れられた invoke イベントは、addEventListener() に対する呼び出しが返された後、短い間隔で一度に 1 つずつ送出されます。このプロセス中に新しい invoke イベントが発生した場合、このイベントを、キューに入れられた 1 つ以上のイベントが送出される前に送出することができます。このイベントキューイングにより、初期化コードが実行される前に発生した任意の invoke イベントを処理できます。イベントリスナーは、実行の比較的遅い時点（アプリケーション初期化の後）で追加された場合でも、アプリケーション起動後に発生したすべての invoke イベントを受け取ります。

1 つの AIR アプリケーションについて、起動されるインスタンスは 1 つだけです。既に実行されているアプリケーションが再び呼び出されると、AIR は実行中のインスタンスに対して新しい invoke イベントを送出します。invoke イベントに応答し、適切なアクションを実行する（例えば新しいドキュメントウィンドウを開く）ことは、AIR アプリケーションの役割です。

InvokeEvent オブジェクトには、アプリケーションと、そのアプリケーションが呼び出されたディレクトリに渡された任意の引数が格納されます。アプリケーションがファイルの種類による関連付けによって呼び出された場合には、コマンドライン引数にそのファイルへのフルパスが含まれます。同様に、アプリケーションがアプリケーションアップデートのために呼び出された場合には、アップデート AIR ファイルへのフルパスが設定されます。

1 回の操作で複数のファイルが開かれる場合、Mac OS X では InvokeEvent オブジェクトが 1 回送出されます。各ファイルは arguments 配列に格納されます。Windows および Linux では、各ファイルごとに InvokeEvent オブジェクトが送出されます。

アプリケーションが invoke イベントを処理できるようにするには、その NativeApplication オブジェクトにリスナーを次のように登録します。

```
air.NativeApplication.nativeApplication.addEventListener(air.InvokeEvent.INVOKE, onInvokeEvent);
```

さらに、イベントリスナーを次のように定義します。

```
var arguments;
var currentDir;
function onInvokeEvent(invocation) {
    arguments = invocation.arguments;
    currentDir = invocation.currentDirectory;
}
```

コマンドライン引数のキャプチャ

Adobe AIR 1.0 およびそれ以降

AIR アプリケーションの呼び出しに関連するコマンドライン引数は、NativeApplication オブジェクトによって送出される InvokeEvent オブジェクトで送出されます。InvokeEvent の arguments プロパティには、AIR アプリケーションの呼び出し時にオペレーティングシステムによって渡される引数の配列が格納されています。引数にファイルの相対パスが含まれている場合、パスを解決するには、通常は currentDirectory プロパティを使用します。

AIR プログラムに渡される引数は、次の表に示すように、二重引用符で囲まれている場合を除いて、空白で区切られたストリングとして扱われます。

引数	配列
tick tock	{tick,tock}
tick "tick tock"	{tick,tick tock}
"tick" “tock”	{tick,tock}
\"tick\" \"tock\"	{"tick","tock"}

InvokeEvent オブジェクトの currentDirectory プロパティには、そのアプリケーションが起動されたディレクトリを表す File オブジェクトが格納されます。

アプリケーションが、そのアプリケーションによって登録された種類のファイルが開かれたことによって呼び出された場合には、そのファイルへのネイティブパスがコマンドライン引数にストリングとして設定されます（ファイルを開いたり、ファイルに対して目的の操作を実行することは、アプリケーションの役割です）。同様に、アプリケーションがアップデートを自動で（標準の AIR アップデートユーザーインターフェイスを使用せずに）実行するようにプログラムされている場合には、一致するアプリケーション ID を持つアプリケーションを格納した AIR ファイルをユーザーがダブルクリックしたときに、AIR ファイルへのネイティブパスが設定されます。

ファイルにアクセスするには、currentDirectory File オブジェクトの resolve() メソッドを次のように使用します。

```
if((invokeEvent.currentDirectory != null)&&(invokeEvent.arguments.length > 0)){
    dir = invokeEvent.currentDirectory;
    fileToOpen = dir.resolvePath(invokeEvent.arguments[0]);
}
```

引数が本当にファイルへのパスであるかどうかについても検証する必要があります。

例：呼び出しイベントログ

Adobe AIR 1.0 およびそれ以降

次の例では、invoke イベントに対するリスナーを登録し、このイベントを処理する方法を示します。この例では、受け取ったすべての呼び出しイベントを記録し、現在のディレクトリとコマンドライン引数を表示します。

注意：この例では、AIRAliases.js ファイルを使用しています。このファイルは、SDK のフレームワークフォルダーにあります。

```
 <html>
<head>
<title>Invocation Event Log</title>
<script src="AIRAliases.js" />
<script type="text/javascript">
function appLoad() {
    air.trace("Invocation Event Log.");
    air.NativeApplication.nativeApplication.addEventListener(
        air.InvokeEvent.INVOKE, onInvoke);
}

function onInvoke(invokeEvent) {
    logEvent("Invoke event received.");
    if (invokeEvent.currentDirectory) {
        logEvent("Current directory=" + invokeEvent.currentDirectory.nativePath);
    } else {
        logEvent("--no directory information available--");
    }

    if (invokeEvent.arguments.length > 0) {
        logEvent("Arguments: " + invokeEvent.arguments.toString());
    } else {
        logEvent("--no arguments--");
    }
}

function logEvent(message) {
    var logger = document.getElementById('log');
    var line = document.createElement('p');
    line.innerHTML = message;
    logger.appendChild(line);
    air.trace(message);
}

window.unload = function() {
        air.NativeApplication.nativeApplication.removeEventListener(
            air.InvokeEvent.INVOKE, onInvoke);
}
</script>
</head>

<body onLoad="appLoad();">
    <div id="log"/>
</body>
</html>
```

ユーザーログイン時の AIR アプリケーションの呼び出し

Adobe AIR 1.0 およびそれ以降

AIR アプリケーションが、現在のユーザーがログインしたときに自動的に起動するように設定するには、NativeApplication startAtLogin プロパティを true に設定します。このように設定すると、アプリケーションはユーザーがログインすると自動的に起動します。ログイン時にアプリケーションが自動起動しないようにするには、設定を false に変更するか、ユーザーが設定をオペレーティングシステムから手動で変更するか、またはアプリケーションをアンインストールします。ログイン時の起動は、実行時設定です。この設定は、現在のユーザーにのみ適用されます。startAtLogin プロパティを true に設定するには、アプリケーションがインストールされている必要があります。アプリケーションがインストールされていない（例えば ADL で起動される）場合、プロパティを設定するとエラーがスローされます。

注意：アプリケーションは、コンピューターシステムの起動時には起動しません。アプリケーションは、ユーザーのログイン時に起動します。

アプリケーションが自動的に起動されたのか、ユーザーアクションの結果起動されたのかを判別するには、InvokeEvent オブジェクトの reason プロパティを確認できます。このプロパティが InvokeEventReason.LOGIN の場合、アプリケーションは自動的に起動したことになります。他の呼び出しパスの場合、reason プロパティは InvokeEventReason.STANDARD になります。reason プロパティにアクセスするには、アプリケーションのターゲットを AIR 1.5.1 にする必要があります（アプリケーション記述ファイルで正しい名前空間値を設定します）。

次の単純化されたアプリケーションでは、InvokeEvent reason プロパティを使用して、invoke イベントの発生時の動作方法を決定します。reason プロパティが「login」である場合、アプリケーションは背景に留まります。それ以外の場合、メインアプリケーションが表示されるようになります。このパターンを使用しているアプリケーションは、バックグラウンド処理やイベント監視を実行できるように通常ログイン時に起動し、ユーザーがトリガーした invoke イベントに応じてウィンドウを開きます。

```
<html>
<head>
<script src="AIRAliases.js"></script>
<script language="javascript">
try
{
    air.NativeApplication.nativeApplication.startAtLogin = true;
}
catch ( e )
{
    air.trace( "Cannot set startAtLogin: " + e.message );
}

air.NativeApplication.nativeApplication.addEventListener( air.InvokeEvent.INVOKE, onInvoke );

function onInvoke( event )
{
    if( event.reason == air.InvokeEventReason.LOGIN )
    {
        //do background processing...
        air.trace( "Running in background..." );
    }
    else
    {
        window.nativeWindow.activate();
    }
}
</script>
</head>
<body>
</body>
</html>
```

注意：動作の違いを確認するには、アプリケーションをパッケージ化してインストールします。startAtLogin プロパティは、インストール済みのアプリケーションに対してのみ設定できます。

ブラウザーからの AIR アプリケーションの呼び出し

Adobe AIR 1.0 およびそれ以降

ブラウザー呼び出し機能を使用すると、Web サイトでは、インストールされている AIR アプリケーションをブラウザーから起動されるように起動できます。ブラウザーによる呼び出しは、アプリケーション記述ファイルで allowBrowserInvocation が次のように true に設定されている場合にのみ許可されます。

```
<allowBrowserInvocation>true</allowBrowserInvocation>
```

アプリケーションがブラウザーによって呼び出されると、アプリケーションの NativeApplication オブジェクトは BrowserInvokeEvent オブジェクトを送出します。

BrowserInvokeEvent イベントを受け取るには、AIR アプリケーションの NativeApplication オブジェクト（NativeApplication.nativeApplication）の addEventListener() メソッドを呼び出します。イベントリスナーは、BrowserInvokeEvent イベントについて登録されると、登録前に発生したすべての BrowserInvokeEvent イベントも受け取ります。これらのイベントは、addEventListener() に対する呼び出しが返された後で送出されますが、登録後に受け取られる可能性のある他の BrowserInvokeEvent イベントよりも前とは限りません。これにより、初期化コードが実行される前（例えばアプリケーションがブラウザーから最初に呼び出されたとき）に発生した BrowserInvokeEvent イベントを処理できます。イベントリスナーは、実行の比較的遅い時点（アプリケーション初期化の後）で追加された場合でも、アプリケーション起動後に発生したすべての BrowserInvokeEvent イベントを受け取ります。

BrowserInvokeEvent オブジェクトには、次のプロパティが含まれます。

プロパティ	説明
arguments	アプリケーションに渡す引数（ストリング）の配列。
isHTTPS	ブラウザー内のコンテンツが https URL スキームを使用する（true）か、使用しない（false）かを指定します。
isUserEvent	ブラウザーによる呼び出しがユーザーイベント（例えばマウスクリック）で発生したかどうかを示します。AIR 1.0 では、このプロパティは常に true に設定されます。AIR では、ブラウザー呼び出し機能に対するユーザーイベントが必要です。
sandboxType	ブラウザー内のコンテンツに対して使用するサンドボックスのタイプ。有効値の定義は、Security.sandboxType プロパティで使用できる有効値の場合と同じで、次のいずれかです。 • Security.APPLICATION — コンテンツは、アプリケーションセキュリティサンドボックスで実行されます。 • Security.LOCAL_TRUSTED — コンテンツは、local-with-filesystem セキュリティサンドボックスで実行されます。 • Security.LOCAL_WITH_FILE — コンテンツは、local-with-filesystem セキュリティサンドボックスで実行されます。 • Security.LOCAL_WITH_NETWORK — コンテンツは、local-with-networking セキュリティサンドボックスで実行されます。 • Security.REMOTE — コンテンツは、リモート（ネットワーク）ドメインで実行されます。
securityDomain	ブラウザー内のコンテンツのセキュリティドメイン。例えば「www.adobe.com」や「www.example.org」など。このプロパティは、リモートのセキュリティサンドボックスで実行されるコンテンツ（ネットワークドメインからのコンテンツ）に対してのみ設定されます。ローカルまたはアプリケーションセキュリティサンドボックスで実行されるコンテンツに対しては設定されません。

ブラウザー呼び出し機能を使用する場合は、セキュリティ上の影響を考慮してください。Web サイトで AIR アプリケーションが起動されると、アプリケーションは BrowserInvokeEvent オブジェクトの arguments プロパティを介してデータを送信できます。このデータを、ファイルやコードを読み込む API のような注意を必要とする操作で使用する場合には、慎重に行う必要があります。リスクのレベルは、アプリケーションがそのデータを使用して実行する処理の種類によって異なります。アプリケーションが特定の Web サイトだけで呼び出されることを想定している場合、そのアプリケーションは

BrowserInvokeEvent オブジェクトの securityDomain プロパティを検査する必要があります。また、そのアプリケーションを呼び出す Web サイトに対して、HTTPs を使用するように要求することもできます。これは、BrowserInvokeEvent オブジェクトの isHTTPS プロパティを検査することで検証できます。

アプリケーションは、渡されたデータを検証する必要があります。例えば、あるアプリケーションが特定のドメインの URL を渡されることを予期している場合、そのアプリケーションは、その URL が本当にそのドメインを指し示しているかどうかを検証する必要があります。これにより、攻撃者がアプリケーションを不正に操作して重要なデータを送信させることを防ぐことができます。

ローカルのリソースを指し示す可能性のある BrowserInvokeEvent 引数を、アプリケーションで使用しないでください。例えば、アプリケーションでは、ブラウザーから渡されたパスに基づいて File オブジェクトを作成することは避けてください。ブラウザーからリモートのパスが渡されることを想定している場合、アプリケーションでは、リモートプロトコルの代わりに file:// プロトコルがパスに使用されていないことを確認する必要があります。

アプリケーションの終了

Adobe AIR 1.0 およびそれ以降

アプリケーションを終了する最も早い方法は、NativeApplication exit() メソッドを呼び出すことです。これは、アプリケーションで保存するデータやクリーンアップするリソースがない場合には、適切に機能します。exit() を呼び出すと、すべてのウィンドウが閉じ、アプリケーションが終了します。しかし、例えば重要なデータを保存するために、アプリケーションのウィンドウやその他のコンポーネントが終了プロセスを中断できるようにする場合は、exit() を呼び出す前に、適切な警告イベントを送出します。

アプリケーションを適切な手順に従ってシャットダウンするためのもう一つの方法は、シャットダウンプロセスを開始する方法とは関係なく、単一の実行経路を設定することです。ユーザー（またはオペレーティングシステム）は、次に示す方法でアプリケーションの終了をトリガーすることができます。

- NativeApplication.nativeApplication.autoExit が true の場合に、最後のアプリケーションウィンドウを閉じる。
- オペレーティングシステムから、アプリケーション終了コマンドを選択する（例えば、ユーザーがデフォルトのメニューからアプリケーションを終了するコマンドを選択した場合）。（これは Mac OS でのみ可能です。Windows および Linux では、システムクロムによるアプリケーション終了コマンドは提供されていません。）
- コンピューターをシャットダウンする。

終了コマンドがオペレーティングシステムを介して、これらのいずれかの方法で渡されると、NativeApplication は exiting イベントを送出します。リスナーが exiting イベントをキャンセルしなければ、開いているすべてのウィンドウが閉じられます。各ウィンドウは closing イベントを送出し、次に close イベントを送出します。いずれかのウィンドウで closing イベントがキャンセルされると、シャットダウンプロセスは停止します。

アプリケーションで、ウィンドウを閉じる順序が重要である場合には、NativeApplication からの exiting イベントをリッスンし、ユーザー自身がウィンドウを正しい順序で閉じます。これは、例えば、ツールパレットを含むドキュメントウィンドウがある場合などに、必要となる可能性があります。システムがパレットを閉じたが、ユーザーはデータを保存するために終了コマンドをキャンセルすることにした、というような場合には、これは非常に不便です。Windows では、exiting イベントを受け取る唯一の機会は、最後のウィンドウを閉じた後です（NativeApplication オブジェクトの autoExit プロパティが true に設定されている場合）。

すべてのプラットフォームで一貫性のある動作を実現するためには、終了シーケンスがオペレーティングシステムクロム、メニューコマンド、アプリケーションロジックのいずれから開始される場合でも、アプリケーションを終了するための、次に示す正しい方法を遵守してください。

1 常に exiting イベントを NativeApplication オブジェクトから、アプリケーションコードで exit() を呼び出す前に送出し、アプリケーションの他のコンポーネントがこのイベントをキャンセルしていないことを確認します。

```
function applicationExit(){
    var exitingEvent = new air.Event(air.Event.EXITING, false, true);
    air.NativeApplication.nativeApplication.dispatchEvent(exitingEvent);
    if (!exitingEvent.isDefaultPrevented()) {
        air.NativeApplication.nativeApplication.exit();
    }
}
```

2 NativeApplication.nativeApplication オブジェクトから送出されたアプリケーションの exiting イベントをリッスンし、ハンドラーですべてのウィンドウを閉じます（最初に closing イベントを送出します）。すべてのウィンドウが閉じたら、アプリケーションデータの保存や一時ファイルの削除などの必要なクリーンアップタスクを実行します。クリーンアップでは同期メソッドのみを使用することで、アプリケーションの終了前にそれらのメソッドが必ず終了するようにします。

ウィンドウを閉じる順序が重要でない場合は、NativeApplication.nativeApplication.openedWindows 配列をループし、各ウィンドウを順次閉じていきます。順序が重要である場合には、ウィンドウを正しい順序で閉じる手段を提供します。

```
function onExiting(exitingEvent) {
    var winClosingEvent;
    for (var i = 0; i < air.NativeApplication.nativeApplication.openedWindows.length; i++) {
        var win = air.NativeApplication.nativeApplication.openedWindows[i];
        winClosingEvent = new air.Event(air.Event.CLOSING,false,true);
        win.dispatchEvent(winClosingEvent);
        if (!winClosingEvent.isDefaultPrevented()) {
            win.close();
        } else {
            exitingEvent.preventDefault();
        }
    }

    if (!exitingEvent.isDefaultPrevented()) {
        //perform cleanup
    }
}
```

3 各ウィンドウは、それ自身の closing イベントをリッスンすることで、常にそれ自身のクリーンアップを処理する必要があります。

4 アプリケーションでは、使用する exiting リスナーを 1 つだけにします。これは、以前に呼び出されたハンドラーは、後続のハンドラーが exiting イベントをキャンセルするかどうかを知ることができないためです（また、実行の順序に依拠することも賢明ではありません）。

第 21 章：AIR ランタイムとオペレーティングシステムに関する情報の操作

Adobe AIR 1.0 およびそれ以降

この節では、オペレーティングシステムファイルの関連付けの管理、ユーザーアクティビティの検出、Adobe® AIR® ランタイムに関する情報の取得を AIR アプリケーションで行う方法について説明します。

関連項目

flash.desktop.NativeApplication

ファイルの関連付けの管理

Adobe AIR 1.0 およびそれ以降

アプリケーションとファイルの種類との関連付けは、アプリケーション記述子で宣言する必要があります。インストールプロセス時、AIR アプリケーションインストーラーでは、宣言済みの各ファイルの種類を開くデフォルトのアプリケーションとして AIR アプリケーションを関連付けます。ただし、別のアプリケーションが既にデフォルトのアプリケーションになっている場合を除きます。AIR アプリケーションのインストールプロセスでは、既存のファイルの種類の関連付けを上書きしません。別のアプリケーションから関連付けを引き継ぐには、実行時に NativeApplication.setAsDefaultApplication() メソッドを呼び出します。

アプリケーションの起動時に、期待されるファイルの関連付けが適切であるかどうかを確認することをお勧めします。この理由は、AIR アプリケーションインストーラーでは既存のファイルの関連付けを上書きしないため、また、ユーザーのシステムにおけるファイルの関連付けは常に変更される可能性があるためです。現在のファイルの関連付けが別のアプリケーションに設定されている場合は、既存の関連付けを変更する前にユーザーに確認することも推奨されます。

NativeApplication クラスの次のメソッドを使用すると、アプリケーションでファイルの関連付けを管理できます。各メソッドは、ファイルの種類の拡張子をパラメーターとして受け取ります。

メソッド	説明
isSetAsDefaultApplication()	指定されたファイルの種類に AIR アプリケーションが現在関連付けられている場合は、true を返します。
setAsDefaultApplication()	AIR アプリケーションと特定のファイルの種類を開くアクションとの間の関連付けを作成します。
removeAsDefaultApplication()	AIR アプリケーションと特定のファイルの種類との間の関連付けを削除します。
getDefaultApplication()	特定のファイルの種類に現在関連付けられているアプリケーションのパスを報告します。

AIR で管理できるのは、アプリケーション記述子であらかじめ宣言されているファイルの種類の関連付けだけです。宣言されていないファイルの種類の関連付けに関する情報は、そのファイルの種類とアプリケーションの間の関連付けをユーザーが手動で作成した場合も含めて取得できません。アプリケーション記述子で宣言されていないファイルの種類の拡張子を指定して、ファイルの関連付けの管理メソッドを呼び出すと、ランタイム例外がアプリケーションからスローされます。

ランタイムのバージョンとパッチレベルの取得

Adobe AIR 1.0 およびそれ以降

NativeApplication オブジェクトには、アプリケーションが実行されているランタイムのバージョン（「1.0.5」などのストリング）を示す runtimeVersion プロパティがあります。また、NativeApplication オブジェクトには、ランタイムのパッチレベル（2960 などの番号）を示す runtimePatchLevel プロパティもあります。次のコードでは、これらのプロパティを使用します。

```
air.trace(air.NativeApplication.nativeApplication.runtimeVersion);
air.trace(air.NativeApplication.nativeApplication.runtimePatchLevel);
```

AIR 機能の検出

Adobe AIR 1.0 およびそれ以降

Adobe AIR アプリケーションに付属するファイルでは、Security.sandboxType プロパティは、Security.APPLICATION 定数で定義されている値に設定されます。ファイルが Adobe AIR セキュリティサンドボックス内にあるかどうかに基づいて、コンテンツ（AIR 固有の API を含んでいるかどうかわからないコンテンツ）を読み込むことができます。これを次のコードに示します。

```
if (window.runtime)
{
    if (air.Security.sandboxType == air.Security.APPLICATION)
    {
        alert("In AIR application security sandbox.");
    }
    else
    {
        alert("Not in AIR application security sandbox.")
    }
}
else
{
    alert("Not in the Adobe AIR runtime.")
}
```

AIR アプリケーションのインストールに含まれないすべてのリソースは、元のドメインに基づいてセキュリティサンドボックスに配置されます。例えば、www.example.com から提供されたコンテンツは、そのドメインのセキュリティサンドボックスに配置されます。

window.runtime プロパティの値をチェックして、コンテンツがランタイムで実行されるかどうかを確認できます。

詳しくは、68 ページの「AIR のセキュリティ」を参照してください。

ユーザープレゼンスの追跡

Adobe AIR 1.0 およびそれ以降

NativeApplication オブジェクトは、ユーザーがコンピューターをアクティブに使用していることを検出するのに役立つ 2 つのイベントを送出します。NativeApplication.idleThreshold プロパティで指定されている期間内に、マウスまたはキーボードの操作が検出されないと、NativeApplication は userIdle イベントを送出します。キーボードまたはマウスの次回の入力

が発生すると、NativeApplication オブジェクトは userPresent イベントを送出します。idleThreshold の期間は秒単位で計測され、デフォルト値は 300（5 分）です。最後のユーザー入力からの経過秒数を NativeApplication.nativeApplication.lastUserInput プロパティから取得することもできます。

次のコード行では、アイドル時間のしきい値を 2 分に設定し、userIdle イベントと userPresent イベントをリッスンします。

```
air.NativeApplication.nativeApplication.idleThreshold = 120;
air.NativeApplication.nativeApplication.addEventListener(air.Event.USER_IDLE, function(event) {
    air.trace("Idle");
});
air.NativeApplication.nativeApplication.addEventListener(air.Event.USER_PRESENT, function(event) {
    air.trace("Present");
});
```

注意：2 つの userPresent イベントの間に、userIdle イベントが 1 つだけ送出されます。

第 22 章：ソケット

Flash Player 9 以降、Adobe AIR 1.0 以降

ソケットとは、2 つのコンピュータープロセス間で確立されるネットワーク接続のタイプです。通常、これらのプロセスは、同じインターネットプロトコル（IP）ネットワークに接続している 2 台の異なるコンピューター上で実行されています。ただし、「local host」という名前の特殊な IP アドレスを使用すれば、接続された 2 つのプロセスを同じコンピューター上で実行することができます。

Adobe Flash Player は、クライアントサイドの伝送制御プロトコル（TCP）ソケットをサポートしています。Flash Player アプリケーションは、ソケットサーバーとして機能している別のプロセスに接続できますが、他のプロセスからの着信接続要求を受け入れることはできません。つまり、Flash Player アプリケーションは、TCP サーバーへの接続は可能ですが、サーバーとして機能することはできません。

Flash Player API には XMLSocket クラスも含まれています。XMLSocket クラスでは、Flash Player 固有のプロトコルを使用します。これにより、当該プロトコルを理解するサーバーと XML メッセージをやりとりできます。XMLSocket クラスは ActionScript 1 で導入され、後方互換性の維持のために、引き続きサポートされています。一般に、Socket クラスは、特に Flash XMLSocket との通信のために作成されたサーバーに接続する場合を除いて、新しいアプリケーションに対して使用する必要があります。

Adobe AIR によって、ソケットベースのネットワークプログラミング用のクラスがいくつか追加されます。AIR アプリケーションは、ServerSocket クラスを使用すれば TCP ソケットサーバーとして機能できます。また、SSL または TLS セキュリティが要求されるソケットサーバーには SecureSocket クラスを使用して接続できます。AIR アプリケーションは、DatagramSocket クラスを使用して、ユニバーサルデータグラムプロトコル（UDP）メッセージを送受信することもできます。

TCP ソケット

Flash Player 9 以降、Adobe AIR 1.0 以降

伝送制御プロトコル（TCP）には、固定ネットワーク接続経由でメッセージを交換する方法が用意されています。TCP では、送信されたメッセージは、（大きなネットワークの問題が発生しない限り）正しい順序で到達することが保証されています。TCP 接続では、「クライアント」と「サーバー」が必要です。Flash Player はクライアントソケットを作成できます。Adobe AIR では、さらに、サーバーソケットも作成できます。

以下の ActionScript API が TCP 接続を提供します。

- Socket - クライアントアプリケーションがサーバーに接続できるようにします。Socket クラスは受信接続を監視できません。
- SecureSocket（AIR）- クライアントアプリケーションが信頼できるサーバーに接続して、暗号化された通信を確立できるようにします。
- ServerSocket（AIR）- アプリケーションが受信接続を監視して、サーバーとして機能できるようにします。
- XMLSocket - クライアントアプリケーションが XMLSocket サーバーに接続できるようにします。

バイナリクライアントソケット

Flash Player 9 以降、Adobe AIR 1.0 以降

バイナリソケット接続は、XML ソケットに似ていますが、クライアント / サーバー間で交換するデータが XML メッセージに限定されないという点が異なります。データはバイナリ情報として伝送されます。したがって、電子メールサーバー（POP3、SMTP、IMAP）やニュースサーバー（NNTP）など様々なサービスに接続できます。

Socket クラス

Flash Player 9 以降、Adobe AIR 1.0 以降

Socket クラスを使用すると、ソケット接続を確立して生のバイナリデータを読み書きできます。Socket クラスは、バイナリプロトコルを使用するサーバーとの通信に役立ちます。バイナリソケット接続を使用すれば、POP3、SMTP、IMAP、NNTP など様々なインターネットプロトコルによる通信のコードを記述できます。この操作によって、アプリケーションから電子メールサーバーやニュースサーバーに接続することもできます。

Flash Player からサーバーに直接接続するには、サーバーが使用するバイナリプロトコルに従う必要があります。バイト順序にビッグエンディアンを使用するサーバーと、リトルエンディアンを使用するサーバーがあります。「ネットワークバイト順序」がビッグエンディアンとされているため、インターネット上の大部分のサーバーはビッグエンディアンのバイト順序を使用しています。また、Intel® x86 アーキテクチャでリトルエンディアンが採用されているので、リトルエンディアンのバイト順序が使用されることもよくあります。データを送受信する対象のサーバーに適したエンディアンバイト順序を使用してください。IDataInput および IDataOutput インターフェイスで実行されるすべての操作と、これらのインターフェイスを実装するクラス（ByteArray、Socket、URLStream）では、デフォルトでビッグエンディアン形式（先頭が最上位バイト）を使用します。Java と公式なネットワークバイト順序を一致させるために、このデフォルトのバイト順序が選択されました。使用するエンディアンを変更する場合は、endian プロパティを Endian.BIG_ENDIAN または Endian.LITTLE_ENDIAN に設定します。

Socket クラスは、（flash.utils パッケージ内の）IDataInput および IDataOutput インターフェイスで定義されているすべてのメソッドを継承します。これらのメソッドは、Socket クラスからの読み取りおよび書き込みに使用する必要があります。

詳しくは、以下を参照してください。

- Socket
- IDataInput
- IDataOutput
- socketData event

セキュアなクライアントソケット（AIR）

Adobe AIR 2 以降

SecureSocket クラスを使用すると、セキュアソケットレイヤーバージョン 4（SSLv4）またはトランスポート層セキュリティバージョン 1（TLSv1）を使用しているソケットサーバーに接続できます。セキュアなソケットには 3 つのメリット（サーバー認証、データの整合性、メッセージの機密性）があります。ランタイムでは、サーバー証明書と、ユーザーの信頼ストアに格納されているルート証明機関証明書または中間証明機関証明書との関連付けを使用してサーバーを認証します。ランタイムでは、SSL および TLS プロトコル実装で使用されている暗号化アルゴリズムに基づいて、データの整合性およびメッセージの機密性を確保します。

SecureSocket オブジェクトを使用してサーバーに接続する場合、ランタイムでは証明書信頼ストアを使用してサーバー証明書を検証します。Windows および Mac では、オペレーティングシステムが信頼ストアを提供します。Linux では、ランタイムが独自の信頼ストアを提供します。

サーバー証明書が有効でないまたは信頼できないと判断されると、ランタイムは ioError イベントを送出します。SecureSocket オブジェクトの serverCertificateStatus プロパティを調べることにより、無効とされた理由を確認できます。有効または信頼された証明書を持たないサーバーとの通信は許可されません。

CertificateStatus クラスでは、想定される検証結果を表す文字列定数を定義します。

- Expired - 証明書の有効期限が切れています。
- Invalid - 証明書が無効と判断される理由は様々です。例えば、証明書が変更されているまたは破損している場合もあれば、証明書のタイプが間違っている場合もあります。
- Invalid chain - サーバーの証明書チェーン内に無効な証明書が 1 つ以上存在します。
- Principal mismatch - サーバーのホスト名と証明書の共通名が一致しません。つまり、サーバーでは間違った証明書が使用されています。
- Revoked - 発行元の証明機関によって証明書が失効されています。
- Trusted - 証明書は有効で信頼されています。SecureSocket オブジェクトでは、有効で信頼された証明書を使用しているサーバーにのみ接続できます。
- Unknown - SecureSocket オブジェクトではまだ証明書を検証していません。connect() を呼び出す前と connect イベントまたは ioError イベントのいずれかが送出される前は、serverCertificateStatus プロパティにはこの状態が設定されています。
- Untrusted signers - 証明書には、クライアントコンピューターの信頼ストアに格納されている信頼されたルート証明書への「チェーン」がありません。

SecureSocket オブジェクトと通信する場合は、セキュアなプロトコルおよび有効で信頼された証明書が使用されているサーバーが必要です。その他の点では、SecureSocket オブジェクトと Socket オブジェクトの使用方法は同じです。

SecureSocket オブジェクト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。SecureSocket クラスの isSupported プロパティを使用すると、ランタイムが現在のクライアントコンピューターで SecureSocket オブジェクトの使用をサポートしているかどうかを確認できます。

詳しくは、以下を参照してください。

- SecureSocket
- CertificateStatus
- IDataInput
- IDataOutput
- socketData event

XML ソケット

Flash Player 9 以降、Adobe AIR 1.0 以降

XML ソケットを使用すると、明示的に閉じるまで開いたままの、リモートサーバーへの接続を作成できます。サーバーとクライアント間で、XML などのストリングデータを交換できます。XML ソケットサーバーを使用するメリットは、クライアントが明示的にデータを要求する必要がないことです。サーバーは要求に対してデータを送信するために待機することなく、すべての接続済みのクライアントにデータを送信できます。

Flash Player およびアプリケーションサンドボックスの外部にある Adobe AIR コンテンツの XML ソケット接続では、ターゲットサーバー上にソケットポリシーファイルが存在することが必要です。詳しくは、Web サイトのコントロール（ポリシーファイル）およびソケットへの接続を参照してください。

XMLSocket クラスは、ファイアウォールをトンネリングによって自動的に通過することはできません。RTMP とは異なり、XMLSocket には HTTP トンネリング機能が備わっていないためです。HTTP トンネリングが必要な場合は、Flash Remoting または（RTMP をサポートする）Flash Media Server の使用をお勧めします。

アプリケーションセキュリティサンドボックスの外部にある、Flash Player または AIR アプリケーションのコンテンツから XMLSocket オブジェクトを介してサーバーに接続する場合、その接続方法と接続先に関して次のような制限が適用されます。

- アプリケーションセキュリティサンドボックスの外部にあるコンテンツの場合、XMLSocket.connect() メソッドでは、1024 以上の TCP ポート番号にしか接続できません。この制限により、XMLSocket オブジェクトと通信するサーバーデーモンにも、1024 以上のポート番号を割り当てる必要があります。1024 未満のポート番号は、FTP（21）、Telnet（23）、SMTP（25）、HTTP（80）、POP3（110）などのシステムサービスによって使用されることが多いため、XMLSocket オブジェクトはセキュリティ上の理由からこれらのポートにアクセスできません。こうしたリソースが不適切な方法でアクセスされたり悪用されたりする可能性を小さくするために、ポート番号が制限されています。
- アプリケーションセキュリティサンドボックスの外部にあるコンテンツの場合、XMLSocket.connect() メソッドでは、コンテンツが存在するドメイン内のコンピューターにしか接続できません（この制限は、URLLoader.load() のセキュリティ規則と同じです）。コンテンツが存在するドメイン以外で実行されるサーバーデーモンに接続するには、特定のドメインからのアクセスを許可するクロスドメインポリシーファイルを作成します。クロスドメインポリシーファイルについて詳しくは、68 ページの「AIR のセキュリティ」を参照してください。

注意：XMLSocket オブジェクトと通信できるようにサーバーを設定することは、場合によっては困難が伴います。リアルタイムのインタラクティブ機能を必要としないアプリケーションでは、XMLSocket クラスではなく URLLoader クラスを使用してください。

XMLSocket クラスの XMLSocket.connect() メソッドと XMLSocket.send() メソッドを使用すると、ソケット接続を介してサーバーとの間で XML を転送できます。XMLSocket.connect() メソッドは、Web サーバーのポートに対してソケット接続を確立します。XMLSocket.send() メソッドは、ソケット接続で指定されたサーバーに XML オブジェクトを渡します。

XMLSocket.connect() メソッドを呼び出すと、アプリケーションによってサーバーへの TCP/IP 接続が開かれ、次のいずれかのイベントが発生するまでその接続が開いたまま維持されます。

- XMLSocket クラスの XMLSocket.close() メソッドが呼び出される。
- XMLSocket オブジェクトへの参照が 1 つも存在しなくなる。
- 接続が切れる（モデムの切断など）。

XMLSocket クラスによるサーバーへの接続

Flash Player 9 以降、Adobe AIR 1.0 以降

ソケット接続を確立するには、ソケット接続の要求を受け付けて Flash Player または AIR アプリケーションに応答を送るサーバーサイドアプリケーションを作成する必要があります。このようなサーバーサイドアプリケーションは、AIR または Java、Python、Perl などの別のプログラミング言語で作成できます。XMLSocket クラスを使用するには、サーバーコンピューターは XMLSocket クラスで使用されるシンプルプロトコルに対応したデーモンを実行する必要があります。

- XML メッセージは、全二重 TCP/IP ストリームソケット接続を介して送られます。
- 個々の XML メッセージは完全な XML ドキュメントであり、ゼロ（0）バイトで終了します。
- 1 つの XMLSocket 接続を使用して送受信できる XML メッセージの数に制限はありません。

サーバーソケット

Adobe AIR 2 以降

ServerSocket クラスを使用すると、他のプロセスが伝送制御プロトコル（TCP）ソケットを使用してアプリケーションに接続できます。接続しているプロセスは、ローカルコンピューターまたはネットワークに接続された別のコンピューター上で動作できます。ServerSocket オブジェクトが接続要求を受信すると、connect イベントが送出されます。このイベントを伴って送出された ServerSocketConnectEvent オブジェクトには Socket オブジェクトが含まれています。それ以降は、この Socket オブジェクトを使用して他のプロセスと通信できます。

受信ソケット接続を監視するには：

1. ServerSocket オブジェクトを作成して、ローカルポートにバインドします。
2. connect イベントに対するイベントリスナーを追加します。
3. listen() メソッドを呼び出します。
4. connect イベントに応答して、各受信接続に Socket オブジェクトを提供します。

ServerSocket オブジェクトは close() メソッドが呼び出されるまで、新しい接続を継続的に監視します。

次のコード例では、ソケットサーバーアプリケーションの作成方法を示します。この例では、ポート 8087 で受信接続を監視します。接続を受信すると、メッセージ（「Connected」という文字列）がクライアントソケットに送信されます。これ以降、サーバーで受信したメッセージはすべてクライアントにエコーバックされます。

```
<html>
<head>
<script src="AIRAliases.js"></script>
<script language="javascript">
        var serverSocket;
        var clientSockets = new Array();
        function startServer()
        {
            try
            {
                // Create the server socket
                serverSocket = new air.ServerSocket();

                // Add the event listener
                serverSocket.addEventListener( air.Event.CONNECT, connectHandler );
                serverSocket.addEventListener( air.Event.CLOSE, onClose );

                // Bind to local port 8087
                serverSocket.bind( 8087, "127.0.0.1" );

                // Listen for connections
                serverSocket.listen();
                air.trace( "Listening on " + serverSocket.localPort );
            }
            catch( e )
            {
                air.trace( e );
            }
        }
        function connectHandler( event )
        {
            //The socket is provided by the event object
            var socket = event.socket;
            clientSockets.push( socket );

            socket.addEventListener( air.ProgressEvent.SOCKET_DATA, socketDataHandler);
            socket.addEventListener( air.Event.CLOSE, onClientClose );
```

```
            socket.addEventListener( air.IOErrorEvent.IO_ERROR, onIOError );

            //Send a connect message
            socket.writeUTFBytes("Connected.");
            socket.flush();

            air.trace( "Sending connect message" );
        }

        function socketDataHandler( event )
        {
            var socket = event.target

            //Read the message from the socket
            var message = socket.readUTFBytes( socket.bytesAvailable );
            air.trace( "Received: " + message);
            // Echo the received message back to the sender
            message = "Echo -- " + message;
            socket.writeUTFBytes( message );
            socket.flush();
            air.trace( "Sending: " + message );
        }

        function onClientClose( event )
        {
            air.trace( "Connection to client closed." );
            //Should also remove from clientSockets array...
        }
        function onIOError( errorEvent )
        {
            air.trace( "IOError: " + errorEvent.text );
        }
        function onClose( event )
        {
            air.trace( "Server socket closed by OS." );
        }
</script>
</head>
<body onload="startServer()">
</body>
</html>
```

詳しくは、以下を参照してください。

- ServerSocket
- ServerSocketConnectEvent
- Socket

UDP ソケット（AIR）

Adobe AIR 2 以降

ユニバーサルデータグラムプロトコル（UDP）は、ステートレスネットワーク接続経由でメッセージを交換する手段です。UDP では、メッセージの配信順序またはメッセージの配信自体が保証されません。UDP を使用すると、通常、オペレーティングシステムのネットワークコードでは、メッセージのマーシャリング、追跡および応答にかかる時間が短縮されます。したがって、一般的に、UDP メッセージが送信先のアプリケーションに到着するまでの遅延時間は TCP メッセージよりも短くなります。

UDP ソケット通信は、ゲームでの位置の更新や音声チャットアプリケーションの音声パケットなど、リアルタイムな情報を送信する必要がある場合に役に立ちます。このようなアプリケーションでは、ある程度のデータ損失は許容されるので、受信が保証されるよりも伝送待ち時間が短縮される方が重要です。その他の場合は、ほとんどすべての用途に TCP ソケットの方が適しています。

AIR アプリケーションでは、DatagramSocket クラスおよび DatagramSocketDataEvent クラスを使用して、UDP メッセージを送受信できます。UDP メッセージを送受信するには：

1. DatagramSocket オブジェクトを作成します。
2. data イベントに対するイベントリスナーを追加します。
3. bind() メソッドを使用して、ソケットを IP アドレスとポートにバインドします。
4. send() メソッドを呼び出し、送信先コンピューターの IP アドレスとポートを渡してメッセージを送信します。
5. data イベントに応答してメッセージを受信します。このイベントに対して送出された DatagramSocketDataEvent オブジェクトには、メッセージデータが格納された ByteArray オブジェクトが含まれています。

次のコード例では、アプリケーションで UDP メッセージを送受信する方法を示します。この例では、文字列「Hello」を含む単一メッセージを送信先コンピューターに送信します。また、すべての受信メッセージをトレースします。

```
<html>
<head>
<script src="AIRAliases.js"></script>
<script language="javascript">
    var datagramSocket;

    //The IP and port for this computer
    var localIP = "192.168.0.1";
    var localPort = 55555;

    //The IP and port for the target computer
    var targetIP = "192.168.0.2";
    var targetPort = 55555;

    function createDatagramSocket()
    {
        //Create the socket
        datagramSocket = new air.DatagramSocket();
        datagramSocket.addEventListener( air.DatagramSocketDataEvent.DATA, dataReceived );

        //Bind the socket to the local network interface and port
        datagramSocket.bind( localPort, localIP );

        //Listen for incoming datagrams
        datagramSocket.receive();
```

```
        //Create a message in a ByteArray
        var data = new air.ByteArray();
        data.writeUTFBytes("Hello.");

        //Send the datagram message
        datagramSocket.send( data, 0, 0, targetIP, targetPort);
    }

    function dataReceived( event )
    {
        //Read the data from the datagram
        air.trace("Received from " + event.srcAddress + ":" + event.srcPort + "> " +
            event.data.readUTFBytes( event.data.bytesAvailable ) );
    }
</script>
</head>
<body onload="createDatagramSocket()">
</body>
</html>
```

UDP ソケットを使用する場合は、以下の事項を考慮してください。

- 単一パケットのデータは、送信者と受信者との間にあるすべてのネットワークノードまたはネットワークインターフェイスの最大転送ユニット（MTU）の最小値を超えることはできません。send() メソッドに渡される ByteArray オブジェクトのデータはすべて単一パケットとして送信されます（TCP では、サイズの大きいメッセージは複数のパケットに分割されます）。
- 送信元と送信先との間にハンドシェイクは確立されません。送信先が存在しないか、指定されたポートにアクティブなリスナーが設定されていない場合、メッセージは破棄され、エラーは通知されません。
- connect() メソッドを使用する場合、他の送信元から送信されたメッセージは無視されます。UDP 接続では、簡易的なパケットフィルタリング機能しか提供されません。ただし、送信先アドレスとポートで必ずしも監視プロセスが有効になっているわけではありません。
- UDP トラフィックによってネットワークが圧迫される可能性があります。ネットワークで輻輳が発生する場合、ネットワーク管理者はサービス品質の管理機能を実装する必要があります（TCP にはビルトインのトラフィック管理機能があり、ネットワークでの輻輳による影響を軽減できます）。

詳しくは、以下を参照してください。

- DatagramSocket
- DatagramSocketDataEvent
- ByteArray

IPv6 アドレス

Flash Player 9 以降、Adobe AIR 1.0 以降

Flash Player 9.0.115.0 以降では、IPv6（インターネットプロトコルバージョン 6）がサポートされています。IPv6 は、128 ビットのアドレスをサポートするインターネットプロトコルのバージョンです（32 ビットのアドレスをサポートする、以前の IPv4 プロトコルの機能を向上したものです）。ネットワークインターフェイスでの IPv6 のアクティブ化が必要になる場合があります。詳細については、データをホストするオペレーティングシステムのヘルプを参照してください。

ホストしているシステムで IPv6 がサポートされる場合、次に示すように、角括弧（[]）内に URL の IPv6 数値リテラルアドレスを指定できます。

```
[2001:db8:ccc3:ffff:0:444d:555e:666f]
```

Flash Player では、以下の規則に従ってリテラル IPv6 値が返されます。

- Flash Player では IPv6 アドレスの長い形式の文字列が返されます。
- IP 値には二重コロンの省略形は含まれません。
- 16 進数は小文字のみです。
- IPv6 アドレスは角括弧（[]）で囲まれています。
- アドレスの各 4 桁は 0 ～ 4 桁の 16 進数として出力され、先頭のゼロは省略されます。
- アドレスの各 4 桁がすべてゼロの場合は、以下に示す例外を除いて、二重コロンではなく、1 個のゼロとして出力されます。

Flash Player で返される IPv6 の値には、以下の例外があります。

- 不明な IPv6 アドレス（すべてゼロ）は [::] として出力されます。
- ループバックまたは localhost の IPv6 アドレスは [::1] として出力されます。
- IPv4 のマップされた（IPv6 に変換された）アドレスは [::ffff:a.b.c.d] として出力されます。a.b.c.d は、一般的な IPv4 のドット付き 10 進表記の値です。
- IPv4 と互換性のあるアドレスは [::a.b.c.d] として出力されます。a.b.c.d は、一般的な IPv4 のドット付き 10 進表記の値です。

第 23 章：HTTP 通信

Flash Player 9 以降、Adobe AIR 1.0 以降

Adobe® AIR® および Adobe® Flash® Player アプリケーションでは、HTTP ベースのサーバーと通信して、データ、イメージおよびビデオのロードやメッセージのやりとりを行うことができます。

このトピックでは、AIR ネットワーキングと通信 API について説明します。これらは、ランタイムで実行されているアプリケーションにのみ提供される機能です。Web ブラウザー内で動作する HTML や JavaScript に固有のネットワーキングおよび通信機能（XMLHttpRequest クラスの使用に関する詳細など）をすべて説明しているわけではありません。

関連項目

flash.net.URLLoader

flash.net.URLStream

flash.net.URLRequest

flash.net.URLRequestDefaults

flash.net.URLRequestHeader

flash.net.URLRequestMethod

flash.net.URLVariables

外部データのロード

Flash Player 9 以降、Adobe AIR 1.0 以降

AIR ランタイムには、外部ソースからデータを読み込むためのメカニズムがあります。外部ソースは、テキストファイルなどの静的コンテンツや、Web スクリプトによって生成されるコンテンツなどの動的コンテンツを提供します。データは様々な形式で書式設定されている可能性があります。そのため、ランタイムには、データのデコードおよびデータへのアクセスのための機能が用意されています。また、データを取得する際に、外部サーバーにデータを送信することもできます。

URLRequest クラスの使用

Flash Player 9 以降、Adobe AIR 1.0 以降

外部データをロードする API の多くは、URLRequest クラスを使用して必要なネットワーク要求のプロパティを定義します。

URLRequest プロパティ

Flash Player 9 以降、Adobe AIR 1.0 以降

次の URLRequest オブジェクトのプロパティは、任意のセキュリティサンドボックス内で設定できます。

プロパティ	説明
contentType	URL 要求で送信されるデータの MIME コンテンツタイプです。contentType が設定されていない場合、値は application/x-www-form-urlencoded 形式で送信されます。
data	URL 要求で送信されるデータを含むオブジェクトです。
digest	Adobe® Flash® Player キャッシュに保存（または Adobe® Flash® Player キャッシュから取得）される署名付き Adobe プラットフォームコンポーネントを一意に識別するストリングです。
method	GET や POST などの HTTP 要求メソッドです。AIR アプリケーションのセキュリティドメインで実行されているコンテンツでは、"GET" や "POST" 以外のストリングを method プロパティとして指定できます。任意の HTTP 動詞を指定できます。デフォルトのメソッドは "GET" です。68 ページの「AIR のセキュリティ」を参照してください。
requestHeaders	HTTP 要求に追加される HTTP 要求ヘッダーの配列です。アプリケーションセキュリティサンドボックスの外部で実行されている Flash Player および AIR コンテンツには、ヘッダーの設定に一定の制限が設けられています。
url	要求される URL を指定します。

URLRequest クラスには、AIR アプリケーションセキュリティサンドボックス内のコンテンツでのみ使用可能な次のプロパティが含まれています。

プロパティ	説明
followRedirects	後でリダイレクトするかどうかを指定します。リダイレクトする場合は true（デフォルト値）、リダイレクトしない場合は false を指定します。これは AIR アプリケーションサンドボックスのみでサポートされます。
manageCookies	HTTP プロトコルスタックで、この要求の Cookie を管理するかどうかを指定します。管理する場合は true（デフォルト値）、管理しない場合は false を指定します。このプロパティの設定は AIR アプリケーションサンドボックスのみでサポートされます。
authenticate	この要求の認証要求を処理するかどうかを指定します。処理する場合は true を指定します。このプロパティの設定は AIR アプリケーションサンドボックスのみでサポートされます。デフォルトでは要求を認証します。そのため、サーバーが資格情報を必要とする場合、認証ダイアログボックスが表示されることがあります。 URLRequestDefaults クラスを使用して、ユーザー名やパスワードを設定することもできます。310 ページの「URLRequest のデフォルトの設定（AIR のみ）」を参照してください。
cacheResponse	この要求に対する応答データをキャッシュするかどうかを指定します。このプロパティの設定は AIR アプリケーションサンドボックスのみでサポートされます。デフォルトでは応答をキャッシュします（true）。
useCache	この URLRequest がデータを取得する前にローカルキャッシュを調べるかどうかを指定します。このプロパティの設定は AIR アプリケーションサンドボックスのみでサポートされます。デフォルト（true）ではローカルキャッシュバージョンを使用します。
userAgent	HTTP 要求でユーザーエージェントストリングを使用するよう指定します。

URLRequest のデフォルトの設定（AIR のみ）

Adobe AIR 1.0 およびそれ以降

URLRequestDefaults クラスを使用すると、URLRequest オブジェクトに対して、アプリケーション固有のデフォルト設定を定義できます。例えば、次のコードでは、manageCookies プロパティおよび useCache プロパティのデフォルト値を設定しています。新たに作成されるすべての URLRequest オブジェクトでは、標準のデフォルト値ではなく、指定した値がこれらのプロパティで使用されます。

```
air.URLRequestDefaults.manageCookies = false;
air.URLRequestDefaults.useCache = false;
```

注意：URLRequestDefaults クラスは、Adobe AIR で実行されているコンテンツに対してのみ定義されます。Flash Player ではサポートされていません。

URLRequestDefaults クラスには、setLoginCredentialsForHost() メソッドが含まれています。このメソッドを使用すると、特定のホストに対して使用するデフォルトのユーザー名とパスワードを指定できます。ホストは、このメソッドの hostname パラメーターで指定します。ホストには、ドメイン（"www.example.com" など）またはドメインとポート番号（"www.example.com:80" など）を指定できます。"example.com"、"www.example.com"、および "sales.example.com" は、それぞれ固有のホストと見なされることに注意してください。

これらの資格情報は、サーバーから要求された場合にのみ使用されます。ユーザーがすでに認証されている場合（認証ダイアログボックスを使用した場合など）は、setLoginCredentialsForHost() メソッドを呼び出しても、認証されたユーザーが変更されることはありません。

次のコードでは、www.example.com に対して送信する要求で使用する、デフォルトのユーザー名とパスワードを設定します。

```
air.URLRequestDefaults.setLoginCredentialsForHost("www.example.com", "Ada", "love1816$X");
```

URLRequestDefaults の設定は、現在のアプリケーションドメインのみに適用されます。ただし、例外的に、setLoginCredentialsForHost() メソッドに渡される資格情報は、AIR アプリケーション内で、任意のアプリケーションドメインで実行された要求に対して使用されます。

詳しくは、『HTML 開発者用 Adobe AIR API リファレンスガイド』の URLRequestDefaults クラスの項を参照してください。

URI スキーム

Flash Player 9 以降、Adobe AIR 1.0 以降

標準の URI スキーム（以下参照）は、任意のセキュリティサンドボックスから実行された要求で使用できます。

http: および https:
標準のインターネット URL で使用します（Web ブラウザーで使用する場合と同じ方法）。

file:
ローカルファイルシステムに保存されているファイルの URL を指定するには、file: を使用します。次に、例を示します。

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

AIR では、アプリケーションセキュリティサンドボックス内で実行されているコンテンツの URL を定義する際には、次のスキームを使用することもできます。

app:
app: は、インストールされたアプリケーションのルートディレクトリを基準とした相対的なパスを指定する場合に使用します。例えば、次のパスは、インストールされているアプリケーションのディレクトリの resources サブディレクトリを参照しています。

```
app:/resources
```

AIR Debug Launcher（ADL）を使用して AIR アプリケーションを起動する場合、アプリケーションディレクトリはアプリケーション記述ファイルが含まれているディレクトリです。

File.applicationDirectory で作成した File オブジェクトの URL（および url プロパティ）では、次のように app URI スキームを使用します。

```
var dir = air.File.applicationDirectory;
dir = dir.resolvePath("assets");
air.trace(dir.url); // app:/assets
```

app-storage:

app-storage: は、アプリケーションのデータ記憶領域ディレクトリを基準とした相対的なパスを指定する場合に使用します。AIR では、インストールされたアプリケーション（およびユーザー）ごとに、一意のアプリケーション記憶領域ディレクトリが作成されます。このディレクトリは、そのアプリケーション固有のデータを格納するための便利な場所として使用できます。例えば、次のパスは、アプリケーション記憶領域ディレクトリの settings サブディレクトリにある prefs.xml ファイルを参照しています。

```
app-storage:/settings/prefs.xml
```

File.applicationStorageDirectory メソッドで作成した File オブジェクトの URL（および url プロパティ）では、次のように app-storage URI スキームを使用します。

```
var prefsFile = air.File.applicationStorageDirectory;
prefsFile = prefsFile.resolvePath("prefs.xml");
air.trace(prefsFile.url); // app-storage:/prefs.xml
```

mailto:

mailto スキームは、navigateToURL() 関数に渡される URLRequest オブジェクトで使用できます。321 ページの「別のアプリケーションで URL を開く」を参照してください。

前述の URI スキームを使用する URLRequest オブジェクトを使用すると、多数の異なるオブジェクト（FileStream オブジェクトや Sound オブジェクトなど）に対する URL 要求を定義できます。また、これらのスキームは、AIR 内で実行されている HTML コンテンツ（img タグの src 属性など）で使用することもできます。

ただし、AIR 固有の URI スキーム（app: および app-storage:）は、アプリケーションセキュリティサンドボックス内のコンテンツでしか使用できません。詳しくは、68 ページの「AIR のセキュリティ」を参照してください。

URL 変数の設定

変数は URL ストリングに直接追加できますが、URLVariables クラスを使用して要求に必要な変数を定義する方が簡単な場合があります。

URLVariables オブジェクトにパラメーターを追加するには、3 つの方法があります。

- URLVariables コンストラクターで指定する方法
- URLVariables.decode() メソッドで指定する方法
- URLVariables オブジェクト自体の動的プロパティとして設定する方法

次の例は、3 つのメソッドすべてと、URLRequest オブジェクトに変数を割り当てる方法を示しています。

```
var urlVar = new air.URLVariables( "one=1&two=2" );
urlVar.decode("amp=" + air.encodeURIComponent( "&" ) );
urlVar.three = 3;
urlVar.amp2 = "&&";
air.trace(urlVar.toString()); //amp=%26&amp2=%26%26&one=1&two=2&three=3

var urlRequest = new air.URLRequest( "http://www.example.com/test.cfm" );
urlRequest.data = urlVar;
```

URLVariables コンストラクターまたは URLVariables.decode() メソッドで変数を定義する場合は必ず、URI ストリング内の特別な意味を持つ文字を URL エンコードしてください。例えば、パラメーター名またはパラメーター値でアンパサンドを使用する場合、アンパサンドはパラメーターの区切り文字として機能するので、& を %26 に変更してアンパサンドをエンコードする必要があります。これには、トップレベルの encodeURIComponent() 関数を使用できます。

URLLoader クラスの使用

Flash Player 9 以降、Adobe AIR 1.0 以降

URLLoader クラスを使用すると、サーバーに要求を送信したり、返された情報にアクセスしたりできます。ローカルファイルへのアクセスが許可されているコンテキストで、ローカルファイルシステム上のファイルにアクセスする場合にも、URLLoader クラスを使用できます。URLLoader クラスは、指定した URL からテキスト、バイナリデータ、または URL エンコード形式の変数をダウンロードする際に使用します。URLLoader クラスは、complete、httpStatus、ioError、open、progress、securityError などのイベントを送出します。

URLLoader クラスは、XMLHttpRequest クラスの代替手段を提供します。このいずれかのクラスを使用して、HTTP 要求経由でデータをダウンロードできます。

ダウンロードデータは、ダウンロードが完了するまで使用できません。ダウンロードの進捗状況（ロード済みバイト数と合計バイト数）は、progress イベントの送出をリッスンすることで監視できます。ただし、ファイルのロードの完了が早すぎると progress イベントは送出されない場合があります。ファイルが正常にダウンロードされた場合、complete イベントが送出されます。URLLoader の dataFormat プロパティを設定することにより、データをテキスト、生のバイナリデータまたは URLVariables オブジェクト形式で受信できます。

URLLoader.load() メソッドのパラメーターは、URLRequest オブジェクトを指定する request パラメーターの 1 つだけです（URLLoader クラスのコンストラクターにも、必要に応じて同じパラメーターを指定できます）。URLRequest インスタンスには、1 件の HTTP 要求に関して、ターゲット URL、要求メソッド（GET または POST）、付加的なヘッダー情報、MIME タイプおよびその他すべての情報が格納されます。

例えば、XML パケットをサーバーサイドスクリプトにアップロードする場合のコードは次のようになります。

```
var secondsUTC = new Date().time;
var dataXML = (new DOMParser()).parseFromString( "<time>" + secondsUTC + "</time>", "application/xml" );
var request = new air.URLRequest("http://www.example.com/time.cfm");
request.contentType = "text/xml";
request.data = dataXML;
request.method = air.URLRequestMethod.POST;
var loader = new air.URLLoader();
loader.load(request);
```

上記のコードスニペットでは、サーバーに送信される XML パケットを格納する dataXML という名前の XML ドキュメントを作成します。この例では、URLRequest の contentType プロパティを "text/xml" に設定し、XML ドキュメントを URLRequest の data プロパティに割り当てます。最後に、URLLoader オブジェクトを作成し、load() メソッドを使用してリモートスクリプトに要求を送信します。

URLStream クラスの使用

Flash Player 9 以降、Adobe AIR 1.0 以降

URLStream クラスを使用すると、データを受信している最中にダウンロードデータにアクセスできます。URLStream クラスでは、ダウンロードの完了前にストリームを閉じることもできます。ダウンロードされたデータは、生のバイナリデータ形式で利用できます。

URLStream オブジェクトからデータを読み込むときは、bytesAvailable プロパティを使用して、読み取るデータが十分にあるかどうかを事前に確認します。読み取り可能なデータ量よりも多くのデータを読み取ろうとすると、EOFError 例外がスローされます。

httpResponseStatus イベント（AIR）
URLStream クラスは、応答データが配信される前に、httpResponseStatus イベントを送出します。HTTPStatusEvent クラスで表わされる httpResponseStatus イベントには、応答を返した URL を示す responseURL プロパティと、応答によって返された応答ヘッダーを表す URLRequestHeader オブジェクトの配列である responseHeaders プロパティが含まれています。

外部ドキュメントからのデータのロード

Flash Player 9 以降、Adobe AIR 1.0 以降

動的なアプリケーションを作成する場合、外部のファイルまたはサーバーサイドスクリプトからデータをロードするようにすると、アプリケーションの編集と再コンパイルを必要としない動的なアプリケーションを実現できます。例えば、「今日の一言」アプリケーションを作成する場合、サーバーサイドスクリプトで、データベースからランダムに選んだ一言を取得して 1 日 1 回テキストファイルに保存するようにします。そうすれば、アプリケーションでは静的なテキストファイルをロードするだけでよく、毎回データベースへのクエリを発行する必要はなくなります。

次のコードでは、URLRequest オブジェクトと URLLoader オブジェクトを作成し、それによって "params.txt" という外部のテキストファイルから内容をロードします。

```
var request = new air.URLRequest("params.txt");
var loader = new air.URLLoader();
loader.load(request);
```

デフォルトでは、要求メソッドを定義しない場合、Flash Player および Adobe AIR では HTTP GET メソッドを使用してコンテンツがロードされます。POST メソッドで要求を送信する場合は、次のように、静的定数 URLRequestMethod.POST を使用して request.method プロパティに POST を設定します。

```
var request = new air.URLRequest("http://www.example.com/sendfeedback.cfm");
request.method = air.URLRequestMethod.POST;
```

実行時に読み込まれる外部ドキュメント params.txt の内容は次のようなデータです。

```
monthNames=January,February,March,April,May,June,July,August,September,October,November,December&dayNames
=Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday
```

このファイルには、monthNames および dayNames の 2 つのパラメーターが含まれています。各パラメーターの内容は、ストリングとして解析されるカンマ区切りリストです。このリストは String.split() メソッドを使用して分割し、配列に変換できます。

外部データファイル内の変数名には、予約語やプログラミング言語の構成要素を含めないでください。そのような変数名を使用すると、コードの解読やデバッグ作業が困難になります。

データの読み込みが完了すると complete イベントが送出され、次のように、URLLoader の data プロパティで外部ドキュメントのコンテンツを使用できるようになります。

```
function completeHandler(event)
{
    var loader2 = URLLoader(event.target);
    air.trace(loader2.data);
}
```

リモートドキュメントに名前と値のペアが格納されている場合は、URLVariables クラスを使用し、読み込んだファイルのコンテンツを次のように渡すことでデータを解析できます。

```
function completeHandler(event)
{
    var loader2 = event.target;
    var variables = new air.URLVariables(loader2.data);
    air.trace(variables.dayNames);
}
```

外部ファイルから読み込んだ個々の名前 / 値ペアから、URLVariables オブジェクトの個別のプロパティが作成されます。このコード例にある variables オブジェクトの各プロパティは、ストリングとして扱われます。名前 / 値ペアの値がアイテムのリストである場合は、次のように String.split() メソッドを呼び出すことでストリングから配列に変換できます。

```
var dayNameArray = variables.dayNames.split(",");
```

外部テキストファイルから数値データを読み込むには、トップレベル関数の parseInt()、parseFloat()、Number() などを使用して値を数値に変換する必要があります。

リモートファイルの内容を文字列としてロードして新しい URLVariables オブジェクトを作成する方法とは別に、URLLoader.dataFormat プロパティを URLLoaderDataFormat クラスの静的プロパティのいずれかに設定する方法もあります。URLLoader.dataFormat プロパティに設定できる値は次の 3 つのうちいずれかです。

- URLLoaderDataFormat.BINARY - URLLoader.data プロパティには、ByteArray オブジェクトに格納されたバイナリデータが含められます。
- URLLoaderDataFormat.TEXT - URLLoader.data プロパティには、String オブジェクトのテキストが含められます。
- URLLoaderDataFormat.VARIABLES - URLLoader.data プロパティには、URLVariables オブジェクトに格納された URL エンコードされた変数が含められます。

次のコードでは URLLoader.dataFormat プロパティを URLLoaderDataFormat.VARIABLES に設定しているので、ロードしたデータは自動的に解析されて URLVariables オブジェクトが作成されます。

```
var request = new air.URLRequest("http://www.example.com/params.txt");
var variables = new air.URLLoader();
variables.dataFormat = air.URLLoaderDataFormat.VARIABLES;
variables.addEventListener(air.Event.COMPLETE, completeHandler);
try
{
    variables.load(request);
}
catch (error)
{
    air.trace("Unable to load URL: " + error);
}

function completeHandler(event)
{
    var loader = event.target;
    air.trace(loader.data.dayNames);
}
```

注意：URLLoader.dataFormat のデフォルト値は URLLoaderDataFormat.TEXT です。

次の例に示すように、外部ファイルから XML を読み込む方法も、URLVariables を読み込む場合と同じです。URLRequest インスタンスと URLLoader インスタンスを作成し、それらを使用してリモート XML ドキュメントをダウンロードします。ファイルのダウンロードが完了すると、complete イベントが送出され、trace() 関数によってファイルのコンテンツがコマンドラインに出力されます。

```
var request = new air.URLRequest("http://www.example.com/data.xml");
var loader = new air.URLLoader();
loader.addEventListener(air.Event.COMPLETE, completeHandler);
loader.load(request);

function completeHandler(event)
{
    var dataXML = event.target.data;
    air.trace(dataXML);
}
```

外部スクリプトとの通信

Flash Player 9 以降、Adobe AIR 1.0 以降

URLVariables クラスを使用すると、外部データファイルからデータをロードできるだけでなく、サーバーサイドスクリプトに変数を送信し、サーバーの応答を処理することもできます。例えば、ゲームを作成する場合、ユーザーの得点をサーバーに送信して高得点者リストに掲載できるかどうか確認することや、ユーザーのログイン情報をサーバーに送信して検証することなどが考えられます。サーバーサイドスクリプトでは、ユーザー名とパスワードを処理してデータベースに照会し、入力されたユーザー資格情報が有効かどうかの確認を応答として返します。

次のコードでは、variables という URLVariables オブジェクトを作成し、このオブジェクトを使用して name という新しい変数を作成しています。次に、変数の送信先となるサーバーサイドスクリプトの URL を指定する URLRequest オブジェクトを作成します。その後、URLRequest オブジェクトの method プロパティに、HTTP POST 要求で変数を送信するように設定します。URL 要求に URLVariables オブジェクトを追加するには、前の例で作成した URLVariables オブジェクトを URLRequest オブジェクトの data プロパティに設定します。最後に、URLLoader インスタンスを作成し、URLLoader.load() メソッドを呼び出して要求を送信します。

```
var variables = new air.URLVariables("name=Franklin");
var request = new air.URLRequest();
request.url = "http://www.[yourdomain].com/greeting.cfm";
request.method = air.URLRequestMethod.POST;
request.data = variables;
var loader = new air.URLLoader();
loader.dataFormat = URLLoaderDataFormat.VARIABLES;
loader.addEventListener(Event.COMPLETE, completeHandler);
try
{
    loader.load(request);
}
catch (error)
{
    air.trace("Unable to load URL");
}

function completeHandler(event)
{
    air.trace(event.target.data.welcomeMessage);
}
```

次のコードには、前の例で使用された ColdFusion® greeting.cfm ドキュメントのコンテンツが含まれています。

```
<cfif NOT IsDefined("Form.name") OR Len(Trim(Form.Name)) EQ 0>
    <cfset Form.Name = "Stranger" />
</cfif>
<cfoutput>welcomeMessage=#UrlEncodedFormat("Welcome, " & Form.name)#
</cfoutput>
```

Web サービス要求

Flash Player 9 以降、Adobe AIR 1.0 以降

HTTP ベースの Web サービスにはさまざまなタイプがあります。主なタイプを以下に示します。

- REST
- XML-RPC
- SOAP[SOAP]

ActionScript 3 で Web サービスを利用するには、URLRequest オブジェクトを作成し、URL 変数または XML ドキュメントを使用して Web サービス呼び出しを作成し、URLLoader オブジェクトを使用してサービスに呼び出しを送信します。Flex フレームワークには、Web サービスの利用を容易にするいくつかのクラスが用意されています。これらのクラスは、複雑な SOAP サービスにアクセスする場合に特に役立ちます。Flash Professional CS3 以降では、Flash Professional で開発したアプリケーションおよび Flash Builder で開発したアプリケーション内で、Flex のクラスを使用できます。

HTML ベースの AIR アプリケーションでは、URLRequest クラスと URLLoader クラスの組み合わせ、または Javascript の XMLHttpRequest クラスのいずれかを使用できます。必要に応じて、Flex フレームワークの Web サービスコンポーネントを Javascript コードに公開する SWF ライブラリを作成することもできます。

アプリケーションをブラウザーで実行する場合は、Web サービスをホストしているサーバーが他のドメインからのアクセスを許可するクロスドメインポリシーファイルをホストしている場合を除き、呼び出し側の SWF と同じインターネットドメイン内にある Web サービスのみを利用できます。クロスドメインポリシーファイルが用意されていない場合は、自分のサーバーをプロキシとして要求を中継するテクニックがよく使用されます。Adobe Blaze DS および Adobe LiveCycle では、Web サービスのプロキシをサポートしています。

AIR アプリケーションでは、アプリケーションセキュリティサンドボックスから Web サービス呼び出しを実行する場合、クロスドメインポリシーファイルは必要ありません。AIR アプリケーションのコンテンツがリモートドメインから提供されることはありません。したがって、クロスドメインポリシーファイルによって防御されている攻撃にさらされる可能性はありません。HTML ベースの AIR アプリケーションでは、アプリケーションセキュリティサンドボックス内のコンテンツからクロスドメイン XMLHttpRequest を実行できます。また、コンテンツが iframe にロードされる場合に限り、他のセキュリティサンドボックス内のコンテンツからクロスドメイン XMLHttpRequest を実行できます。

関連項目

Adobe BlazeDS

Adobe LiveCycle ES2

REST アーキテクチャ

XML-RPC

SOAP プロトコル

REST スタイルの Web サービス要求

Flash Player 9 以降、Adobe AIR 1.0 以降

REST スタイルの Web サービスでは、HTTP メソッド（動詞）を使用して基本操作を指定し、URL 変数で操作の詳細を指定します。例えば、アイテムのデータを取得する要求では、GET 動詞と URL 変数を使用して、メソッド名とアイテム ID を指定することができます。つまり、URL ストリングは次のようになります。

```
http://service.example.com/?method=getItem&id=d3452
```

ActionScript で REST スタイルの Web サービスにアクセスするには、URLRequest、URLVariables および URLLoader の各クラスを使用できます。AIR アプリケーション内の Javascript コードでは、XMLHttpRequest も使用できます。

REST スタイルの Web サービス呼び出しを ActionScript でプログラミングする場合、通常は次の手順を実行します。

1. URLRequest オブジェクトを作成します。
2. URLRequest で、サービス URL および HTTP メソッド（動詞）を設定します。
3. URLVariables オブジェクトを作成します。
4. サービス呼び出しのパラメーターを URLVariables オブジェクトの動的プロパティとして設定します。
5. URLRequest オブジェクトの data プロパティに URLVariables オブジェクトを割り当てます。

6 URLLoader オブジェクトを使用して、サービスに呼び出しを送信します。

7 URLLoader から送出される complete イベントを処理します。このイベントは、サービス呼び出しが完了したことを示します。URLLoader オブジェクトから送出される可能性のあるエラーイベントを監視することもお勧めします。

例えば、呼び出しパラメーターを送信元にエコーバックするテストメソッドを公開している Web サービスについて検討してみましょう。次の ActionScript コードは、サービスの呼び出しに使用できます。

```
function restServiceCall()
{
    //Create the HTTP request object
    var request = new air.URLRequest( "http://service.example.com/" );
    request.method = air.URLRequestMethod.GET;

    //Add the URL variables
    var variables = new air.URLVariables();
    variables.method = "test.echo";
    variables.api_key = "123456ABC";
    variables.message = "Able was I, ere I saw Elba.";
    request.data = variables;

    //Initiate the transaction
    window.requestor = new air.URLLoader();
    requestor.addEventListener( air.Event.COMPLETE, httpRequestComplete );
    requestor.addEventListener( air.IOErrorEvent.IOERROR, httpRequestError );
    requestor.addEventListener( air.SecurityErrorEvent.SECURITY_ERROR, httpRequestError );
    requestor.load( request );
}
function httpRequestComplete( event )
{
    air.trace( event.target.data );
}

function httpRequestError( error ){
    air.trace( "An error occured: " + error.message );
}
```

AIR アプリケーション内の Javascript では、XMLHttpRequest オブジェクトを使用して同様の要求を実行できます。

```
<html>
<head><title>RESTful web service request</title>
<script type="text/javascript">

function makeRequest()
{
    var requestDisplay = document.getElementById( "request" );
    var resultDisplay  = document.getElementById( "result" );

    //Create a conveninece object to hold the call properties
    var request = {};
    request.URL = "http://service.example.com/";
    request.method = "test.echo";
    request.HTTPmethod = "GET";
    request.parameters = {};
    request.parameters.api_key = "ABCDEF123";
    request.parameters.message = "Able was I ere I saw Elba.";
    var requestURL = makeURL( request );
    xmlhttp = new XMLHttpRequest();
    xmlhttp.open( request.HTTPmethod, requestURL, true);
    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4) {
            resultDisplay.innerHTML = xmlhttp.responseText;
        }
```

```
    }
    xmlhttp.send(null);

    requestDisplay.innerHTML = requestURL;
}
//Convert the request object into a properly formatted URL
function makeURL( request )
{
    var url = request.URL + "?method=" + escape( request.method );
    for( var property in request.parameters )
    {
        url += "&" + property + "=" + escape( request.parameters[property] );
    }

    return url;
}
</script>
</head>
<body onload="makeRequest()">
<h1>Request:</h1>
<div id="request"></div>
<h1>Result:</h1>
<div id="result"></div>
</body>
</html>
```

XML-RPC Web サービス要求

Flash Player 9 以降、Adobe AIR 1.0 以降

XML-RPC Web サービスでは、呼び出しパラメーターを URL 変数のセットとしてではなく XML 形式のドキュメントとして取得します。XML-RPC Web サービスでトランザクションを実行するには、適切にフォーマットされた XML メッセージを作成し、そのメッセージを HTTP POST メソッドを使用して Web サービスに送信します。また、要求に Content-Type ヘッダーを設定して、サーバーが要求データを XML 形式で扱えるようにする必要があります。

次の例では、DOM メソッドを使用して XML-RPC メッセージを作成し、XMLHttpRequest を使用して Web サービストランザクションを実行します。

```
<html>
<head>
<title>XML-RPC web service request</title>
<script type="text/javascript">

function makeRequest()
{
    var requestDisplay = document.getElementById( "request" );
    var resultDisplay  = document.getElementById( "result" );

    var request = {};
    request.URL = "http://services.example.com/xmlrpc/";
    request.method = "test.echo";
    request.HTTPmethod = "POST";
    request.parameters = {};
    request.parameters.api_key = "123456ABC";
    request.parameters.message = "Able was I ere I saw Elba.";
    var requestMessage = formatXMLRPC( request );

    xmlhttp = new XMLHttpRequest();
    xmlhttp.open( request.HTTPmethod, request.URL, true);
    xmlhttp.onreadystatechange = function() {
```

```
        if (xmlhttp.readyState == 4) {
            resultDisplay.innerText = xmlhttp.responseText;
        }
    }
    xmlhttp.send( requestMessage );

    requestDisplay.innerText = xmlToString( requestMessage.documentElement );
}

//Formats a request as XML-RPC document
function formatXMLRPC( request )
{
    var xmldoc = document.implementation.createDocument( "", "", null );
    var root = xmldoc.createElement( "methodCall" );
    xmldoc.appendChild( root );
    var methodName = xmldoc.createElement( "methodName" );
    var methodString = xmldoc.createTextNode( request.method );
    methodName.appendChild( methodString );

    root.appendChild( methodName );

        var params = xmldoc.createElement( "params" );
        root.appendChild( params );

        var param = xmldoc.createElement( "param" );
        params.appendChild( param );
        var value = xmldoc.createElement( "value" );
        param.appendChild( value );
        var struct = xmldoc.createElement( "struct" );
        value.appendChild( struct );

        for( var property in request.parameters )
        {
            var member = xmldoc.createElement( "member" );
            struct.appendChild( member );

            var name = xmldoc.createElement( "name" );
            var paramName = xmldoc.createTextNode( property );
            name.appendChild( paramName )
            member.appendChild( name );

            var value = xmldoc.createElement( "value" );
            var type = xmldoc.createElement( "string" );
            value.appendChild( type );
            var paramValue = xmldoc.createTextNode( request.parameters[property] );
            type.appendChild( paramValue )
            member.appendChild( value );
        }
    return xmldoc;
}

//Returns a string representation of an XML node
function xmlToString( rootNode, indent )
{
    if( indent == null ) indent = "";
    var result = indent + "<" + rootNode.tagName + ">\n";
    for( var i = 0; i < rootNode.childNodes.length; i++)
    {
        if(rootNode.childNodes.item( i ).nodeType == Node.TEXT_NODE )
        {
            result += indent + "    " + rootNode.childNodes.item( i ).textContent + "\n";
        }
```

```
    }
    if( rootNode.childElementCount > 0 )
    {
        result += xmlToString( rootNode.firstElementChild, indent + "    " );
    }
    if( rootNode.nextElementSibling )
    {
        result += indent + "</" + rootNode.tagName + ">\n";
        result += xmlToString( rootNode.nextElementSibling, indent );
    }
    else
    {
        result += indent +"</" + rootNode.tagName + ">\n";
    }
    return result;
}

</script>
</head>
<body onload="makeRequest()">
<h1>Request:</h1>
<pre id="request"></pre>
<h1>Result:</h1>
<pre id="result"></pre>
</body>
</html>
```

別のアプリケーションで URL を開く

Flash Player 9 以降、Adobe AIR 1.0 以降

navigateToURL() 関数を使用すると、デフォルトのシステム Web ブラウザーで URL を開くことができます。

この関数の request パラメーターとして URLRequest オブジェクトを渡す場合は、url プロパティのみが使用されます。

navigateToURL() 関数の最初のパラメーターである navigate は URLRequest オブジェクトです（309 ページの「URLRequest クラスの使用」を参照してください）。2 番目のパラメーターである window では、ウィンドウ名を指定できます。このパラメーターはオプションです。例えば、次のコードでは、Web ページ www.adobe.com を開いています。

```
var url = "http://www.adobe.com";
var urlReq = new air.URLRequest(url);
air.navigateToURL(urlReq);
```

注意：navigateToURL() 関数を使用する場合、ランタイムは GET メソッドを使用する場合と同じように、（method プロパティが URLRequestMethod.POST に設定されている）POST メソッドを使用する URLRequest オブジェクトを扱います。

navigateToURL() 関数を使用する場合、使用可能な URI スキームは、navigateToURL() 関数を呼び出しているコードのセキュリティサンドボックスに基づいて決定されます。

一部の API では、Web ブラウザーでコンテンツを起動することができます。これらの API を AIR で使用する場合、セキュリティ上の理由のため、一部の URI スキームの使用が禁止されています。使用できないスキームは、API を使用するコードのセキュリティサンドボックスに応じて異なります。（セキュリティサンドボックスについて詳しくは、68 ページの「AIR のセキュリティ」を参照してください）。

アプリケーションサンドボックス（AIR のみ）
AIR アプリケーションサンドボックス内で実行しているコンテンツによって起動された URL では、どの URI スキームも使用できます。アプリケーションは URI スキームを処理するよう登録する必要があり、そうしなければリクエストをしても何も実行されません。次のスキームは多くのコンピューターやデバイスでサポートされています。

- http:
- https:
- file:
- mailto: — AIR は、登録されているシステムの電子メールアプリケーションにこれらの要求を送ります。
- sms : — AIR は、デフォルトのテキストメッセージアプリケーションに sms : リクエストを送ります。アプリケーションを実行するシステムの規則に準拠した URL の形式を使用する必要があります。例えば、Android の URI スキームは小文字です。

```
navigateToURL( new URLRequest( "sms:+15555550101") );
```

- tel : — AIR は、デフォルトの電話ダイヤルアプリケーションに tel : リクエストを送ります。アプリケーションを実行するシステムの規則に準拠した URL の形式を使用する必要があります。例えば、Android の URI スキームは小文字です。

```
navigateToURL( new URLRequest( "tel:5555555555") );
```

- market : — AIR は、Android デバイスで通常サポートされている Market アプリケーションに market : リクエストを送ります。

```
navigateToURL( new URLRequest( "market://search?q=Adobe Flash") );
navigateToURL( new URLRequest( "market://search?q=pname:com.adobe.flashplayer") );
```

オペレーティングシステムに許可されている場所であれば、アプリケーションはカスタム URI スキームを定義および登録できます。スキームを使用する URL を作成し、AIR からアプリケーションを起動できます。

リモートサンドボックス
次のスキームを使用できます。これらのスキームの使用方法は、Web ブラウザーで使用する場合と同様です。

- http:
- https:
- mailto: — AIR は、登録されているシステムの電子メールアプリケーションにこれらの要求を送ります。

他の URI スキームはすべて使用できません。

Local-with-file サンドボックス
次のスキームを使用できます。これらのスキームの使用方法は、Web ブラウザーで使用する場合と同様です。

- file:
- mailto: — AIR は、登録されているシステムの電子メールアプリケーションにこれらの要求を送ります。

他の URI スキームはすべて使用できません。

Local-with-networking サンドボックス
次のスキームを使用できます。これらのスキームの使用方法は、Web ブラウザーで使用する場合と同様です。

- http:
- https:
- mailto: — AIR は、登録されているシステムの電子メールアプリケーションにこれらの要求を送ります。

他の URI スキームはすべて使用できません。

Local-trusted サンドボックス

次のスキームを使用できます。これらのスキームの使用方法は、Web ブラウザーで使用する場合と同様です。

- file:
- http:
- https:
- mailto: — AIR は、登録されているシステムの電子メールアプリケーションにこれらの要求を送ります。

他の URI スキームはすべて使用できません。

サーバーへの URL の送信

Flash Player 9 以降、Adobe AIR 1.0 以降

sendToURL() 関数を使用すると、サーバーに URL 要求を送信できます。この関数は、サーバーからの応答を無視します。sendToURL() 関数の引数は、URLRequest オブジェクトを指定する request の 1 つだけです（309 ページの「URLRequest クラスの使用」を参照してください）。次に例を示します。

```
var url = "http://www.example.com/application.jsp";
var variables = new air.URLVariables();
variables.sessionId = new Date().getTime();
variables.userLabel = "Your Name";
var request = new air.URLRequest(url);
request.data = variables;
air.sendToURL(request);
```

この例では、URLVariables クラスを使用して URLRequest オブジェクトに変数データを追加しています。詳しくは、313 ページの「URLLoader クラスの使用」を参照してください。

第 24 章: Flash Player および AIR の他のインスタンスとの通信

Flash Player 9 以降、Adobe AIR 1.0 以降

LocalConnection クラスを使用すると、ブラウザーで実行される SWF コンテンツ間だけでなく、Adobe® AIR® アプリケーション間でも通信できます。また、LocalConnection クラスを使用して、AIR アプリケーションとブラウザーで実行される SWF コンテンツとの間で通信することもできます。LocalConnection クラスを使用すれば、Flash Player および AIR の複数のインスタンス間でデータを共有し、用途の広いアプリケーションを作成できます。

LocalConnection クラスについて

Flash Player 9 以降、Adobe AIR 1.0 以降

LocalConnection オブジェクトが通信できるのは、同じクライアントコンピューター上で動作する AIR アプリケーションと SWF ファイルとの間に限られます。ただし、同じアプリケーションで動作している必要はありません例えば、AIR アプリケーションとブラウザーで実行される SWF ファイルで可能なように、LocalConnection クラスを使用して 2 つの AIR アプリケーションで通信できます。

LocalConnection オブジェクトの最も簡単な使用方法は、同じドメイン内または同じ AIR アプリケーション内の LocalConnection オブジェクト間の通信のみを許可することです。この場合、セキュリティに関する問題への対処は不要です。しかし、異なるドメイン間で通信を行う必要がある場合は、セキュリティ対策を実施する必要があります。いくつかの方法があります。詳しくは、send() メソッドの connectionName パラメーターおよび LocalConnection クラスの allowDomain() と domain のエントリに関する説明を参照してください。このクラスは、『Adobe Flash Platform 用 ActionScript 3.0 リファレンスガイド』のリストに含まれています。

LocalConnection オブジェクトにコールバックメソッドを追加するには、次のコードに示すように、LocalConnection.client プロパティに、メンバーメソッドを持つオブジェクトを設定します。

```
 var lc = new air.LocalConnection();
var clientObject = new Object();
clientObject.doMethod1 = function() {
    air.trace("doMethod1 called.");
}
clientObject.doMethod2 = function(param1) {
    air.trace("doMethod2 called with one parameter: " + param1);
    air.trace("The square of the parameter is: " + param1 * param1);
}
lc.client = clientObject;
```

LocalConnection.client プロパティに、呼び出すことができるすべてのコールバックメソッドが含まれます。

isPerUser プロパティ

isPerUser プロパティは、複数のユーザーが Mac コンピューターにログインしているときに発生する競合を解決するために、Flash Player （10.0.32）と AIR （1.5.2）に追加されました。他のオペレーティングシステムでは、ローカル接続は常に個別のユーザーが対象なので、このプロパティは無視されます。新規に作成するコードでは、isPerUser プロパティを true に設定することが必要です。ただし、後方互換性のために、デフォルト値は現在 false です。このデフォルト値は、今後のバージョンのランタイムで変更される可能性があります。

2 つのアプリケーション間でのメッセージ送信

Flash Player 9 以降、Adobe AIR 1.0 以降

LocalConnection クラスを使用すると、異なる AIR アプリケーション間でも、ブラウザーで実行される異なる Adobe® Flash® Player（SWF）アプリケーション間でも通信できます。また、LocalConnection クラスを使用して、AIR アプリケーションとブラウザーで実行される SWF アプリケーションとの間で通信することもできます。

次のコードでは、サーバーとして動作し、他のアプリケーションから受信する LocalConnection 呼び出しを受け付ける LocalConnection オブジェクトを定義します。

```
var lc = new air.LocalConnection();
lc.connect("connectionName");
var clientObject = new Object();
clientObject.echoMsg = function(msg) {
    air.trace("This message was received: " + msg);
}
lc.client = clientObject;
```

このコードでは、まず、lc という LocalConnection オブジェクトを作成し、client プロパティを clientObject オブジェクトに設定します。この LocalConnection インスタンスのメソッドを別のアプリケーションが呼び出すと、ランタイムは clientObject オブジェクトでそのメソッドを探します。

指定した名前の接続が既に存在する場合は Argument Error 例外がスローされます。この例外は、オブジェクトが接続済みであるために接続の試行が失敗したことを示します。

次のコードでは、conn1 という名前を指定して LocalConnection を作成する方法を示します。

```
connection.connect("conn1");
```

第 1 のアプリケーションに第 2 のアプリケーションから接続するには、まず、送信側 LocalConnection オブジェクト内で LocalConnection オブジェクトを作成し、次に、接続名と実行するメソッドの名前を指定して LocalConnection.send() メソッドを呼び出す必要があります。例えば、前述の例で作成した LocalConnection オブジェクトに doQuit メソッドを送信するには、次のコードを使用します。

```
sendingConnection.send("conn1", "doQuit");
```

このコードでは、既存の LocalConnection オブジェクトに接続名 conn1 で接続し、リモートアプリケーションの doMessage() メソッドを呼び出しています。リモートアプリケーションにパラメーターを送信する場合は、次のコードに示すように、send() メソッドに対するメソッド名の指定の後に追加パラメーターを指定します。

```
sendingConnection.send("conn1", "doMessage", "Hello world");
```

異なるドメインのコンテンツおよび AIR アプリケーションへの接続

Flash Player 9 以降、Adobe AIR 1.0 以降

特定のドメインからの通信のみ許可するには、LocalConnection クラスの allowDomain() メソッドまたは allowInsecureDomain() メソッドを呼び出し、この LocalConnection オブジェクトへのアクセスを許可するドメインのリストを渡し、許可するドメインの 1 つ以上の名前を渡します。

LocalConnection.allowDomain() メソッドおよび LocalConnection.allowInsecureDomain() メソッドに渡すことができる特別な値として、* と localhost の 2 つがあります。アスタリスク値（*）は、すべてのドメインにアクセスを許可することを示します。localhost というストリングは、ローカル（ただし、アプリケーションリソースディレクトリ外）にインストールされているコンテンツから該当のアプリケーションへの呼び出しを許可することを示します。

LocalConnection.send() メソッドで、呼び出し元コードでのアクセスが認められていないアプリケーションにセキュリティサンドボックスから通信しようとすると、securityError イベント（SecurityErrorEvent.SECURITY_ERROR）が送出されます。このエラーを回避するには、受信側の LocalConnection.allowDomain() メソッドで、呼び出し元のドメインを指定してください。

同じドメイン内のコンテンツ間のみで通信を行う場合は、connectionName パラメーターに対して、先頭がアンダースコア（_）以外で、かつドメイン名を含まない名前を指定します（例：myDomain:connectionName）。また、それと同じストリングを LocalConnection.connect(connectionName) コマンドでも使用します。

異なるドメインにあるコンテンツ間での通信を実装する場合は、connectionName パラメーターに対して、アンダースコアで始まる名前を指定します。このアンダースコアを付けると、受信側 LocalConnection オブジェクトを含むコンテンツのドメイン間におけるポータビリティが高まります。考えられる 2 つの状況を次に示します。

- connectionName のストリングがアンダースコア（_）で始まっていない場合、myDomain:connectionName のように、スーパードメイン名とコロンの接頭辞がランタイムによって追加されます。この処理には他のドメインにある同じ名前の接続との競合を回避する意味がありますが、その代わり、このスーパードメインをすべての送信側 LocalConnection オブジェクトで指定する必要があります（例：myDomain:connectionName）。受信側 LocalConnection オブジェクトを含んだ HTML ファイルまたは SWF ファイルを別のドメインに移動すると、ランタイムによって追加される接頭辞は、スーパードメインの変更を反映して変化します（例：anotherDomain:connectionName）。したがって、新しいスーパードメインを参照するようにすべての送信側 LocalConnection オブジェクトを手動で編集する必要が生じます。
- connectionName のストリングが、_connectionName のようにアンダースコアで始まっている場合、ストリングに接頭辞は追加されません。つまり、受信側と送信側の LocalConnection オブジェクトは、connectionName にまったく同じストリングを使用します。すべてのドメインからの接続を受け付けることを受信側オブジェクトが LocalConnection.allowDomain() によって指定していれば、受信側 LocalConnection オブジェクトを含んだ HTML ファイルまたは SWF ファイルを別のドメインに移動した場合でも、送信側 LocalConnection オブジェクトに変更を加える必要はありません。

  connectionName でアンダースコア付きの名前を使用する場合の欠点の 1 つは、2 つのアプリケーションの両方が同じ connectionName を使用して接続しようとする場合など、競合の可能性があることです。もう 1 つの欠点は、セキュリティ関連です。アンダースコアシンタックスを使用する接続名では、リッスンしているアプリケーションのドメインは識別されません。これらの理由から、ドメイン修飾名の使用が推奨されます。

Adobe AIR

AIR アプリケーションセキュリティサンドボックス内で実行するコンテンツ（AIR アプリケーションでインストールされたコンテンツ）と通信するには、接続名に AIR アプリケーションを識別するスーパードメイン名を接頭辞として付加する必要があります。スーパードメインのストリングは、app# で始まり、アプリケーション ID、ドット（.）文字、発行者 ID（定義されている場合）と続きます。例えば、アプリケーション ID が com.example.air.MyApp で、発行者 ID はないアプリケーションに対して、connectionName パラメーターで使用する適切なスーパードメインは、「app#com.example.air.MyApp」です。このようにして、接続名のベース部分が appConnection の場合、connectionName パラメーターで使用されるストリング全体は、「app#com.example.air.MyApp:appConnection」となります。アプリケーションに発行者 ID がある場合は、その ID もスーパードメインのストリングに組み込んで、
「app#com.example.air.MyApp.B146A943FBD637B68C334022D304CEA226D129B4.1」などとする必要があります。

ローカル接続を通じて別の AIR アプリケーションからユーザーのアプリケーションに通信できるようにする場合は、LocalConnection オブジェクトの allowDomain() を呼び出し、ローカル接続のドメイン名を渡す必要があります。AIR アプリケーションでは、このドメイン名は接続ストリングと同じ要領でアプリケーション ID と発行者 ID により構成されます。

例えば、送信側 AIR アプリケーションのアプリケーション ID が com.example.air.FriendlyApp で、発行者 ID が 214649436BD677B62C33D02233043EA236D13934.1 である場合、このアプリケーションから接続するときに使用するドメインストリングは「app#com.example.air.FriendlyApp.214649436BD677B62C33D02233043EA236D13934.1」になります。（AIR 1.5.3 の時点では、すべての AIR アプリケーションに発行者 ID が付いているわけではありません）。

第 25 章：JavaScript 開発者のための ActionScript の基礎

Adobe® ActionScript® 3.0 は JavaScript に似たプログラミング言語です。どちらも ECMAScript に基づいています。Adobe® Flash® Player 9 でリリースされた ActionScript 3.0 を使用すると、Adobe® Flash® CS3 Professional、Adobe® Flash® CS4 Professional および Adobe® Flex™ 3 でリッチインターネットアプリケーションを開発できます。

現行バージョンの ActionScript 3.0 は、ブラウザーの Flash Player 9 用 SWF コンテンツの開発時にのみ使用できました。今回、Adobe® AIR® で実行する SWF コンテンツの開発時にも使用できるようになりました。

HTML 開発者用 Adobe AIR API リファレンスガイドには、HTML ベースのアプリケーションの JavaScript コードで役立つクラスの情報が含まれています。これは、ランタイムのクラスセット全体のサブセットです。ランタイムの他のクラス（ビジュアルコンテンツの構造を定義する DisplayObject クラスなど）は、SWF ベースのアプリケーションの開発に役立ちます。これらのクラスを JavaScript で使用する際は、次に示す ActionScript マニュアルを参照してください。

- **ActionScript 3.0** 開発ガイド
- **Adobe Flash Platform** 用 **ActionScript 3.0** リファレンスガイド（AIR で実行される HTML コンテンツで使用できるのは、flash パッケージ内の最上位のクラスと関数だけです。mx パッケージ内のクラスは、Flex ベースの SWF アプリケーションでのみ使用できます）

ActionScript と JavaScript の相違点について

ActionScript は、JavaScript と同じ ECMAScript 言語仕様に基づいています。したがって、両言語の基本的シンタックスは共通しています。例えば、次のコードは JavaScript でも ActionScript でも同様に機能します。

```
var str1 = "hello";
var str2 = " world.";
var str = reverseString(str1 + str2);

function reverseString(s) {
    var newString = "";
    var i;
    for (i = s.length - 1; i >= 0; i--) {
        newString += s.charAt(i);
    }
    return newString;
}
```

しかし、これら 2 つの言語のシンタックスと機能には相違点もあります。例えば、前述のコード例は ActionScript 3.0（SWF ファイル）で次のように記述することもできます。

```
function reverseString(s:String):String {
    var newString:String = "";
    for (var i:int = s.length - 1; i >= 0; i--) {
        newString += s.charAt(i);
    }
    return newString;
}
```

Adobe AIR の HTML コンテンツでサポートされる JavaScript のバージョンは JavaScript 1.7 です。このトピックでは、JavaScript 1.7 と ActionScript 3.0 の違いについて説明します。

ランタイムには、高度な機能を提供するビルトインクラスがあります。HTML ページの JavaScript から、実行時にこれらのクラスにアクセスできます。これらのランタイムクラスは、ActionScript（SWF ファイル）と JavaScript（ブラウザーで実行される HTML ファイル）の両方で使用できます。ただし、現在、これらのクラスの API ドキュメント（『HTML 開発者用 Adobe AIR API リファレンスガイド』には含まれていません）では、ActionScript のシンタックスを使用して説明されています。このため、ランタイムの高度な機能の一部については、『**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。ActionScript に関する基礎知識は、JavaScript におけるこれらのランタイムクラスの使用方法を理解するのに役立ちます。

例として、MP3 ファイルのサウンドを再生する JavaScript コードを次に示します。

```
var file = air.File.userDirectory.resolve("My Music/test.mp3");
var sound = air.Sound(file);
sound.play();
```

これらの各コード行では、JavaScript のランタイム機能が呼び出されます。

SWF ファイルでは、次に示す ActionScript コードでこれらのランタイム機能にアクセスできます。

```
var file:File = File.userDirectory.resolve("My Music/test.mp3");
var sound = new Sound(file);
sound.play();
```

ActionScript 3.0 のデータ型

ActionScript 3.0 は、厳密な型指定を行う言語です。つまり、1 つの変数に 1 つのデータ型を割り当てることができます。例えば、前述の例の第 1 行は次のように記述することもできます。

```
var str1:String = "hello";
```

ここでは、str1 変数が String 型として宣言されています。後続のすべての str1 変数には、ストリング値が割り当てられます。

変数、関数のパラメーター、関数の戻り値の型に対して、型を割り当てることができます。したがって、前述の例における関数の宣言を ActionScript で記述すると次のようになります。

```
function reverseString(s:String):String {
       var newString:String = "";
       for (var i:int = s.length - 1; i >= 0; i--) {
           newString += s.charAt(i);
       }
       return newString;
}
```

注意：関数の s パラメーターと戻り値の両方に、String 型が割り当てられます。

ActionScript では型指定を省略できますが、次のようにオブジェクトの型を宣言することによる利点もあります。

- オブジェクトが型指定されていると、実行時だけでなく、厳密モードによるコンパイル時にもデータの型チェックが行われます。コンパイル時の型チェックはエラーの特定に役立ちます（厳密モードはコンパイラーのオプションです）。
- 型指定されたオブジェクトを使用すると、効率性の高いアプリケーションを作成できます。

  このため、ActionScript マニュアルの例ではデータ型が使用されています。多くの場合、型宣言（":String"）を削除するだけで、ActionScript のコード例を JavaScript に変換できます。

カスタムクラスに対応するデータ型

ActionScript 3.0 オブジェクトには、String、Number、Date などの最上位クラスに対応するデータ型を指定できます。

ActionScript 3.0 では、カスタムクラスを定義できます。各カスタムクラスでもデータ型を定義します。したがって、そのクラスによって定義されたアノテーション型が、ActionScript 変数、関数パラメーターまたは関数の戻り値に付与される場合があります。詳しくは、330 ページの「ActionScript 3.0 のカスタムクラス」を参照してください。

void データ型

void データ型は、実際には値を返さない関数の戻り値として使用します（return ステートメントを含まない関数は値を返しません）。

* データ型

アスタリスク文字（*）をデータ型として使用することは、データ型を割り当てないことと同じです。例えば、次の関数では、パラメーター n と戻り値のどちらにもデータ型が割り当てられていません。

```
function exampleFunction(n:*):* {
       trace("hi, " + n);
}
```

* をデータ型として使用した場合、データ型がまったく定義されません。ActionScript 3.0 コードでは、データ型を定義しないことを明示するためにアスタリスクを使用します。

ActionScript 3.0 のクラス、パッケージおよび名前空間

ActionScript 3.0 には、JavaScript 1.7 に存在しないクラスに関する機能が含まれています。

ランタイムクラス

Array、Date、Math、String など、ランタイムに組み込まれているクラスの多くは、標準の JavaScript にも含まれています。ただし、ランタイムには、標準の JavaScript には存在しないクラスも含まれています。これらの追加的なクラスには、リッチメディア（サウンドなど）の再生からソケットの操作まで、幅広い用途があります。

ほとんどのランタイムクラスは、flash パッケージ内か、flash パッケージに含まれているいずれかのパッケージ内にあります。パッケージは、ActionScript 3.0 クラスを編成する手法です（331 ページの「ActionScript 3.0 のパッケージ」を参照）。

ActionScript 3.0 のカスタムクラス

ActionScript 3.0 では、開発者が独自のカスタムクラスを作成できます。例えば、次のコードでは ExampleClass というカスタムクラスを定義しています。

```
public class ExampleClass {
    public var x:Number;
    public function ExampleClass(input:Number):void {
        x = input;
    }
    public function greet():void {
        trace("The value of x is: ", x);
    }
}
```

このクラスのメンバは次のとおりです。

- コンストラクターメソッド ExampleClass()。これを使用して、ExampleClass 型の新しいオブジェクトをインスタンス化できます。

- パブリックプロパティ x（Number 型）。ExampleClass 型のオブジェクトに対して取得および設定できます。
- パブリックメソッド greet()。ExampleClass 型のオブジェクトで呼び出すことができます。

  この例の x プロパティと greet() メソッドは、public 名前空間にあります。public 名前空間によって、メソッドおよびプロパティはクラス外部のオブジェクトやクラスからもアクセスできるようになります。

ActionScript 3.0 のパッケージ

パッケージは、ActionScript 3.0 クラスを整理する手段を提供します。例えば、コンピューター上のファイルやディレクトリの操作に関連する多くのクラスは、flash.filesystem パッケージに含まれています。この場合、flash というパッケージの中に、filesystem という別のパッケージが含まれています。さらに、そのパッケージに他のクラスやパッケージが含まれていることもあります。実際、この flash.filesystem パッケージには File、FileMode、FileStream の各クラスが含まれています。ActionScript で File クラスを参照するには、次のように記述します。

```
flash.filesystem.File
```

ビルトインクラスとカスタムクラスのどちらもパッケージに編成できます。

JavaScript から ActionScript パッケージを参照するには、特殊な runtime オブジェクトを使用します。例えば、次のコードでは、JavaScript で ActionScript の新しい File オブジェクトをインスタンス化します。

```
var myFile = new air.flash.filesystem.File();
```

このコードの File() メソッドは、同じ名前（File）のクラスに対応するコンストラクター関数です。

ActionScript 3.0 の名前空間

ActionScript 3.0 では、名前空間を使用して、アクセス可能なクラス内のプロパティと関数のスコープを定義します。

public 名前空間にあるプロパティとメソッドだけを、JavaScript で使用できます。

例えば、（flash.filesystem パッケージ内の）File クラスに含まれるパブリックプロパティとメソッドには、userDirectory や resolve() などがあります。どちらも、File オブジェクトを（runtime.flash.filesystem.File() コンストラクターメソッド経由で）インスタンス化する JavaScript 変数のプロパティとして使用できます。

定義済みの名前空間には、次の 4 つがあります。

名前空間	説明
public	ある型のオブジェクトをインスタンス化するコードでは、その型を定義しているクラス内のパブリックプロパティとパブリックメソッドにアクセスできます。また任意のコードで、public クラスのパブリック静的プロパティとパブリック静的メソッドにアクセスできます。
private	private に指定されたプロパティとメソッドは、クラス内部のコードでのみ使用できます。そのクラスで定義されたオブジェクトのプロパティやメソッドとしてアクセスすることはできません。private 名前空間のプロパティとメソッドは、JavaScript で使用することができません。
protected	protected に指定されたプロパティとメソッドは、クラス定義内のコードと、そのクラスを継承するクラスでのみ使用できます。protected 名前空間のプロパティとメソッドは、JavaScript で使用することができません。
internal	internal に指定されたプロパティとメソッドは、同じパッケージ内部の任意の呼び出し元で使用できます。デフォルトでは、クラス、プロパティおよびメソッドは internal 名前空間に属します。

また、カスタムクラスでは JavaScript コードで使用できないその他の名前空間を使用できます。

ActionScript 3.0 関数の必須パラメーターとデフォルト値

ActionScript 3.0 でも JavaScript でも、関数にパラメーターを含めることができます。ActionScript 3.0 のパラメーターは、必須の場合とオプションの場合があります。これに対して、JavaScript のパラメーターは常にオプションです。

次の ActionScript 3.0 コードで定義される関数では、n が必須パラメーターです。

```
function cube(n:Number):Number {
    return n*n*n;
}
```

次の ActionScript 3.0 コードで定義される関数では、n が必須パラメーターです。また、p はオプションのパラメータで、デフォルト値は 1 です。

```
function root(n:Number, p:Number = 1):Number {
    return Math.pow(n, 1/p);
}
```

ActionScript 3.0 関数では、任意の数の引数を受け取ることもできます。その場合、次に示すように、パラメーターのリストの末尾に ...rest シンタックスで記述します。

```
function average(... args) : Number{
    var sum:Number = 0;
    for (var i:int = 0; i < args.length; i++) {
        sum += args[i];
    }
    return (sum / args.length);
}
```

ActionScript 3.0 イベントリスナー

ActionScript 3.0 のプログラミングでは、すべてのイベントをイベントリスナーで処理します。イベントリスナーは関数です。オブジェクトからイベントが送出されると、イベントリスナーがイベントに応答します。ActionScript オブジェクトの一種であるイベントは、関数のパラメータとしてイベントリスナーに渡されます。このようなイベントオブジェクトの使用方法は、JavaScript で使用される DOM イベントモデルとは異なります。

例えば、Sound オブジェクトの load() メソッドを（MP3 ファイルを読み込むために）呼び出すと、Sound オブジェクトはサウンドの読み込みを試みます。Sound オブジェクトは、その後で、次に示すイベントのいずれかを送出します。

イベント	**説明**
complete	データが正常に読み込まれたときに送出されます。
id3	MP3 ID3 データが使用可能なときに送出されます。
ioError	入出力エラーが発生して読み込み操作が失敗したときに送出されます。
open	読み込み操作が開始されたときに送出されます。
progress	読み込み処理の実行中にデータを受信したときに送出されます。

イベントを送出できるクラスは、EventDispatcher クラスを拡張するか、IEventDispatcher インターフェイスを実装します（ActionScript 3.0 インターフェイスは、クラスで実装可能な一連のメソッドの定義に使用するデータ型です）。『ActionScript リファレンスガイド』のこれらのクラスの項目には、各クラスで送出可能なイベントの一覧が記載されています。

これらのイベントを処理するイベントリスナー関数を登録するには、イベントを送出するオブジェクトの addEventListener() メソッドを使用します。例えば Sound オブジェクトの場合、次の ActionScript コードに示すように、progress イベントと complete イベントを登録できます。

```
var sound:Sound = new Sound();
var urlReq:URLRequest = new URLRequest("test.mp3");
sound.load(urlReq);
sound.addEventListener(ProgressEvent.PROGRESS, progressHandler);
sound.addEventListener(Event.COMPLETE, completeHandler);

function progressHandler(progressEvent):void {
      trace("Progress " + progressEvent.bytesTotal + " bytes out of " + progressEvent.bytesTotal);
}

function completeHandler(completeEvent):void {
      trace("Sound loaded.");
}
```

AIR で実行する HTML コンテンツで、JavaScript 関数をイベントリスナーとして登録できます。次のコードはこの例を示しています（ここでは、HTML ドキュメントに progressTextArea という名前の TextArea オブジェクトが含まれていると仮定しています）。

```
var sound = new runtime.flash.media.Sound();
var urlReq = new runtime.flash.net.URLRequest("test.mp3");
sound.load(urlReq);
sound.addEventListener(runtime.flash.events.ProgressEvent.PROGRESS, progressHandler);
sound.addEventListener(runtime.flash.events.Event.COMPLETE, completeHandler);

function progressHandler(progressEvent) {
      document.progressTextArea.value += "Progress " + progressEvent.bytesTotal + " bytes out of " +
progressEvent.bytesTotal;
}

function completeHandler(completeEvent) {
      document.progressTextArea.value += "Sound loaded.";
```

第 26 章：ローカルデータベースでの SQL サポート

Adobe AIR には、SQL データベースエンジンが含まれています。このデータベースエンジンは、オープンソースの SQLite データベースシステムを使用して、多くの標準 SQL 機能を備えたローカル SQL データベースをサポートします。ランタイムでは、データベースのデータをファイルシステムのどこに、どのように格納するのかは指定しません。各データベースは、完全に単一ファイル内に格納されます。開発者は、データベースファイルを格納する場所をファイルシステム内に指定できます。また、1 つの AIR アプリケーションから 1 つまたは複数の異なるデータベース（別々のデータベースファイル）にアクセスできます。このドキュメントでは、Adobe AIR ローカル SQL データベースでサポートされる SQL 構文とデータ型について概説します。このドキュメントは、包括的な SQL リファレンスとしての利用を想定したものではなく、Adobe AIR でサポートする SQL ダイアレクトの詳細を説明したものです。このランタイムは、SQL-92 標準 SQL ダイアレクトの大部分をサポートしています。SQL 学習用には様々な参考文献、Web サイト、書籍、およびトレーニング資料が存在するので、このドキュメントでは、包括的な SQL のリファレンスまたはチュートリアルの体裁はとらず、AIR でサポートしている SQL 構文について、また、このダイアレクトと SQL-92 との違いについての説明を主眼としています。

SQL ステートメントの定義で使用する表記規則

このドキュメントに記載された SQL 文の定義では、次の表記規則を使用しています。

- テキストの大文字と小文字
  - UPPER CASE - リテラルの SQL キーワードはすべて大文字で表記されています。
  - lower case - プレースホルダー項または句の名前はすべて小文字で表記されています。
- 定義文字
  - ::= 句またはステートメントの定義を示します。
- グループおよび置換可能を示す文字
  - | パイプ文字は置換可能なオプションの間で使用されており、「または」に読み替えることができます。
  - [] 角括弧で囲まれた項目はオプション項目です。角括弧の中には、単一の項目、または互いに置換可能な複数の項目が記載されています。
  - () 置換可能な複数の項目（パイプ文字で区切られた項目）が丸括弧で囲まれている場合は、必須の項目グループ（つまり、単一の必須項目に対して有効な値を示す項目のセット）を示します。
- 繰り返し制御文字
  - ＋丸括弧で囲まれた項目の後にプラス文字が付いている場合は、前の項目を 1 回以上繰り返すことができることを示します。
  - * 角括弧で囲まれた項目の後にアスタリスク文字が付いている場合は、前の（角括弧で囲まれた）項目を 0 回以上繰り返すことができることを示します。
- リテラル文字
  - * 列名、または関数名の後の丸括弧の中でアスタリスク文字が使用されている場合は、0 回以上の繰り返しではなく、リテラルのアスタリスク文字を示します。
  - . ピリオド文字はリテラルのピリオドを表します。
  - , カンマ文字はリテラルのカンマを表します。
  - () 単一の句または項目を囲む 1 組の丸括弧は、その丸括弧が必須のリテラル丸括弧文字であることを示します。
  - その他の文字は、特に記載されていない限り、そのリテラル文字を表します。

サポートされている SQL 構文

Adobe AIR SQL データベースエンジンでサポートされている SQL 構文について、ステートメントと句の種類、式、ビルトイン関数および演算子に分けて説明します。ここで取り上げるトピックは次のとおりです。

- SQL 構文全般
- データ操作ステートメント（SELECT、INSERT、UPDATE および DELETE）
- データ定義ステートメント（テーブル、インデックス、ビューおよびトリガーの CREATE ステートメント、ALTER ステートメントおよび DROP ステートメント）
- 特殊な文と句
- ビルトイン関数（集計関数、スカラ関数、および日付 / 時刻書式設定関数）
- 演算子
- パラメーター
- サポートされていない SQL 機能
- 追加の SQL 機能

SQL 構文全般

各種の文と式に関する具体的な構文に加えて、SQL 構文には一般に次のような規則があります。

大文字と小文字の区別 SQL ステートメント（オブジェクト名を含む）では、大文字と小文字は区別されません。ただし、SQL 文では SQL キーワードが大文字で表記されることが多いので、このドキュメントでもこの表記規則を使用します。SQL シンタックスでは大文字と小文字が区別されませんが、SQL のリテラルテキスト値は大文字と小文字が区別され、比較およびソート操作では、列または操作に対して定義されている照合シーケンスの指定に従って大文字と小文字を区別できます。詳しくは、「COLLATE」を参照してください。

空白 SQL ステートメントでは、個々の単語を区切るために空白文字（スペース、タブ、改行など）を使用する必要があります。ただし、単語と記号の間の空白はオプションです。SQL ステートメントに含まれる空白文字の種類と個数は意味を持ちません。インデントや改行などの空白文字を使用して SQL 文を読みやすいように整形できます。書式を整形しても、SQL 文の意味は変わりません。

データ操作文

データ操作文は最もよく使用する SQL 文です。データ操作文は、データベーステーブルのデータの取得、追加、変更、および削除に使用します。サポートされているデータ操作ステートメントは、SELECT、INSERT、UPDATE および DELETE です。

SELECT

SELECT 文は、データベースに問い合わせるときに使用します。SELECT 文の結果は、それぞれ固定数の列を持つ 0 行以上のデータ行です。結果として返される列の数は、SELECT とオプションの FROM キーワードとの間に列名または式リスト（下記の result）で指定します。

```
sql-statement   ::=  SELECT [ALL | DISTINCT] result
                     [FROM table-list]
                     [WHERE expr]
                     [GROUP BY expr-list]
                     [HAVING expr]
                     [compound-op select-statement]*
                     [ORDER BY sort-expr-list]
                     [LIMIT integer [( OFFSET | , ) integer]]
result          ::=  result-column [, result-column]*
result-column   ::=  * | table-name . * | expr [[AS] string]
table-list      ::=  table [ join-op table join-args ]*
table           ::=  table-name [AS alias] |
                     ( select ) [AS alias]
join-op         ::=  , | [NATURAL] [LEFT | RIGHT | FULL] [OUTER | INNER | CROSS] JOIN
join-args       ::=  [ON expr] [USING ( id-list )]
compound-op     ::=  UNION | UNION ALL | INTERSECT | EXCEPT
sort-expr-list  ::=  expr [sort-order] [, expr [sort-order]]*
sort-order      ::=  [COLLATE collation-name] [ASC | DESC]
collation-name  ::=  BINARY | NOCASE
```

結果（result）には任意の式を使用できます。結果の式が * の場合は、その 1 つの式がすべてのテーブルのすべての列に置き換えられます。テーブル名の後に .* を続けて指定すると、その 1 つのテーブルのすべての列が返されます。

DISTINCT キーワードを指定すると、結果行の重複しないサブセットが返されます。NULL 値はすべて同じものとして扱われます。デフォルトでは、すべての結果行が返されます。これを明示的に指定するには、キーワード ALL を使用します。

クエリは FROM キーワードの後に指定した 1 つ以上のテーブルに対して実行されます。複数のテーブル名をカンマで区切ると、各テーブルがクロス結合されます。JOIN 構文を使用して、テーブルの結合方法を指定することもできます。外部結合の種類としては、LEFT OUTER JOIN のみがサポートされています。join-args 内の ON 句の式はブール値に解決される必要があります。FROM 句のテーブルとして、括弧で囲んだサブクエリを使用できます。FROM 句全体を省略することもできます。この場合は、result 式リストの値で構成された 1 つの行が返されます。

WHERE 句は、クエリで取得する行数を限定する場合に使用します。WHERE 句の式はブール値に解決される必要があります。WHERE 句によるフィルタリングはグループ化の前に行われるので、WHERE 句の式に集計関数を含めることはできません。

GROUP BY 句を使用すると、1 行以上の結果が集約されて 1 行の出力行になります。GROUP BY 句は、結果に集約関数が含まれている場合に特に役立ちます。GROUP BY 句の式は、SELECT 式リストに含まれていなくてもかまいません。

HAVING 句は、ステートメントによって返される行を制限するという点で WHERE と同様です。ただし、HAVING 句は、GROUP BY 句によって指定されたグループ化が行われた後に適用されます。したがって、HAVING 式は、集約関数を含む値を参照できます。HAVING 句の式は、SELECT リストに含まれていなくてもかまいません。WHERE 句の式と同様に、HAVING 句の式もブール値に解決される必要があります。

ORDER BY 句を使用すると、出力行がソートされます。ORDER BY 句の sort-expr-list 引数には、ソートキーとして使用する式のリストを指定します。これらの式は、単純な SELECT 文では結果に含まれていなくてもかまいませんが、複合 SELECT 文（いずれかの compound-op 演算子を使用した SELECT 文）では、各ソート式が結果の列の 1 つに正確に対応する必要があります。各ソートキーの後にはオプションで sort-order 句を付けることができます。この句には、COLLATE キーワードとテキストの並べ替えに使用する照合関数の名前およびソート順序を指定するキーワード ASC（昇順）または DESC（降順）を指定できます。sort-order を省略すると、デフォルト（昇順）が使用されます。COLLATE 句と照合関数の定義については、「COLLATE」を参照してください。

LIMIT 句は、結果で返される行数の上限を設定します。負の LIMIT は上限がないことを示します。LIMIT の後にオプションの OFFSET を付けると、結果セットの先頭から指定した行数がスキップされます。複合 SELECT ステートメントでは、LIMIT 句は必ず最後の SELECT ステートメントの後に付けますが、上限はクエリ全体に適用されます。LIMIT 句で OFFSET キーワードを使用すると、最初の整数が上限になり、2 番目の整数がオフセットになります。OFFSET キーワードの代わりにカンマを使用すると、最初の数がオフセットになり、2 番目の数が上限になります。この見かけの違いは意図的なもので、古い SQL データベースシステムとの互換性を最大限確保することを目的としています。

複合 SELECT ステートメントは、UNION、UNION ALL、INTERSECT、EXCEPT のいずれかの演算子で接続された複数の単純な SELECT ステートメントで構成されます。複合 SELECT ステートメントでは、構成するすべての SELECT ステートメントで同じ数の結果列を指定する必要があります。最後の SELECT ステートメントの後（また、単一の LIMIT 句が指定されている場合はその前）には、単一の ORDER BY 句のみを指定できます。UNION 演算子と UNION ALL 演算子はどちらも、前と後の SELECT ステートメントの結果を 1 つのテーブルに結合します。違いは、UNION ではすべての結果行が異なり、UNION ALL では重複する可能性があることです。INTERSECT 演算子は、前と後の SELECT ステートメントの結果の共通部分をとります。EXCEPT は、前の SELECT 文の結果から後の SELECT 文の結果を除去します。3 つ以上の SELECT 文を接続すると、最初から順に結果が結合されます。

指定可能な式の定義については、「式」を参照してください。

AIR 2.5 以降では、BLOB データを ActionScript ByteArray オブジェクトに変換するための読み取りの際に、SQL CAST 演算子がサポートされます。例えば、次のコードは AMF 形式で格納されていない未処理のデータを読み取って、ByteArray オブジェクトに格納します。

```
stmt.text = "SELECT CAST(data AS ByteArray) AS data FROM pictures;";
stmt.execute();
var result:SQLResult = stmt.getResult();
var bytes:ByteArray = result.data[0].data;
```

INSERT

INSERT はテーブルにデータを格納するために使用する SQL 文で、次の 2 つの基本形式があります。

```
sql-statement  ::=  INSERT [OR conflict-algorithm] INTO [database-name.] table-name [(column-list)] VALUES
(value-list) |
                    INSERT [OR conflict-algorithm] INTO [database-name.] table-name [(column-list)] select-
statement
                     REPLACE INTO [database-name.] table-name [(column-list)] VALUES (value-list) |
                     REPLACE INTO [database-name.] table-name [(column-list)] select-statement
```

VALUES キーワードを使用する最初の形式では、既存のテーブルに新しい行が 1 行作成されます。column-list を指定しない場合は、テーブルの列と同じ数の値を指定する必要があります。column-list を指定する場合は、指定した列と同じ数の値を指定する必要があります。列リストに含まれないテーブルの列には、テーブルの作成時に定義されたデフォルト値が格納されます。デフォルト値が定義されていない場合は NULL が格納されます。

2 番目の形式の INSERT ステートメントは、SELECT ステートメントからデータを取得します。column-list を指定しない場合は、SELECT 文の結果に含まれる列の数とテーブルの列の数が一致する必要があります。column-list を指定する場合は、そこで指定した列の数と SELECT 文の結果に含まれる列の数が一致する必要があります。SELECT 文の結果行ごとに、テーブル内に新しいエントリが 1 つ作成されます。SELECT 文は単純な SELECT と複合 SELECT のどちらでもかまいません。指定可能な SELECT 文の定義については、「SELECT」を参照してください。

オプションの conflict-algorithm では、この特定のコマンドで使用する制約競合解決アルゴリズムを指定します。conflict-algorithm の説明および定義については、344 ページの「特殊な文と句」を参照してください。

2 つの REPLACE INTO 形式は、標準の INSERT [OR conflict-algorithm] 形式を REPLACE 競合アルゴリズムと共に使用する（つまり、INSERT OR REPLACE... 形式を使用する）のと同じです。

2 つの REPLACE INTO 形式は、標準の INSERT [OR conflict-algorithm] 形式を REPLACE 競合アルゴリズムと共に使用する（つまり、INSERT OR REPLACE... 形式を使用する）のと同じです。

UPDATE

UPDATE コマンドは、テーブル内の既存のレコードを変更します。

```
sql-statement  ::=  UPDATE [database-name.] table-name SET column1=value1, column2=value2,... [WHERE expr]
```

このコマンドは、UPDATE キーワードの後に、レコードを更新する対象のテーブルの名前を指定します。SET キーワードの後に、変更する列名と変更後の値を、カンマ区切りリストとして指定します。WHERE 句の式には、レコードが更新される行（複数の場合もあり）を指定します。

DELETE

DELETE 文は、テーブルからレコードを削除するときに使用します。

```
sql-statement  ::=  DELETE FROM [database-name.] table-name [WHERE expr]
```

このコマンドでは、DELETE FROM キーワードの後に、レコードを削除する対象のテーブルの名前を指定します。

WHERE 句を省略すると、テーブルのすべての行が削除されます。WHERE 句を指定すると、式に一致する行のみが削除されます。WHERE 句の式はブール値に解決される必要があります。指定可能な式の定義については、「式」を参照してください。

データ定義文

データ定義文は、テーブル、ビュー、インデックス、トリガーなどのデータベースオブジェクトを作成、変更、および削除するときに使用します。次のデータ定義文がサポートされています。

- テーブル：
  - CREATE TABLE
  - ALTER TABLE
  - DROP TABLE
- インデックス：
  - CREATE INDEX
  - DROP INDEX
- ビュー：
  - CREATE VIEWS
  - DROP VIEWS
- トリガー：
  - CREATE TRIGGERS
  - DROP TRIGGERS

CREATE TABLE

CREATE TABLE ステートメントでは、キーワード CREATE TABLE の後に新しいテーブルの名前を指定し、その後に列定義と列制約のリストを括弧で囲んで記述します。テーブル名には、識別子またはストリングを使用できます。

```
sql-statement       ::=  CREATE [TEMP | TEMPORARY] TABLE [IF NOT EXISTS] [database-name.] table-name
                         ( column-def [, column-def]* [, constraint]* )
sql-statement       ::=  CREATE [TEMP | TEMPORARY] TABLE [database-name.] table-name AS select-statement
column-def          ::=  name [type] [[CONSTRAINT name] column-constraint]*
type                ::=  typename | typename ( number ) | typename ( number , number )
column-constraint   ::=  NOT NULL [ conflict-clause ] |
                         PRIMARY KEY [sort-order] [ conflict-clause ] [AUTOINCREMENT] |
                         UNIQUE [conflict-clause] |
                         CHECK ( expr ) |
                         DEFAULT default-value |
                         COLLATE collation-name
constraint          ::=  PRIMARY KEY ( column-list ) [conflict-clause] |
                         UNIQUE ( column-list ) [conflict-clause] |
                         CHECK ( expr )
conflict-clause     ::=  ON CONFLICT conflict-algorithm
conflict-algorithm  ::=  ROLLBACK | ABORT | FAIL | IGNORE | REPLACE
default-value       ::=  NULL | string | number | CURRENT_TIME | CURRENT_DATE | CURRENT_TIMESTAMP
sort-order          ::=  ASC | DESC
collation-name      ::=  BINARY | NOCASE
column-list         ::=  column-name [, column-name]*
```

各列定義は、列名とその列のデータ型、およびオプションで 1 つ以上の列制約を並べたものです。列のデータ型は、その列に格納できるデータを制限します。異なるデータ型の列に値を格納しようとすると、可能な場合にはランタイムによって値が適切な型に変換され、可能でない場合にはエラーが発生します。詳しくは、「データ型のサポート」を参照してください。

NOT NULL 列制約は、その列に NULL 値を格納できないことを示します。

UNIQUE 制約を指定すると、指定された 1 つまたは複数の列にインデックスが作成されます。このインデックスには、一意キーが含まれている必要があります。つまり、任意の行と任意の別の行の間で、指定された列の値、または指定された複数列の値の組み合わせが必ず異なっていなくてはなりません。CREATE TABLE ステートメントには複数の UNIQUE 制約を含めることができます。列定義内の複数の列に UNIQUE 制約を指定できるほか、複数テーブルレベルで UNIQUE 制約を設定することもできます。

CHECK 制約では、行のデータを挿入または更新する際の条件となる式を定義します。この式が true に評価された場合のみ、行のデータが挿入または更新されます。CHECK 制約の式の結果はブール値として解決される必要があります。

列定義内の COLLATE 句は、列のテキストエントリを比較するときに使用するテキスト照合関数を指定します。デフォルトでは、BINARY 照合関数が使用されます。COLLATE 句と照合関数について詳しくは、「COLLATE」を参照してください。

DEFAULT 制約では、INSERT の実行時に使用するデフォルト値を指定します。値は、NULL、ストリング定数、または数値にすることができます。また、状況に依存しない特殊なキーワード CURRENT_TIME、CURRENT_DATE、CURRENT_TIMESTAMP のいずれかを指定することもできます。値が NULL、ストリング定数、または数値の場合は、INSERT ステートメントで列の値が指定されていない場合に、これらがそのまま列に挿入されます。値が CURRENT_TIME、CURRENT_DATE、または CURRENT_TIMESTAMP の場合は、現在の UTC 日付と時刻の一方または両方が列に挿入されます。CURRENT_TIME の書式は HH:MM:SS です。CURRENT_DATE の書式は YYYY-MM-DD です。CURRENT_TIMESTAMP の書式は YYYY-MM-DD HH:MM:SS です。

PRIMARY KEY を指定すると、通常は対応する 1 つまたは複数の列に UNIQUE インデックスが作成されます。ただし、データ型が INTEGER（または int などのシノニムのいずれか）である単一の列に PRIMARY KEY 制約を指定すると、データベースではその列がテーブルの実際の主キーとして使用されます。これは、その列に一意の整数値しか格納できないことを意味します多くの SQLite 実装では、内部主キーとして使用するために指定できる列の型は、INTEGER だけです。ただし、Adobe AIR では、INTEGER のシノニム（int など）は同様にこの動作を指定できます。

テーブルに INTEGER PRIMARY KEY 列がない場合は、行が挿入されたときに整数キーが自動的に生成されます。行の主キーには、ROWID、OID、_ROWID_ のいずれかの特殊名を使用していつでもアクセスできます。これらの名前は、INTEGER PRIMARY KEY で明示的に宣言されているか、内部的に生成された値であるかに関係なく使用できます。ただし、テーブルに明示的な INTEGER PRIMARY KEY がない場合、結果データの列の名前は、特殊名ではなく、実際の列名になります。

INTEGER PRIMARY KEY 列には、キーワード AUTOINCREMENT も指定できます。AUTOINCREMENT キーワードを使用する場合、データベースは、列に明示的な値を指定しない INSERT ステートメントの実行時に、順番にインクリメントされる整数キーを自動生成して INTEGER PRIMARY KEY 列内に挿入します。

CREATE TABLE ステートメントには PRIMARY KEY 制約を 1 つだけ含めることができます。これは、ある列の定義の一部または単一のテーブルレベル PRIMARY KEY 制約にすることができます。主キー列は暗黙的に NOT NULL になります。

いくつかの制約の後にオプションの conflict-clause を続けると、その制約に対するデフォルトの制約競合解決アルゴリズムを指定できます。デフォルトは ABORT です。同じテーブル内の異なる制約に対して、異なるデフォルト競合解決アルゴリズムを指定することも可能です。INSERT ステートメントまたは UPDATE ステートメントで異なる競合解決アルゴリズムが指定された場合は、CREATE TABLE ステートメントで指定されたアルゴリズムの代わりにそのアルゴリズムが使用されます。詳しくは、344 ページの「特殊な文と句」の「ON CONFLICT」を参照してください。

FOREIGN KEY 制約などの追加の制約を指定してもエラーにはなりませんが、実行時に無視されます。

TEMP または TEMPORARY キーワードが CREATE と TABLE の間にある場合、作成されるテーブルは、同じデータベース接続（SQLConnection インスタンス）内でのみ参照できます。データベース接続が終了すると、自動的に削除されます。一時テーブルでは、作成されたインデックスも一時的です。一時テーブルと一時インデックスは、メインのデータベースファイルとは別のファイルに格納されます。

オプションの database-name 接頭辞を指定すると、指定されたデータベース（データベース名を指定して attach() メソッドを呼び出すことによって SQLConnection インスタンスに接続されたデータベース）内にテーブルが作成されます。database-name 接頭辞と TEMP キーワードを両方とも指定すると、（database-name 接頭辞が temp である場合を除き）エラーになります。データベース名が指定されておらず、TEMP キーワードが存在しない場合は、メインデータベース（open() または openAsync() メソッドを使用して SQLConnection インスタンスに接続されたデータベース）にテーブルが作成されます。

列数またはテーブル内の制約数に恣意的な制限はありません。1 行のデータ量にも恣意的な制限はありません。

CREATE TABLE AS 形式は、クエリの結果セットでテーブルを定義します。結果に含まれる列の名前がテーブル列の名前になります。

オプションの IF NOT EXISTS 句があり、同じ名前の別のテーブルが既に存在する場合、データベースは CREATE TABLE コマンドを無視します。

テーブルは DROP TABLE ステートメントを使用して削除できます。また、ALTER TABLE ステートメントを使用して、限られた範囲で変更を加えることができます。

ALTER TABLE

ALTER TABLE コマンドを使用すると、テーブルの名前を変更したり、既存のテーブルに新しい列を追加したりできます。テーブルから列を削除することはできません。

```
sql-statement ::= ALTER TABLE [database-name.] table-name alteration
alteration    ::= RENAME TO new-table-name
alteration    ::= ADD [COLUMN] column-def
```

RENAME TO 構文は、[database-name.] table-name で指定されたテーブルの名前を new-table-name に変更します。このコマンドはあくまでも同じデータベース内でテーブル名を変更するためのもので、アタッチされたデータベース間でテーブルを移動することはできません。

名前変更されるテーブルにトリガーまたはインデックスがある場合、それらはテーブル名が変更された後もテーブルにアタッチされたままになります。ただし、名前変更されるテーブルを参照するトリガーによって実行されたビュー定義またはステートメントがある場合は、新しいテーブル名を使用するように自動的に変更されることはありません。名前変更されたテーブルにビューまたはトリガーが関連付けられている場合は、新しいテーブル名を使用してトリガーまたはビュー定義を手動で削除し、再作成する必要があります。

ADD [COLUMN] 構文は、既存のテーブルに新しい列を追加するときに使用します。新しい列は常に、既存の列リストの最後に追加されます。column-def 句では CREATE TABLE ステートメントで許可されるすべての形式を使用できますが、次のような制限があります。

- 列に PRIMARY KEY 制約または UNIQUE 制約を設定することはできません。
- 列のデフォルト値を CURRENT_TIME、CURRENT_DATE、または CURRENT_TIMESTAMP にすることはできません。
- NOT NULL 制約を指定する場合は、列のデフォルト値を NULL 以外にする必要があります。

ALTER TABLE 文の実行時間は、テーブルに含まれるデータ量には関係ありません。

DROP TABLE

DROP TABLE ステートメントは、CREATE TABLE ステートメントで追加されたテーブルを削除します。削除するテーブルの名前を table-name で指定します。指定したテーブルは、データベースおよびディスクファイルから完全に削除されます。テーブルを復元することはできません。テーブルに関連付けられたインデックスもすべて削除されます。

```
sql-statement  ::=  DROP TABLE [IF EXISTS] [database-name.] table-name
```

デフォルトでは、DROP TABLE 文を実行してもデータベースファイルのサイズは小さくなりません。データベース内に空の領域が保持され、後続の INSERT 操作でその領域が使用されます。データベース内の空き領域を削除するには、SQLConnection.clean() メソッドを使用します。データベースが最初に作成されたときに autoClean パラメーターが true に設定された場合は、領域が自動的に解放されます。

オプションの IF EXISTS 句を指定すると、テーブルが存在しない場合に通常発生するエラーが抑制されます。

CREATE INDEX

CREATE INDEX コマンドでは、キーワード CREATE INDEX の後に新しいインデックスの名前、キーワード ON およびインデックスの作成対象となる作成済みのテーブルの名前を指定し、その後にインデックスキーに使用するテーブル内の列名のリストを括弧で囲んで記述します。

```
sql-statement  ::=  CREATE [UNIQUE] INDEX [IF NOT EXISTS] [database-name.] index-name
                    ON table-name ( column-name [, column-name]* )
column-name    ::=  name [COLLATE collation-name] [ASC | DESC]
```

各列名の後にはソート順序を指定する ASC または DESC キーワードを続けることができますが、ソート順序の指定はランタイムでは無視されます。ソートは常に昇順で行われます。

各列の後の COLLATE 句は、その列のテキスト値で使用する照合順序を定義します。デフォルトの照合順序は、CREATE TABLE 文でその列に定義された照合順序です。照合順序が指定されていない場合は、BINARY 照合順序が使用されます。COLLATE 句と照合関数の定義については、「COLLATE」を参照してください。

1 つのテーブルにアタッチできるインデックス数に恣意的な制限はありません。インデックス内の列数にも制限はありません。

DROP INDEX

DROP INDEX 文は、CREATE INDEX 文で追加されたインデックスを削除します。指定したインデックスは、データベースファイルから完全に削除されます。インデックスを復元する唯一の方法は、適切な CREATE INDEX コマンドを再度発行することです。

```
sql-statement ::= DROP INDEX [IF EXISTS] [database-name.] index-name
```

デフォルトでは、DROP INDEX 文を実行してもデータベースファイルのサイズは小さくなりません。データベース内に空の領域が保持され、後続の INSERT 操作でその領域が使用されます。データベース内の空き領域を削除するには、SQLConnection.clean() メソッドを使用します。データベースが最初に作成されたときに autoClean パラメーターが true に設定された場合は、領域が自動的に解放されます。

CREATE VIEW

CREATE VIEW コマンドは、あらかじめ定義された SELECT ステートメントに名前を割り当てます。この新しく割り当てた名前は、別の SELECT ステートメントの FROM 句でテーブル名の代わりに使用できます。通常、ビューは、複雑な（および最近使用された）データのセットを他の操作で使用できる構造に結合することで、クエリーを単純化するために使用されます。

```
sql-statement ::= CREATE [TEMP | TEMPORARY] VIEW [IF NOT EXISTS] [database-name.] view-name AS select-statement
```

CREATE と VIEW の間に TEMP または TEMPORARY キーワードを指定すると、作成されたビューはデータベースを開いた SQLConnection インスタンスからのみ見えるようになり、データベースが閉じると自動的に削除されます。

[database-name] を指定すると、指定されたデータベース（name 引数を指定した attach() メソッドを使用して SQLConnection インスタンスに接続されたデータベース）内にビューが作成されます。[database-name] と TEMP キーワードを両方とも指定すると、（[database-name] が temp である場合を除き）エラーになります。データベース名が指定されておらず、TEMP キーワードが存在しない場合は、メインデータベース（open() または openAsync() メソッドを使用して SQLConnection インスタンスに接続されたデータベース）にビューが作成されます。

ビューは読み取り専用です。対応するタイプのトリガー（INSTEAD OF DELETE、INSTEAD OF INSERT、INSTEAD OF UPDATE）が少なくとも 1 つ定義されていない限り、ビューに対して DELETE ステートメント、INSERT ステートメント、または UPDATE ステートメントを使用することはできません。ビューに対するトリガーの作成について詳しくは、「CREATE TRIGGER」を参照してください。

ビューをデータベースから削除するには、DROP VIEW 文を使用します。

DROP VIEW

DROP VIEW ステートメントは、CREATE VIEW ステートメントで作成されたビューを削除します。

```
sql-statement ::= DROP VIEW [IF EXISTS] view-name
```

削除するビューの名前を view-name で指定します。ビューはデータベースから削除されますが、基になるテーブルのデータは変更されません。

CREATE TRIGGER

CREATE TRIGGER 文は、データベーススキーマにトリガーを追加するときに使用します。トリガーとは、指定されたデータベースイベント（database-event）が発生したときに自動的に実行されるデータベース操作（trigger-action）のことです。

```
sql-statement   ::=  CREATE [TEMP | TEMPORARY] TRIGGER [IF NOT EXISTS] [database-name.] trigger-name
                     [BEFORE | AFTER] database-event
                     ON table-name
                     trigger-action
sql-statement   ::=  CREATE [TEMP | TEMPORARY] TRIGGER [IF NOT EXISTS] [database-name.] trigger-name
                     INSTEAD OF database-event
                     ON view-name
                     trigger-action
database-event  ::=  DELETE |
                     INSERT |
                     UPDATE |
                     UPDATE OF column-list
trigger-action  ::=  [FOR EACH ROW] [WHEN expr]
                     BEGIN
                       trigger-step ;
                       [ trigger-step ; ]*
                     END
trigger-step    ::=  update-statement |
                     insert-statement |
                     delete-statement |
                     select-statement
column-list     ::=  column-name [, column-name]*
```

トリガーは、特定のデータベーステーブルに対して DELETE、INSERT、または UPDATE が発生したときや、テーブルの指定された 1 つ以上の列に対して UPDATE が行われたときにその都度実行されるよう指定できます。トリガーは、TEMP または TEMPORARY キーワードが使用されていない限り永続的です。その場合、トリガーは、SQLConnection インスタンスのメインデータベース接続が終了すると削除されます。タイミング（BEFORE または AFTER）が指定されていない場合、デフォルトのタイミングは BEFORE になります。

FOR EACH ROW トリガーのみがサポートされているので、FOR EACH ROW キーワードはオプションです。FOR EACH ROW トリガーでは、WHEN 句の式が true に評価された場合、トリガーの起動元のステートメントによって挿入、更新または削除された各データベース行に対して、trigger-step ステートメントが実行されます。

WHEN 句を指定すると、WHEN 句が true に評価された行に対してのみ、trigger-steps として指定した SQL 文が実行されます。WHEN 句を指定しなければ、すべての行に対して SQL 文が実行されます。

トリガーの本文内で（trigger-action 句）、特別なテーブル名 OLD および NEW を使用して、影響を受けるテーブルの変更前および変更後の値を使用できます。OLD テーブルと NEW テーブルの構造は、トリガーを作成するテーブルの構造と一致します。OLD テーブルには、トリガーを発生させた文によって変更または削除された行の、操作が行われる前の状態が含まれます。NEW テーブルには、トリガーを発生させた文によって変更または作成された行の、操作が行われた後の状態が含まれます。WHEN 句と trigger-step ステートメントのどちらからも、NEW.column-name および OLD.column-name（column-name はトリガーが関連付けられているテーブルの列の名前）の形式の参照を使用して、挿入、削除、または更新される（された）行の値にアクセスできます。OLD テーブルと NEW テーブルの参照を使用できるかどうかは、トリガーが処理する database-event の種類によって決まります。

- INSERT – NEW 参照が有効です。
- UPDATE – NEW および OLD 参照が有効です。
- DELETE – OLD 参照が有効です。

タイミングの指定（BEFORE、AFTER、または INSTEAD OF）は、trigger-step ステートメントが、関連する行の挿入時、変更時、または削除時を基準としていつ実行されるかを決定します。trigger-step 内の UPDATE ステートメントまたは INSERT ステートメントの一部として ON CONFLICT 句を指定することもできますが、トリガーを発生させた文の一部として ON CONFLICT 句が指定されている場合は、代わりにその競合処理ポリシーが使用されます。

テーブルのトリガーに加えて、INSTEAD OF トリガーをビューに対して作成できます。1 つ以上の INSTEAD OF INSERT、INSTEAD OF DELETE、または INSTEAD OF UPDATE トリガーがビューに定義されている場合は、対応するタイプのステートメント（INSERT、DELETE、または UPDATE）をビューで実行してもエラーとは見なされません。

その場合は、INSERT、DELETE、または UPDATE をビューで実行すると、対応するトリガーが発生します。トリガーは INSTEAD OF トリガーなので、ビューの基礎となるテーブルは、トリガーが起動する原因となったステートメントによって変更されません。ただし、トリガーは、基礎となるテーブルに対して変更操作を実行するために使用できます。

INTEGER PRIMARY KEY 列を持つテーブルにトリガーを作成するときは、注意すべき重要な問題があります。トリガーを起動したステートメントによって更新される行の INTEGER PRIMARY KEY 列が BEFORE トリガーによって変更された場合は、更新が行われません。この問題を回避するには、INTEGER PRIMARY KEY 列の代わりに PRIMARY KEY 列を持つテーブルを作成します。

トリガーは DROP TRIGGER 文を使用して削除できます。テーブルまたはビューが削除されると、そのテーブルまたはビューに関連付けられたトリガーもすべて自動的に削除されます。

RAISE() 関数

特殊な SQL 関数である RAISE() は、トリガーの trigger-step ステートメントの中で使用できます。この関数の構文は次のとおりです。

```
raise-function  ::=  RAISE ( ABORT, error-message ) |
                     RAISE ( FAIL, error-message ) |
                     RAISE ( ROLLBACK, error-message ) |
                     RAISE ( IGNORE )
```

最初の 3 つの形式の 1 つがトリガーの実行中に呼び出されると、指定された ON CONFLICT 処理アクション（ABORT、FAIL、または ROLLBACK）が実行され、現在のステートメントの実行が終了します。ROLLBACK はステートメントの実行エラーと見なされるため、execute() メソッドが実行された SQLStatement インスタンスによって error（SQLErrorEvent.ERROR）イベントが送出されます。送出されたイベントオブジェクトの error プロパティ内にある SQLError オブジェクトの details プロパティは、RAISE() 関数で指定された error-message に設定されています。

RAISE(IGNORE) を呼び出すと、現在のトリガーの残りの部分、トリガーを発生させた文、および通常であれば引き続き実行されるはずの別のトリガーがすべて中止されます。データベースの変更はロールバックされません。トリガーを発生させた文自体がトリガーの一部である場合、そのトリガープログラムは次のステップの最初から再開されます。競合解決アルゴリズムについて詳しくは、「ON CONFLICT（競合アルゴリズム）」を参照してください。

DROP TRIGGER

DROP TRIGGER 文は、CREATE TRIGGER 文で作成されたトリガーを削除します。

```
sql-statement  ::=  DROP TRIGGER [IF EXISTS] [database-name.] trigger-name
```

トリガーはデータベースから削除されます。関連付けられたテーブルが削除されたときにも、トリガーは自動的に削除されます。

特殊な文と句

この節では、SQL の拡張機能としてランタイムによって提供されるいくつかの句と、多くの文で使用できる 2 つの言語要素（コメントと式）について説明します。

COLLATE

COLLATE 句は、値の比較またはソート時に使用する比較アルゴリズムを指定するために、SELECT ステートメント、CREATE TABLE ステートメントおよび CREATE INDEX ステートメントで使用します。

```
sql-statement   ::=  COLLATE collation-name
collation-name  ::=  BINARY | NOCASE
```

列のデフォルトの照合タイプは BINARY です。TEXT 格納クラスの値で BINARY 照合を使用すると、テキストがどのようにしてエンコードされているかにかかわらず、値を表すメモリ内のバイトを比較することによってバイナリ照合が行われます。

NOCASE 照合シーケンスは、TEXT 格納クラスの値に対してのみ適用されます。NOCASE 照合を使用すると、大文字と小文字を区別せずに比較されます。

NULL、BLOB、INTEGER、REAL 型の格納クラスでは、照合シーケンスは使用されません。

BINARY 以外の照合タイプを列で使用するには、CREATE TABLE ステートメントの列定義の一部として COLLATE 句を指定する必要があります。2 つの TEXT 値を比較するときは常に、照合順序を使用し、次の規則に従って比較結果が決定されます。

- 二項比較演算子では、一方のオペランドが列である場合、その列のデフォルトの照合タイプによって、比較で使用される照合シーケンスが決まります。両方のオペランドが列である場合は、左のオペランドの照合タイプによって、使用される照合順序が決まります。どちらのオペランドも列でない場合は、BINARY 照合順序が使用されます。
- BETWEEN...AND 演算子は、2 つの式に対して >= 演算子と <= 演算子を使用するのと同じです。例えば、x BETWEEN y AND z という式は x >= y AND x <= z と同じです。したがって、BETWEEN...AND 演算子も上記の規則に従って照合シーケンスが決定されます。
- IN 演算子は、使用する照合シーケンスを決定するという点では、= 演算子と同じように動作します。例えば、x IN (y, z) という式で使用される照合シーケンスは、x が列の場合は x のデフォルトの照合タイプになります。それ以外の場合は、BINARY 照合が使用されます。
- SELECT ステートメントでは、ORDER BY 句の中で、ソート操作で使用する照合シーケンスを明示的に指定できます。その場合は、明示的に指定した照合順序が常に使用されます。それ以外の場合は、ORDER BY 句によってソートされる式が列の場合には、その列のデフォルトの照合タイプを使用してソート順序が決定されます。式が列でない場合は、BINARY 照合順序が使用されます。

EXPLAIN

EXPLAIN コマンド修飾子は、SQL の非標準の拡張機能です。

```
sql-statement  ::=  EXPLAIN sql-statement
```

SQL ステートメントの前に EXPLAIN キーワードを付けると、コマンドを実際に実行する代わりに、EXPLAIN キーワードを指定せずにコマンドを実行した場合に使用される一連の仮想マシン命令が報告されます。この高度な EXPLAIN 機能を使用すると、SQL ステートメントのテキストを調整してパフォーマンスを最適化したり、適切に機能しない SQL ステートメントをデバッグしたりできます。

ON CONFLICT（競合アルゴリズム）

ON CONFLICT 句は独立した SQL コマンドではありません。これは他の多くの SQL コマンドで使用できる非標準の句です。

```
conflict-clause     ::=  ON CONFLICT conflict-algorithm
conflict-clause     ::=  OR conflict-algorithm
conflict-algorithm  ::=  ROLLBACK |
                         ABORT |
                         FAIL |
                         IGNORE |
                         REPLACE
```

最初の形式の、キーワード ON CONFLICT を使用する ON CONFLICT 句は、CREATE TABLE ステートメントで使用します。INSERT ステートメントまたは UPDATE ステートメントでは、構ステートメントがより自然に見えるように、ON CONFLICT が OR に置き換えられた 2 番目の形式を使用します。例えば、INSERT ON CONFLICT IGNORE は INSERT OR IGNORE になります。キーワードは異なりますが、句の意味はどちらの形式でも同じです。

ON CONFLICT 句は、制約競合の解決に使用するアルゴリズムを指定します。指定できるアルゴリズムは、ROLLBACK、ABORT、FAIL、IGNORE、REPLACE の 5 種類です。デフォルトのアルゴリズムは ABORT です。これら 5 種類の競合アルゴリズムについて以下に説明します。

ROLLBACK 制約違反が発生すると、直ちに ROLLBACK が実行され、現在のトランザクションが終了します。コマンドは中止され、SQLStatement インスタンスによって error イベントが送出されます。トランザクションがアクティブでない場合（すべてのコマンドについて作成される暗黙的なトランザクションを除く）、このアルゴリズムは ABORT と同じです。

ABORT 制約違反が発生すると、そのコマンドによる変更がすべて取り消され、SQLStatement インスタンスによって error イベントが送出されます。ROLLBACK は実行されないので、トランザクション内の前のコマンドで行われた変更はそのまま残ります。ABORT はデフォルトの動作です。

FAIL 制約違反が発生すると、コマンドが中止され、SQLStatement によって error イベントが送出されます。ただし、制約違反が発生する前にそのステートメントによってデータベースに加えられた変更は取り消されずにそのまま残ります。例えば、UPDATE ステートメントの実行時に更新対象の 100 行目で制約違反が発生した場合、それまでの 99 行の変更はそのまま残りますが、100 行目以降の変更は行われません。

IGNORE 制約違反が発生したとき、制約違反を含む 1 行は挿入も変更もされません。行が無視されるという点を除き、コマンドは通常どおり続行されます。制約違反が発生した行の前後の行は通常どおり挿入または更新されます。エラーは返されません。

REPLACE UNIQUE 制約違反が発生すると、制約違反の原因となる既存の行が削除された後で、現在の行が挿入または更新されます。したがって、挿入または更新は常に実行され、コマンドは通常どおり続行されます。エラーは返されません。NOT NULL 制約違反が発生した場合は、NULL 値がその列のデフォルト値で置き換えられます。列にデフォルト値がない場合は、ABORT アルゴリズムが使用されます。CHECK 制約違反が発生した場合は、IGNORE アルゴリズムが使用されます。この競合解決方法により、制約を満たすために行が削除されたときは、それらの行に対する削除トリガーは呼び出されません。

INSERT ステートメントまたは UPDATE ステートメントの OR 句で指定されたアルゴリズムは、CREATE TABLE ステートメントで指定されたアルゴリズムよりも優先されます。CREATE TABLE ステートメントおよび実行中の INSERT ステートメントまたは UPDATE ステートメントのどちらでもアルゴリズムが指定されていない場合は、ABORT アルゴリズムが使用されます。

REINDEX

REINDEX コマンドは、1 つ以上のインデックスを削除して再作成するときに使用します。このコマンドは、照合シーケンスの定義が変更されたときに便利です。

```
sql-statement  ::=  REINDEX collation-name
sql-statement  ::=  REINDEX [database-name .] ( table-name | index-name )
```

最初の形式では、アタッチされたデータベース内の指定された照合順序を使用するすべてのインデックスが再作成されます。2 番目の形式では、table-name を指定した場合、そのテーブルに関連付けられたすべてのインデックスが再構築されます。index-name を指定した場合は、指定したインデックスのみが削除され、再作成されます。

コメント

コメントは SQL コマンドではありませんが、SQL クエリの中に記述できます。コメントはランタイムでは空白として扱われます。コメントは、空白が許可される任意の場所（複数行にわたる式の内部など）に記述できます。

```
comment             ::=  single-line-comment |
                         block-comment
single-line-comment ::=  -- single-line
block-comment       ::=  /* multiple-lines or block [*/]
```

単一行のコメントには 2 つのダッシュを使用します。単一行コメントでは、その行の末尾までがコメントになります。

ブロックコメントは、任意の行数にわたることも、単一行に埋め込むこともできます。コメントの終わりを示す区切り記号がない場合は、入力の最後までがコメントになります。これはエラーとして扱われません。新しい SQL 文は、ブロックコメントが終了した後の行から開始できます。ブロックコメントは、空白が許可される任意の場所（式の内部など）のほか、他の SQL 文の中にも埋め込むことができます。ブロックコメントはネストされません。ブロックコメントの中に単一行コメントを記述しても無視されます。

式

式は他の SQL ブロック内のサブコマンドです。SQL 文で使用する式の有効な構文を次に示します。

```
expr             ::=  expr binary-op expr |
                      expr [NOT] like-op expr [ESCAPE expr] |
                      unary-op expr |
                      ( expr ) |
                      column-name |
                      table-name.column-name |
                      database-name.table-name.column-name |
                      literal-value |
                      parameter |
                      function-name( expr-list | * ) |
                      expr ISNULL |
                      expr NOTNULL |
                      expr [NOT] BETWEEN expr AND expr |
                      expr [NOT] IN ( value-list ) |
                      expr [NOT] IN ( select-statement ) |
                      expr [NOT] IN [database-name.] table-name |
                      [EXISTS] ( select-statement ) |
                      CASE [expr] ( WHEN expr THEN expr )+ [ELSE expr] END |
                      CAST ( expr AS type ) |
                      expr COLLATE collation-name
like-op          ::=  LIKE | GLOB
binary-op        ::=  see Operators
unary-op         ::=  see Operators
parameter        ::=  :param-name | @param-name | ?
value-list       ::=  literal-value [, literal-value]*
literal-value    ::=  literal-string | literal-number | literal-boolean | literal-blob | literal-null
literal-string   ::=  'string value'
literal-number   ::=  integer | number
literal-boolean  ::=  true | false
literal-blob  ::=  X'string of hexadecimal data'
literal-null  ::=  NULL
```

式とは、単一の値に解決できる値と演算子の任意の組み合わせを指します。式はブール値（true または false）に解決されるものと、非ブール値に解決されるものの 2 種類に大別されます。

WHERE 句、HAVING 句、JOIN 句の中の ON 式、CHECK 式などで使用する式は、ブール値に解決される必要があります。この条件を満たす式の種類は次のとおりです。

- ISNULL
- NOTNULL
- IN ()
- EXISTS ()
- LIKE
- GLOB
- いくつかの関数
- いくつかの演算子（具体的には比較演算子）

リテラル値

リテラルの数値は、整数または浮動小数点数として記述します。科学的表記法もサポートされます。小数点には常に .（ピリオド）を使用します。

ストリングリテラルを示すときは、ストリングを一重引用符（'）で囲みます。ストリングの中に一重引用符を含めるには、'' のように一重引用符を行で 2 つ重ねます。

ブールリテラルの値は true または false で示します。リテラルブール値は、列のデータ型が Boolean の場合に使用します。

BLOB リテラルは 16 進数データを含むストリングリテラルで、1 つの x または X を先頭に付けます（X'53514697465' など）。

リテラル値にはトークン NULL も使用できます。

列名

列名には、CREATE TABLE ステートメントで定義されたいずれかの名前、または特殊識別子の ROWID、OID、_ROWID_ のいずれかを使用できます。これらの特殊識別子はすべて、各テーブルのすべての行に関連付けられた一意のランダムな整数キー（'' 行キー ''）を表します。特殊識別子は、CREATE TABLE 文で同じ名前を持つ実際の列が定義されていない場合のみ、行キーを参照します。行キーは読み取り専用列のように動作します。行キーは通常の列が使用できる場所であればどこでも使用できますが、行キーの値を UPDATE ステートメントまたは INSERT ステートメントで変更することはできません。SELECT * FROM table 文で返される結果セットには、行キーは含まれません。

SELECT 文

IN 演算子の右オペランド、スカラ量（単一の結果値）、または EXISTS 演算子のオペランドとして式を使用するときは、SELECT ステートメントを使用できます。スカラ量または IN 演算子のオペランドとして使用する場合、SELECT 文で返すことができるのは 1 つの列のみです。複合 SELECT ステートメント（UNION や EXCEPT などのキーワードで接続された SELECT ステートメント）も使用できます。EXISTS 演算子では、SELECT 文の結果セットに含まれる列は無視され、行が 1 行以上存在する場合は TRUE、結果セットが空の場合は FALSE が返されます。SELECT 式のどの項も外側のクエリの値を参照しない場合、式は他の処理が実行される前に 1 回だけ評価され、その結果が必要に応じて再利用されます。SELECT 式に外部クエリからの変数が含まれる場合（これを相関サブクエリと呼びます）、SELECT 文は必要になるたびに再評価されます。

SELECT ステートメントが IN 演算子の右オペランドである場合、IN 演算子は、左オペランドの結果が SELECT ステートメントの結果セットに含まれるいずれかの値と等しい場合に TRUE を返します。IN 演算子の前に NOT キーワードを付けると、テストの意味が逆になります。

IN 演算子の右オペランド以外の式で SELECT 文を使用する場合は、SELECT 文の結果の 1 行目が値として使用されます。SELECT ステートメントが複数行の結果を返す場合、2 行目以降の行はすべて無視されます。SELECT 文が行を 1 行も返さない場合、SELECT 文の値は NULL になります。

CAST 式

CAST 式は、指定された値のデータ型を別の指定されたデータ型に変更します。指定できるデータ型は、CREATE TABLE 文の列定義でデータ型として使用できる空でない型名です。詳しくは、「データ型のサポート」を参照してください。

その他の式要素

式で使用できるその他の SQL 要素は以下のとおりです。

- ビルトイン関数：集計関数、スカラ関数、および日付 / 時刻書式設定関数
- 演算子
- パラメーター

ビルトイン関数

ビルトイン関数は次の 3 種類に大別されます。

- 集計関数
- スカラ関数
- 日時関数

これらの関数に加えて、特殊関数の RAISE() もあります。これは、トリガーの実行時にエラー通知を提供するために使用します。この関数は、CREATE TRIGGER 文の本体でのみ使用できます。RAISE() 関数について詳しくは、「CREATE TRIGGER」の「RAISE()」を参照してください。

SQL のすべてのキーワードと同様に、関数名も大文字と小文字は区別されません。

集計関数

集計関数は、複数行の値に対して演算を実行します。これらは主に SELECT ステートメントの中で GROUP BY 句と一緒に使用します。

AVG(X)	グループ内の NULL でないすべての X の平均値を返します。数値に見えないストリング値および BLOB 値は 0 と見なされます。AVG() の結果は、入力がすべて整数であっても、常に浮動小数点値です。
COUNT(X) COUNT(*)	最初の形式は、グループ内で X が NULL でない回数を返します。* 引数の付いた 2 番目の形式は、グループ内の行の総数を返します。
MAX(X)	グループ内のすべての値の最大値を返します。通常のソート順序を使用して最大値が決定されます。
MIN(X)	グループ内のすべての値のうち、NULL でない最小の値を返します。通常のソート順序を使用して最小値が決定されます。グループ内の値がすべて NULL の場合は、NULL が返されます。
SUM(X)	グループ内の NULL でないすべての値の合計を返します。値がすべて NULL の場合には、SUM() は NULL を返し、TOTAL() は 0.0 を返します。TOTAL() の結果は常に浮動小数点値です。SUM() の結果は、NULL 以外の入力がすべて整数の場合は整数値になります。SUM() への入力に整数でも NULL でもない値が含まれている場合、SUM() は浮動小数点値を返しますが、この値は正確な合計の近似値である場合があります。
TOTAL(X)	

上記の集計関数のうち引数を 1 つ受け取るものはいずれも、その引数の前にキーワード DISTINCT を付けることができます。その場合は、重複した要素が除外されてから集計関数に渡されます。例えば、関数が COUNT(DISTINCT x) を呼び出すと、列 x の NULL でない値の総数ではなく、列 X の互いに異なる値の数が返されます。

スカラ関数

スカラ関数は一度に 1 つの行の値に作用します。

ABS(X)	引数 X の絶対値を返します。
COALESCE(X, Y, ...)	NULL でない最初の引数のコピーを返します。引数がすべて NULL の場合は、NULL が返されます。引数は 2 つ以上指定する必要があります。
GLOB(X, Y)	この関数は、X GLOB Y 構文を実装するために使用されます。
IFNULL(X, Y)	NULL でない最初の引数のコピーを返します。引数がどちらも NULL の場合は、NULL が返されます。この関数の動作は COALESCE() と同じです。
HEX(X)	引数は BLOB 格納型の値と解釈されます。その値の内容の 16 進数表現が返されます。
LAST_INSERT_ROWID()	現在の SQLConnection を通じてデータベースに最後に挿入された行の行識別子（生成された主キー）を返します。この値は、SQLConnection.lastInsertRowID プロパティによって返される値と同じです。
LENGTH(X)	X のストリングの長さ（文字数）を返します。
LIKE(X, Y [, Z])	この関数は、SQL の X LIKE Y [ESCAPE Z] 構文を実装するために使用されます。オプションの ESCAPE 句を指定すると、3 つの引数を使用して関数が呼び出されます。それ以外の場合は、2 つの引数のみを使用して関数が呼び出されます。
LOWER(X)	ストリング X の、すべての文字が小文字に変換されたコピーを返します。
LTRIM(X) LTRIM(X, Y)	X の左側からスペースを除去したストリングを返します。Y 引数を指定すると、Y に含まれるいずれかの文字が X の左側から除去されます。
MAX(X, Y, ...)	引数の中の最大値を返します。引数には、数字だけでなくストリングも指定できます。最大値は、定義されたソート順序に従って決定されます。MAX() は、引数が 2 個以上ある場合は単純な関数ですが、引数が 1 つだけの場合は集計関数になります。

MIN(X, Y, ...)	引数の中の最小値を返します。引数には、数字だけでなくストリングも指定できます。最小値は、定義されたソート順序に従って決定されます。MIN() は、引数が 2 個以上ある場合は単純な関数ですが、引数が 1 つだけの場合は集計関数になります。
NULLIF(X, Y)	引数が異なる場合は最初の引数を返し、それ以外の場合は NULL を返します。
QUOTE(X)	このルーチンは、引数の値を別の SQL 文に挿入するのに適した形に変更したストリングを返します。ストリングは一重引用符で囲まれ、必要に応じて内部引用符がエスケープされます。BLOB 格納クラスは 16 進数のリテラルとしてエンコードされます。この関数は、元に戻す / 繰り返し機能を実装するためのトリガーを記述する場合に役立ちます。
RANDOM(*)	-9223372036854775808 から 9223372036854775807 までの疑似乱数（整数）を返します。これは暗号的に強い乱数ではありません。
RANDOMBLOB(N)	疑似ランダムバイトを含む N バイトの BLOB を返します。N は正の整数である必要があります。これは暗号的に強いランダム値ではありません。N の値が負の場合は、単一のバイトが返されます。
ROUND(X) ROUND(X, Y)	数値 X を小数第 Y 位に四捨五入します。Y 引数を省略すると、0 が使用されます。
RTRIM(X) RTRIM(X, Y)	X の右側からスペースを除去したストリングを返します。Y 引数を指定すると、Y に含まれるいずれかの文字が X の右側から除去されます。
SUBSTR(X, Y, Z)	入力ストリング X の Y 番目の文字から Z 個の文字を取り出したサブストリングを返します。X の一番左の文字のインデックス位置は 1 です。Y が負の場合は、サブストリングの開始位置が左からではなく右からカウントされます。
TRIM(X) TRIM(X, Y)	X の右側からスペースを除去したストリングを返します。Y 引数を指定すると、Y に含まれるいずれかの文字が X の右側から除去されます。
TYPEOF(X)	式 X の型を返します。有効な戻り値は、「null」、「integer」、「real」、「text」および「blob」です。データ型について詳しくは、「データ型のサポート」を参照してください。
UPPER(X)	ストリング X の、すべての文字が大文字に変換されたコピーを返します。
ZEROBLOB(N)	N バイトの 0x00 を含む BLOB を返します。

日付 / 時刻書式設定関数

日付 / 時刻書式設定関数は、書式設定された日付および時刻データの作成に使用するスカラ関数のグループです。これらはストリング値および数値を引数とし、ストリング値および数値を返します。DATE データ型に対して使用するものではありません。DATE データ型で宣言された列のデータを使用する場合、これらの関数は期待どおりに動作しません。

DATE(T, ...)	DATE() 関数は、YYYY-MM-DD 形式の日付を含むストリングを返します。最初のパラメーター（T）では、「時刻書式」で示された書式の時刻ストリングを指定します。時刻ストリングの後に任意の数の修飾子を指定できます。修飾子については、「修飾子」を参照してください。
TIME(T, ...)	TIME() 関数は、HH:MM:SS 形式の時刻を含むストリングを返します。最初のパラメーター（T）では、「時刻書式」で示された書式の時刻ストリングを指定します。時刻ストリングの後に任意の数の修飾子を指定できます。修飾子については、「修飾子」を参照してください。

DATETIME(T, ...)	DATETIME() 関数は、YYYY-MM-DD HH:MM:SS 形式の日時を含むストリングを返します。最初のパラメーター（T）では、「時刻書式」で示された書式の時刻ストリングを指定します。時刻ストリングの後に任意の数の修飾子を指定できます。修飾子については、「修飾子」を参照してください。
JULIANDAY(T, ...)	JULIANDAY() 関数は、グリニッジでの紀元前 4714 年 11 月 24 日の正午から指定された日付までの日数を示す数字を返します。最初のパラメーター（T）では、「時刻書式」で示された書式の時刻ストリングを指定します。時刻ストリングの後に任意の数の修飾子を指定できます。修飾子については、「修飾子」を参照してください。
STRFTIME(F, T, ...)	STRFTIME() ルーチンは、最初の引数 F で指定された書式ストリングに従って書式設定された日付を返します。書式ストリングでは、次の置換がサポートされます。 %d - 月の日にち %f - 秒の端数（SS.SSS） %H - 時間（00 ～ 24） %j - 1 月 1 日からの日数（001 ～ 366） %J - ユリウス日 %m - 月（01 ～ 12） %M - 分（00 ～ 59） %s - 1970 年 1 月 1 日からの秒数 %S - 秒（00 ～ 59） %w - 曜日（0 ～ 6、日曜日 = 0） %W - 1 月 1 日からの週数（00 ～ 53） %Y - 年（0000 ～ 9999） %% - % 2 番目のパラメーター（T）では、「時刻書式」で示された書式の時刻ストリングを指定します。時刻ストリングの後に任意の数の修飾子を指定できます。修飾子については、「修飾子」を参照してください。

時刻書式

時刻ストリングには、次のいずれかの書式を使用できます。

YYYY-MM-DD	2007-06-15
YYYY-MM-DD HH:MM	2007-06-15 07:30
YYYY-MM-DD HH:MM:SS	2007-06-15 07:30:59
YYYY-MM-DD HH:MM:SS.SSS	2007-06-15 07:30:59.152
YYYY-MM-DDTHH:MM	2007-06-15T07:30
YYYY-MM-DDTHH:MM:SS	2007-06-15T07:30:59
YYYY-MM-DDTHH:MM:SS.SSS	2007-06-15T07:30:59.152
HH:MM	07:30（日付は 2000-01-01）
HH:MM:SS	07:30:59（日付は 2000-01-01）
HH:MM:SS.SSS	07:30:59:152（日付は 2000-01-01）
now	現在の日時（世界標準時）
DDDD.DDDD	ユリウス日（浮動小数点数）

上記の書式に含まれる T は、日付と時間を区切るリテラル文字の "T" です。時刻のみを含む書式では、日付は 2001-01-01 と見なされます。

修飾子

時刻ストリングの後に修飾子を 0 個以上続けることで、日付または日付の解釈を変更できます。使用できる修飾子は次のとおりです。

NNN days	時刻に加算する日数です。
NNN hours	時刻に加算する時間数です。
NNN minutes	時刻に加算する分数です。
NNN.NNNN seconds	時刻に加算する秒数およびミリ秒数です。

NNN months	時刻に加算する月数です。
NNN years	時刻に加算する年数です。
start of month	時刻を月の初めに戻します。
start of year	時刻を年の初めに戻します。
start of day	時刻を日の初めに戻します。
weekday N	時刻を指定した曜日に進めます（0 = 日曜日、1 = 月曜日など）。
localtime	日付をローカル時間に変換します。
utc	日付を世界標準時に変換します。

演算子

SQL は様々な演算子をサポートしており、ほとんどのプログラミング言語に存在する一般的な演算子のほか、SQL に固有の演算子もあります。

一般的な演算子

次のリストは、SQL ブロックで使用できる二項演算子を優先順位の高い順に並べたものです。

```
*    /    %
+    -
<< >> & |
< >=   > >=
=    ==   !=   <> IN
AND
OR
```

単項前置演算子は次のものがサポートされています。

```
 !    ~    NOT
```

COLLATE 演算子は単項後置演算子と見なすことができます。COLLATE 演算子は優先順位が最も高く、常に他のどの前置単項演算子や二項演算子よりも強く結び付きます。

等号演算子と不等号演算子はそれぞれ 2 種類あります。等号演算子は = または ==、不等号演算子は != または <> です。

|| はストリングを連結する演算子で、そのオペランドである 2 つのストリングを 1 つにつなげます。

% 演算子は、右オペランドを法とする左オペランドの剰余を返します。

二項演算子の結果は数値です。ただし、|| 連結演算子だけは例外で、結果がストリングになります。

SQL 演算子

LIKE

LIKE 演算子は、パターンマッチングによる比較を行います。

```
expr     ::=  (column-name | expr) LIKE pattern
pattern  ::=  '[ string | % | _ ]'
```

LIKE 演算子の右オペランドにはパターンを指定し、左オペランドにはそのパターンと照合するストリングを指定します。パターン内のパーセント記号（%）はワイルドカード文字で、ストリング内の 0 個以上連続した文字に一致します。パターン内のアンダースコア（_）はストリング内の任意の 1 文字に一致します。その他の文字はすべて、その文字そのものとして、大文字と小文字を区別せずに照合されます。（メモ：データベースエンジンでは、7 ビットラテン文字の大文字と小文字のみが理解されます。したがって、8 ビット iso8859 文字または UTF-8 文字では、LIKE 演算子は大文字と小文字を区別します。例えば、'a' LIKE 'A' という式は TRUE になりますが、'æ' LIKE 'Æ' は FALSE になります）。ラテン文字の大文字と小文字を区別するかどうかを切り替えるには、SQLConnection.caseSensitiveLike プロパティを使用します。

オプションの ESCAPE 句を指定する場合、ESCAPE キーワードの後に続く式は、1 文字で構成されるストリングに評価される必要があります。この文字は、LIKE パターンの中で、リテラルのパーセントまたはアンダースコア文字を照合するために使用できます。パターン内でエスケープ文字の後にパーセント記号、アンダースコア、またはエスケープ文字自体を記述すると、それらはそれぞれ、ストリング内のリテラルのパーセント記号、アンダースコア、またはエスケープ文字に一致します。

GLOB

GLOB 演算子は LIKE と似ていますが、ワイルドカードとして UNIX のファイルグロビング構文を使用します。LIKE とは異なり、GLOB では大文字と小文字は区別されます。

IN

IN 演算子は、左オペランドが、右オペランド（括弧で囲まれた値のリスト）内のいずれかの値と等しいかどうかを返します。

```
in-expr         ::=  expr [NOT] IN ( value-list ) |
                     expr [NOT] IN ( select-statement ) |
                     expr [NOT] IN [database-name.] table-name
value-list      ::=  literal-value [, literal-value]*
```

右オペランドには複数のリテラル値をカンマで区切って記述するか、SELECT 文を記述します。IN 演算子の右オペランドとして SELECT ステートメントを使用する際の説明と制限については、「式」の「SELECT ステートメント」を参照してください。

BETWEEN...AND

BETWEEN...AND 演算子は、2 つの式に対して >= 演算子と <= 演算子を使用するのと同じです。例えば、x BETWEEN y AND z という式は x >= y AND x <= z と同じです。

NOT

NOT は否定の演算子です。GLOB、LIKE および IN 演算子の前に NOT キーワードを付けると、テストの意味が逆になります（つまり、値が指定のパターンと一致しないことが確認されます）。

パラメーター

パラメーターは、式の中でリテラル値を表すためのプレースホルダーを指定します。このプレースホルダーには、SQLStatement.parameters 結合配列に値を設定することで、実行時に値が格納されます。パラメーターは次の 3 つの形式をとります。

?	疑問符はインデックス付きのパラメーターを示します。この形式のパラメーターには、SQL ステートメントに登場した順に数値インデックス値が割り当てられます。インデックス値は 0 から始まります。
:AAAA	コロンの後に識別子名が付いた形式は、AAAA という名前の名前付きパラメーター用の場所を保持します。名前付きパラメーターにも、SQL 文に登場した順に数値インデックスが付けられます。混乱を避けるために、名前付きパラメーターとインデックス付きパラメーターを混在させないことをお勧めします。
@AAAA	"@" はコロンと同じです。

サポートされていない SQL 機能

Adobe AIR でサポートされていない標準 SQL 要素は次のとおりです。

FOREIGN KEY 制約 FOREIGN KEY 制約は、解析はされますが適用されません。

トリガー FOR EACH STATEMENT トリガーはサポートされていません（トリガーはすべて FOR EACH ROW にする必要があります）。INSTEAD OF トリガーはテーブルではサポートされていません（INSTEAD OF トリガーはビューに対してのみ使用できます）。再帰トリガー（自分自身をトリガーするトリガー）はサポートされていません。

ALTER TABLE ALTER TABLE コマンドでは、RENAME TABLE バリアントと ADD COLUMN バリアントのみがサポートされています。その他の種類の ALTER TABLE 操作（DROP COLUMN、ALTER COLUMN、ADD CONSTRAINT など）は無視されます。

ネストされたトランザクション 単一のアクティブトランザクションのみが許可されます。

RIGHT および FULL OUTER JOIN RIGHT OUTER JOIN または FULL OUTER JOIN はサポートされていません。

更新可能なビュー ビューは読み取り専用です。ビューに対して DELETE ステートメント、INSERT ステートメント、または UPDATE ステートメントは実行できません。ビューを DELETE、INSERT、または UPDATE しようとしたときに起動される INSTEAD OF トリガーはサポートされており、このトリガーの本体で基になるテーブルを更新できます。

GRANT と REVOKE データベースは通常のディスクファイルです。つまり、アクセス許可として適用できるのは、基になるオペレーティングシステムの通常のファイルアクセス許可のみです。クライアント / サーバー型の RDBMS でよく見られる GRANT コマンドと REVOKE コマンドは実装されていません。

次に示す SQL 要素と SQLite 機能は、一部の SQLite 実装ではサポートされていますが、Adobe AIR ではサポートされていません。これらの機能のほとんどは、SQLConnection クラスのメソッドを通じて使用できます。

トランザクション関連の SQL 要素（BEGIN、END、COMMIT、ROLLBACK）この機能は SQLConnection クラスのトランザクション関連メソッド（SQLConnection.begin()、SQLConnection.commit() および SQLConnection.rollback()）を通じて使用できます。

ANALYZE この機能は SQLConnection.analyze() メソッドを通じて使用できます。

ATTACH この機能は SQLConnection.attach() メソッドを通じて使用できます。

COPY このステートメントはサポートされていません。

CREATE VIRTUAL TABLE このステートメントはサポートされていません。

DETACH この機能は SQLConnection.detach() メソッドを通じて使用できます。

PRAGMA このステートメントはサポートされていません。

VACUUM この機能は SQLConnection.compact() メソッドを通じて使用できます。

システムテーブルにはアクセスできません システムテーブル（sqlite_master や、「sqlite_」接頭辞の付いたその他のテーブルなど）は、SQL ステートメントでは使用できません。ランタイムには、オブジェクト指向的な方法でスキーマデータにアクセスできるスキーマ API が含まれています。詳しくは、「SQLConnection.loadSchema() メソッド」を参照してください。

正規表現関数（MATCH() と REGEX()） これらの関数は、SQL ステートメントでは使用できません。

次の機能は、多くの SQLite の実装と Adobe AIR とで違いがあります。

ステートメントパラメーターのインデックス 多くの実装では、ステートメントパラメーターのインデックスは 1 から始まります。それに対して、Adobe AIR の場合、ステートメントパラメーターのインデックスは 0 から始まります（最初のパラメーターのインデックス値は 0、2 番目のパラメーターのインデックス値は 1 というようになります）。

INTEGER PRIMARY KEY 列定義 多くの実装では、INTEGER PRIMARY KEY として正確に定義されている列だけが、テーブルの実際の主キー列として使用されます。このような実装では、別のデータ型（通常は int などの、INTEGER のシノニム）を使用した場合、その列は内部主キーとして使用されません。ただし、Adobe AIR では、int データ型（および他の INTEGER のシノニム）は、INTEGER と完全に同等とみなされます。したがって、int PRIMARY KEY として定義された列は、テーブルの内部の主キーとして使用されます。詳しくは、「CREATE TABLE」および「列親和型」を参照してください。

追加の SQL 機能

次の列親和型は、SQLite ではデフォルトでサポートされていませんが、Adobe AIR ではサポートされています（SQL のすべてのキーワードと同様に、これらのデータ型の名前では大文字と小文字が区別されません）。

Boolean Boolean クラスに対応します。

Date Date クラスに対応します。

int int クラスに対応します（INTEGER 列親和型に相当）。

Number Number クラスに対応します（REAL 列親和型に相当）。

Object Object クラスまたは AMF3 を使用して直列化および非直列化することが可能なすべてのサブクラスに対応します（これには、カスタムクラスを含む多くのクラスが含まれますが、表示オブジェクトまたは表示オブジェクトをプロパティとして持つオブジェクトは含まれません）。

String String クラスに対応します（TEXT 列親和型に相当）。

XML ActionScript（E4X）XML クラスに対応します。

XMLList ActionScript（E4X）XMLList クラスに対応します。

次のリテラル値は、SQLite ではデフォルトでサポートされていませんが、Adobe AIR ではサポートされています。

true BOOLEAN 型の列に使用するリテラルブール値 true を表します。

false BOOLEAN 型の列に使用するリテラルブール値 false を表します。

データ型のサポート

ほとんどの SQL データベースとは異なり、Adobe AIR の SQL データベースエンジンでは、テーブルの列に特定の型の値を格納する必要はなく、特定の型が強制されることもありません。その代わりに、格納クラスと列親和型という 2 つの概念を使用してデータ型が管理されます。この節では、格納クラスと列親和型について説明し、様々な条件下でデータ型の違いがどのように解決されるのかを示します。

- 355 ページの「格納クラス」
- 356 ページの「列親和型」
- 358 ページの「データ型と比較演算子」
- 359 ページの「データ型と数値演算子」
- 359 ページの「データ型とソート」
- 359 ページの「データ型とグループ化」
- 359 ページの「データ型と複合 SELECT 文」

格納クラス

格納クラスは、データベースへの値の格納に使用される実際のデータ型を表します。次の格納クラスがデータベースで使用されます。

NULL 値は NULL 値です。

INTEGER 値は符号付き整数です。

REAL 値は浮動小数点数値です。

TEXT 値はテキストストリングです（256 MB に制限されます）。

BLOB 値はバイナリラージオブジェクト（BLOB）です。生のバイナリデータとも呼ばれ、256 MB に制限されます。

SQL 文に埋め込まれたリテラルとしてデータベースに渡される値、またはパラメーターを使用して SQL 文にバインドされた値はすべて、SQL 文が実行される前に格納クラスに割り当てられます。

SQL 文に含まれるリテラルは、一重引用符または二重引用符で囲まれている場合は格納クラス TEXT に、引用符で囲まれておらず、小数点または指数のない数字として指定されている場合は INTEGER に、引用符で囲まれておらず、小数点または指数の付いた数字として指定されている場合は REAL に、値が NULL の場合は NULL に、それぞれ割り当てられます。リテラルを BLOB に割り当てるには、X'ABCD' 表記を使用します。詳しくは、「式」の「リテラル値」を参照してください。

SQLStatement.parameters 結合配列を使用してパラメーターとして提供された値は、バインドされたネイティブのデータ型に最も近い格納クラスに割り当てられます。例えば、int 値は INTEGER 格納クラスに、Number 値は REAL 格納クラスに、String 値は TEXT 格納クラスに、ByteArray オブジェクトは BLOB 格納クラスに、それぞれ割り当てられます。

列親和型

列の親和型は、その列に格納するデータに推奨される型です。INSERT ステートメントまたは UPDATE ステートメントで値を列に格納すると、ランタイムはその値のデータ型を指定された親和型に変換しようとします。例えば、TEXT を親和型とする列に Date 型の値（ActionScript または JavaScript の Date インスタンス）を挿入すると、Date 型の値は String 表現に変換されます。これは、データベースに格納する前にオブジェクトの toString() メソッドを呼び出した場合に相当します。値を指定された親和型に変換できない場合はエラーが発生し、処理は行われません。SELECT ステートメントを使用してデータベースから値を取得すると、格納時に別のデータ型から変換されているかどうかに関係なく、親和型に対応するクラスのインスタンスが返されます。

NULL 値を受け入れる列の場合、ActionScript または JavaScript の値 null をパラメーター値として指定し、列に NULL を格納することができます。NULL 格納クラスの値を SELECT ステートメントで取得すると、列の親和型に関係なく、常に ActionScript または JavaScript の null 値が返されます。NULL 値を受け入れる列の場合、null を許容しない Number や Boolean などの型に値をキャストする前に、その列から取得した値が null かどうか必ず確認されます。

データベースの各列には、次のいずれかの親和型が割り当てられます。

- TEXT（または String）
- NUMERIC
- INTEGER（または int）
- REAL（または Number）
- Boolean
- Date
- XML
- XMLLIST
- Object
- NONE

TEXT（または String）

TEXT または String 親和型の列には、記憶クラス NULL、TEXT、または BLOB を使用してすべてのデータが格納されます。数値データを TEXT 親和型の列に挿入すると、データがテキスト形式に変換されてから格納されます。

NUMERIC

NUMERIC 親和型の列には、記憶クラス NULL、REAL、または INTEGER を使用して値が格納されます。テキストデータを NUMERIC 列に挿入すると、データを格納する前に整数または実数への変換が試みられます。変換に成功した場合は、INTEGER または REAL 記憶クラスを使用して値が格納されます（例えば、値 '10.05' は REAL 記憶クラスに変換されてから格納されます）。変換できない場合は、エラーが発生します。NULL 値の変換は行われません。NUMERIC 列から取得し

た値は、値に適した最も具体的な数値型のインスタンスとして返されます。つまり、値が正の整数または 0 の場合、uint インスタンスが返され、負の整数の場合、int インスタンスが返されます。また、値に浮動小数点部分（非整数）がある場合、Number インスタンスが返されます。

INTEGER（または int）

INTEGER 親和型を使用する列は、NUMERIC 親和型の列と同じように動作します。相違点は、浮動小数点部分のない実数値（Number インスタンスなど）または浮動小数点部分のない実数値に変換できるテキスト値を挿入した場合に、値が整数に変換され、INTEGER 記憶クラスを使用して格納される点のみです。浮動小数点部分のある実数値を格納しようとすると、エラーが発生します。

REAL（または Number）

REAL 親和型または NUMBER 親和型の列も NUMERIC 親和型の列と同じように動作しますが、整数値が強制的に浮動小数点表現に変更される点が異なります。REAL 型の列の値は、データベースから常に Number インスタンスとして返されます。

Boolean

Boolean 親和型の列には、true または false の値が格納されます。Boolean 列は、値として ActionScript または JavaScript の Boolean インスタンスを受け入れます。コードで String 型の値が格納される際には、長さが 0 より大きいストリングの場合は true、空のストリングの場合は false と判別されます。コードで数値データが格納される際には、格納される値が 0 以外の場合は true、0 の場合は false と判別されます。SELECT ステートメントを使用して BOOLEAN 値を取得すると、その値は Boolean インスタンスとして返されます。NULL 以外の値は INTEGER 記憶クラスを使用して格納され（false の場合は 0、true の場合は 1）、データの取得時に Boolean オブジェクトに変換されます。

Date

Date 親和型の列には、日付と時刻の値が格納されます。Date 列は値として ActionScript または JavaScript の Date インスタンスを受け入れるように設計されており、Date 列に String 値を格納しようとすると、ランタイムでユリウス日付に変換されます。変換に失敗するとエラーが発生します。コードで Number、int、または uint の値を格納しようとした場合、日付の検証は行われず、有効なユリウス日付値と見なされます。SELECT ステートメントを使用して Date 値を取得すると自動的に Date インスタンスに変換されます。Date 値は REAL 記憶クラスを使用してユリウス日付値として格納されるので、ソート操作や比較操作も期待どおりに動作します。

XML または XMLList

XML または XMLList 親和型を使用する列には、XML 構造が格納されます。コードで SQLStatement パラメーターを使用して XML 列にデータを格納すると、ランタイムでは ActionScript の XML() または XMLList() 関数を使用して値の変換と検証が行われます。値を有効な XML に変換できない場合、エラーが発生します。リテラル SQL テキスト値（例：INSERT INTO (col1) VALUES ('Invalid XML (no closing tag)'）を使用してデータを格納すると値の解析や検証は行われず、正しい形式と判断されます。無効な値を格納した場合、その値の取得時に空の XML オブジェクトが返されます。XML および XMLList のデータは TEXT 記憶クラスまたは NULL 記憶クラスを使用して格納されます。

Object

Object 親和型の列には、Object クラスインスタンスおよび Object サブクラスインスタンスなど（例：Array インスタンスやカスタムクラスインスタンスなど）の ActionScript または JavaScript の複合オブジェクトを格納します。Object 列のデータは AMF3 形式で直列化され、BLOB 記憶クラスを使用して格納されます。値の取得時に AMF3 形式から非直列化され、格納されていたクラスのインスタンスとして返されます。表示オブジェクトなど一部の ActionScript クラスは、元のデータ型のインスタンスとして非直列化できません。カスタムクラスインスタンスの場合、格納する前に flash.net.registerClassAlias() メソッドを使用して（または Flex でクラス宣言に [RemoteObject] メタデータを追加して）、クラスのエイリアスを登録する必要があります。また、このデータを取得する前に、クラスに対して同じエイリアスを登録する必要があります。本質的に非直列化できないクラスであるため、あるいはクラスのエイリアスが未指定または不一致であるために適切に非直列化できないデータは、格納されている元のインスタンスに対応するプロパティと値を持つ匿名オブジェクト（Object クラスインスタンス）として返されます。

NONE

NONE 親和型の列には、優先される格納クラスはありません。データを挿入する前にデータの変換は行われません。

親和型の決定

列の親和型は、CREATE TABLE 文で列に対して宣言された型によって決まります。親和型を決定する際は、次の規則が適用されます（大文字と小文字は区別されません）。

- 列のデータ型に "CHAR"、"CLOB"、"STRING"、"TEXT" のいずれかのストリングが含まれている場合は、TEXT/String 親和型になります。VARCHAR 型にはストリング "CHAR" が含まれているので、TEXT 親和型が割り当てられることに注意してください。
- 列のデータ型にストリング "BLOB" が含まれている場合、またはデータ型が指定されていない場合、NONE 親和型になります。
- 列のデータ型にストリング "XMLL" が含まれている場合は、XMLList 親和型になります。
- 列のデータ型がストリング "XML" の場合は、XML 親和型になります。
- 列のデータ型にストリング "OBJE" が含まれている場合は、Object 親和型になります。
- 列のデータ型にストリング "BOOL" が含まれている場合は、Boolean 親和型になります。
- 列のデータ型にストリング "DATE" が含まれている場合は、Date 親和型になります。
- 列のデータ型にストリング "INT" が含まれている場合は（"UINT" など）、INTEGER/int 親和型が列に割り当てられます。
- 列のデータ型に "REAL"、"NUMB"、"FLOA"、"DOUB" のいずれかのストリングが含まれている場合は、REAL/Number 親和型になります。
- 上記以外の場合は、NUMERIC 親和型になります。
- CREATE TABLE t AS SELECT... ステートメントを使用してテーブルを作成する場合は、どの列にもデータ型が指定されていないので、すべての列が NONE 親和型になります。

データ型と比較演算子

比較操作では、二項比較演算子の =、<、<=、>= および != がサポートされています。また、複数の値のいずれかに一致するかどうかをチェックする IN 演算子と、三項比較演算子の BETWEEN も使用できます。これらの演算子について詳しくは、「演算子」を参照してください。

比較の結果は、比較する 2 つの値の格納クラスによって異なります。2 つの値を比較するときは、次の規則が適用されます。

- 格納クラス NULL の値は、他のどの値（格納クラス NULL の別の値も含む）よりも小さいと見なされます。
- INTEGER 値または REAL 値は、どの TEXT 値または BLOB 値よりも小さいと見なされます。INTEGER または REAL を別の INTEGER または REAL と比較したときは、数値による比較が行われます。
- TEXT 値は BLOB 値より小さいと見なされます。2 つの TEXT 値を比較するときは、二項比較が行われます。
- 2 つの BLOB 値を比較した結果は、常に二項比較を使用して決定されます。

三項演算子 BETWEEN は常に、同等の二項表現として再構成されます。例えば、a BETWEEN b AND c は a >= b AND a <= c に再構成されます。この処理は、この式を評価するために必要なそれぞれの比較において a に異なる親和型が適用される場合でも変わりません。

a IN (SELECT b ....) という形式の式も、上記の二項比較に関する 3 つの規則に従って（つまり、a = b と同じように）処理されます。例えば、b が列の値で a が式の場合は、比較が行われる前に、b の親和型が a に適用されます。式 a IN (x, y, z) は a = +x OR a = +y OR a = +z として再構成されます。IN 演算子の右の値（この例では x、y および z の値）は、たとえそれが列の値であっても、式と見なされます。IN 演算子の左の値が列の場合は、その列の親和型が使用されます。左の値が式の場合は、変換は行われません。

また、COLLATE 句を使用して比較方法を変更することもできます。詳しくは、「COLLATE」を参照してください。

データ型と数値演算子

サポートされている数値演算子（*、/、%、+ および -）のいずれを使用するときも、式を評価する前に演算子の各オペランドに数値親和型が適用されます。どのオペランドも NUMERIC 記憶クラスに変換できない場合、その式は NULL に評価されます。

連結演算子 || を使用するときは、式が評価される前に各オペランドが TEXT 格納クラスに変換されます。どのオペランドも TEXT 記憶クラスに変換できない場合、その式の結果は NULL になります。このように、値を変換できない状況は次の 2 つの場合に起こります。1 つはオペランドの値が NULL の場合、もう 1 つはオペランドの値が TEXT 以外の記憶クラスを含む BLOB である場合です。

データ型とソート

ORDER BY 句によって値をソートしたときの並び順は、最初が記憶クラス NULL の値、次が数字順に INTEGER 値と REAL 値、その次がバイナリ順または指定された照合順序（BINARY または NOCASE）に基づく TEXT 値、最後がバイナリ順の BLOB 値です。ソートの前に格納クラスの変換は行われません。

データ型とグループ化

GROUP BY 句を使用して値をグループ化するとき、記憶クラスの異なる値は別のものと見なされます。ただし、INTEGER 値と REAL 値は、数値的に等しい場合には同じと見なされます。GROUP BY 句の結果として、値に親和型が適用されることはありません。

データ型と複合 SELECT 文

複合 SELECT 演算子の UNION、INTERSECT および EXCEPT は、値どうしを暗黙的に比較します。これらの比較が行われる前に、それぞれの値に親和型が適用される場合があります。可能であれば、複合 SELECT 文の結果セットの 1 つの列で返されるすべての値に同じ親和型が適用されます。適用される親和型は、その位置に（他の種類の式ではなく）列値を持つ最初の構成 SELECT 文によって返される列の親和型です。特定の複合 SELECT 列について、どの構成 SELECT ステートメントも列値を返さない場合、その列では親和型を適用せずに値の比較が行われます。

第 27 章：SQL エラー詳細メッセージ、ID、および引数

SQLError クラスは、Adobe AIR ローカル SQL データベースでの作業中に発生する可能性のある様々なエラーを表します。発生した各例外用に、SQLError インスタンスには、英語のエラーメッセージを含む details プロパティが用意されています。さらに、各エラーメッセージには SQLError オブジェクトの detailID プロパティで使用できる関連する一意の識別子があります。detailID プロパティを使用して、アプリケーションは特定の details エラーメッセージを識別できます。アプリケーションは、エンドユーザーのロケールの言語で、代替テキストをエンドユーザーに提供できます。detailArguments 配列の引数の値は、エラーメッセージ列の適切な場所と置き換えることができます。これは、アプリケーションで特定のロケールのエンドユーザーに、直接エラーの details プロパティエラーメッセージを表示する場合に便利です。

次の表に、detailID 値の一覧と、関連するエラーメッセージの内容を示します。メッセージのプレースホルダーテキストは detailArguments 値がランタイムによって置き換えられる場所を示しています。この一覧は SQL データベース操作で発生する可能性のあるエラーメッセージをローカライズする際のソースとして使用できます。

SQLError detailID	エラー詳細メッセージおよびパラメーター
1001	接続が閉じられました。
1102	この操作を実行するにはデータベースが開いている必要があります。
1003	%s [,|and %s] パラメーター名がパラメーターのプロパティで見つかりましたが、指定された SQL では見つかりませんでした。
1004	パラメーター数が一致しません。SQL では指定された %d が、またパラメータのプロパティでは設定された %d 値が見つかりました。%s [,|and %s] の値が必要です。
1005	自動最適化をオンにできませんでした。
1006	pageSize 値を設定できませんでした。
1007	'%s' という名前（'%s' 型、データベース '%s'）のスキーマオブジェクトが見つかりませんでした。
1008	'%s' という名前のスキーマオブジェクトがデータベース '%s' で見つかりませんでした。
1009	'%s' 型（データベース '%s'）のスキーマオブジェクトが見つかりませんでした。
1010	データベース '%s' でスキーマオブジェクトが見つかりませんでした。
2001	パーサーのスタックオーバーフロー
2002	関数 '%s' の引数の数が多すぎます。
2003	'%s' に近い：シンタックスエラー
2004	'%s' という名前の別のテーブルまたはインデックスが既に存在します。
2005	SQL では PRAGMA は許可されていません。
2006	書き込みできないディレクトリです。
2007	不明なまたはサポートされていない結合のタイプ：'%s %s %s'
2008	RIGHT および FULL OUTER JOIN は現在サポートされていません。
2009	NATURAL 結合に ON または USING 句がない可能性があります。
2010	同じ結合に ON 句および USING 句の両方を持つことはできません。
2011	列 '%s' を使用して結合できません - どちらのテーブルにも列がありません。
2012	式の一部である SELECT には 1 つの結果しか許可されていません。
2013	テーブル '[%s.]%s' が存在しません。
2014	指定されたテーブルがありません。
2015	結果セットの列数が多すぎます |'%s' の列数が多すぎます。
2016	%s ORDER|GROUP BY 項が範囲外です。1 ～ %d である必要があります。
2017	ORDER BY 句に項が多すぎます。
2018	%s ORDER BY 項が範囲外です。1 ～ %d である必要があります。
2019	%r ORDER BY 項が結果セットのどの列とも一致しません。
2020	ORDER BY 句は '%s' の前ではなく後に来る必要があります。
2021	LIMIT 句は '%s' の前ではなく後に来る必要があります。
2022	'%s' の左および右にある SELECT に、結果列数と同じ列数がありません。
2023	HAVING の前に GROUP BY 句が必要です。

2024	GROUP BY 句では集約関数は使用できません。
2025	集約の DISTINCT の後に式が続く必要があります。
2026	複合 SELECT の項が多すぎます。
2027	ORDER|GROUP BY 句の項が多すぎます。
2028	一時トリガーに修飾名がない可能性があります。
2030	トリガー '%s' は既に存在します。
2032	'%s' ビューに BEFORE|AFTER トリガーを作成することはできません。
2033	テーブル '%s' に INSTEAD OF トリガーを作成することはできません。
2034	トリガー '%s' が存在しません。
2035	再帰トリガーはサポートされていません（'%s'）。
2036	列 %s[.%s[.%s]] が存在しません。
2037	SQL では VACUUM は許可されていません。
2043	テーブル '%s': インデックス処理関数で無効なプランが返されました。
2044	結合に最大 %d のテーブルがあります。
2046	PRIMARY KEY 列を追加できません。
2047	UNIQUE 列を追加できません。
2048	デフォルト値が NULL の NOT NULL 列は追加できません。
2049	デフォルト値が定数でない列は追加できません。
2050	ビューに列を追加することはできません。
2051	SQL では ANALYZE は許可されていません。
2052	無効な名前：'%s'
2053	SQL では ATTACH は許可されていません。
2054	%s '%s' はデータベース '%s' のオブジェクトを参照できません。
2055	'[%s.]%s.%s' へのアクセスは禁止されています。
2056	許可されていません。
2058	ビュー '[%s.]%s' が存在しません。
2060	一時テーブル名は無修飾である必要があります。
2061	テーブル '%s' は既に存在します。
2062	'%s' という名前のインデックスは既に存在します。
2064	列の名前 '%s' が重複しています。
2065	テーブル '%s' に主キーが 2 つ以上あります。
2066	AUTOINCREMENT は INTEGER PRIMARY KEY でのみ許可されています。
2067	照合シーケンス '%s' が存在しません。
2068	ビューでパラメーターが有効ではありません。
2069	ビュー '%s' が循環的に定義されています。
2070	テーブル '%s' が削除されていない可能性があります。
2071	DROP VIEW を使用してビュー '%s' を削除します。
2072	DROP TABLE を使用してテーブル '%s' を削除します。
2073	'%s' の外部キーはテーブル '%s' の 1 列のみを参照している必要があります。
2074	外部キーの列数が参照されているテーブルの列数と一致していません。
2075	外部キー定義の列 '%s' が不明です。
2076	テーブル '%s' がインデックス付けされていない可能性があります。
2077	ビューがインデックス付けされていない可能性があります。
2080	競合した ON CONFLICT 句が指定されています。
2081	インデックス '%s' が存在しません。
2082	UNIQUE または PRIMARY KEY 制約と関連付けられたインデックスは削除できません。
2083	SQL では BEGIN は許可されていません。
2084	SQL では COMMIT は許可されていません。
2085	SQL では ROLLBACK は許可されていません。
2086	一時テーブルを格納するための一時データベースファイルを開くことができません。
2087	再度インデックス付けするオブジェクトを識別できません。
2088	テーブル '%s' は変更されていない可能性があります。
2089	'%s' はビューであるため変更できません。
2090	変数は ?0 ～ ?%d の間である必要があります。
2092	エイリアス処理された集約 '%s' が誤って使用されています。
2093	列の名前 '[%s.[%s.]]%s' があいまいです。
2094	関数 '%s' が存在しません。

2095	関数 '%s' の引数の数が間違っています。
2096	CHECK 制約ではサブクエリは禁止されています。
2097	CHECK 制約ではパラメーターは禁止されています。
2098	式のツリーが大きすぎます（最大の深さ %d）。
2099	RAISE() はトリガープログラム内でのみ使用できます。
2100	テーブル '%s' の %d 列に %d 値が設定されていませんでした。
2101	データベーススキーマがロックされています：'%s'
2102	ステートメントが長すぎます。
2103	アクティブなステートメントがあるため照合シーケンスを削除 / 変更できません。
2104	アタッチされているデータベースが多すぎます。最大数は %d です。
2105	トランザクション内ではデータベースの ATTACH はできません。
2106	データベース '%s' は既に使用されています。
2108	アタッチされたデータベースでは、メインデータベースと同じテキストエンコードを使用する必要があります。
2200	メモリ不足です。
2201	データベースを開くことができません。
2202	トランザクション内ではデータベースの DETACH はできません。
2203	データベース '%s' をデタッチできません。
2204	データベース '%s' がロックされています。
2205	データベースの読み取りロックを取得できません。
2206	列 '%s'['.'%s'] が一意ではありません。
2207	不正なデータベーススキーマ
2208	サポートされていないファイル形式
2209	認識できないトークン：'%s'
2300	テキスト値を数値に変換できませんでした。
2301	ストリング値を日付に変換できませんでした。
2302	データが失われることなく、浮動小数値を整数に変換できませんでした。
2303	トランザクションをロールバックできません。SQL ステートメントが実行中です。
2304	トランザクションをコミットできません。SQL ステートメントが実行中です。
2305	データベーステーブルがロックされています：'%s'
2306	読み取り専用テーブル
2307	ストリングまたは BLOB が大きすぎます。
2309	インデックス付けされた列を書き込み用に開くことができません。
2400	型 %s の値を開くことができません。
2401	rowid %s が存在しません。
2402	内部的な使用のために予約されたオブジェクト名：'%s'
2403	ビュー '%s' を変更することはできません。
2404	列 '%s' のデフォルト値が定数ではありません。
2405	関数 '%s' の使用が許可されていません。
2406	集約関数 '%s' が誤って使用されています。
2407	集約 '%s' が誤って使用されています。
2408	データベース '%s' が存在しません。
2409	テーブル '%s' に '%s' という名前の列がありません。
2501	モジュール '%s' が存在しません。
2508	セーブポイント '%s' が存在しません。
2510	ロールバックできません。アクティブなトランザクションがありません。
2511	コミットできません。アクティブなトランザクションがありません。