



メディアアプリケーション 開発ガイド

商標

Afterburner、AppletAce、Attain、Attain Enterprise Learning System、Attain Essentials、Attain Objects for Dreamweaver、Authorware、Authorware Attain、Authorware Interactive Studio、Authorware Star、Authorware Synergy、Backstage、Backstage Designer、Backstage Desktop Studio、Backstage Enterprise Studio、Backstage Internet Studio、Contribute、Design in Motion、Director、Director Multimedia Studio、Doc Around the Clock、Dreamweaver、Dreamweaver Attain、Drumbeat、Drumbeat 2000、Extreme 3D、Fireworks、Flash、Fontographer、FreeHand、FreeHand Graphics Studio、Generator、Generator Developer's Studio、Generator Dynamic Graphics Server、Knowledge Objects、Knowledge Stream、Knowledge Track、Lingo、Live Effects、Macromedia、Macromedia M Logo & Design、Macromedia Contribute、Macromedia Flash、Macromedia Xres、Macromind、Macromind Action、MAGIC、Mediamaker、Object Authoring、Power Applets、Priority Access、Roundtrip HTML、Scriptlets、SoundEdit、ShockRave、Shockmachine、Shockwave、Shockwave Remote、Shockwave Internet Studio、Showcase、Tools to Power Your Ideas、Universal Media、Virtuoso、Web Design 101、Whirlwind、および Xtra は、Macromedia, Inc. の商標であり、米国およびその他の国の法域で登録される場合があります。本マニュアルにおけるその他の製品名、ロゴ、デザイン、タイトル、語句は、Macromedia, Inc. またはその他の団体の商標、サービスマーク、または商号である場合があります、米国およびその他の国の特定の法域で登録されている場合があります。

サードパーティー情報

Jabber は、Jabber Software Foundation の登録商標です。



Sorenson™ Spark™ ビデオ圧縮および圧縮解除テクノロジーは、Sorenson Media, Inc. のライセンス供与によって提供されます。

本マニュアルには、Macromedia 社が管理していない、サードパーティーの Web サイトへのリンクが掲載されていますが、Macromedia 社はいかなるリンク先サイトの内容についても責任を持ちません。本マニュアルに記載されているサードパーティーの Web サイトには、自己責任においてアクセスしてください。Macromedia 社はこれらのリンクを便宜上の目的においてのみ掲載しています。リンクを掲載することにより、Macromedia 社がこれらのサードパーティーのサイトの内容について何らかの責任を持つことを示すものではありません。

Copyright © 2002-2005 Macromedia, Inc. All rights reserved. 本マニュアルの一部または全体を Macromedia, Inc. の書面による許諾を受けることなく、コピー、複写、複製、翻訳、電子的または物理的に変換することは禁じられています。

マニュアル制作スタッフ

プロジェクト管理 : Suzanne Smith

執筆 : John Norton、Suzanne Smith

編集 : Evelyn Eldridge、Mary Ferguson、Lisa Stanziano、Anne Szabla

制作管理 : Adam Barnett

メディアデザイン・制作 : Aaron Begley、Paul Benkman、John Francis、Mario Reynoso

初版 : 2005 年 10 月

Macromedia, Inc.

601 Townsend St.

San Francisco, CA 94103

マクロメディア株式会社

東京都港区赤坂 2-17-22 赤坂ツインタワー本館 13F

目次

このマニュアルについて	7
対象となる読者	7
Flash Media Server ドキュメントについて	8
Flash Media Server サポート	9
表記規則	9
第1章：ファーストステップ	11
インストールとシステム要件	11
ハードウェアおよびソフトウェアのセットアップ	11
開発環境の作成	13
開発タスクの概要	14
アプリケーションとアプリケーションインスタンスのデプロイメント	15
サーバーサイドとクライアントサイドのファイルの格納	15
アプリケーションのサーバーサイドスクリプトファイルの格納	17
アプリケーションインスタンスの使用	17
Flash Media Server で使用されているファイル形式	19
サーバーへの接続	20
サービスを開始する	20
サーバーへの接続を開く	20
第2章：Flash Media Server アーキテクチャ	23
Flash Media Server アーキテクチャの概要	24
ストリームと共有オブジェクトについて	25
外部データソースへの接続	28
アプリケーションの作成とデプロイメントのワークフロー	29
アプリケーションフローの概要	31
接続フロー	32
リモートメソッドの呼び出し	33
共有オブジェクトフローの概要	36
第3章：メディアクラスの使用	39
Flash Media Server のクラスについて	39
クライアントサイドクラスについて	39

サーバーサイドクラスについて	41
クライアントとサーバーの通信	42
共有オブジェクトの概要	43
Application クラスについて	44
application.onConnect ハンドラの使用	45
application.onDisconnect ハンドラの使用	47
コンポーネントベースのアプリケーションにおけるイベントの処理	48
Camera クラス	48
カメラをオフにする	48
帯域幅速度に合わせた設定	49
複数のアプリケーションによる 1 台のカメラの使用	49
Client クラス	49
Microphone クラス	50
オーディオフィードバックの回避	50
マイクのオン状態の維持	50
NetConnection クラス (クライアントサイド)	51
NetStream クラス	52
ストリームにおける複数のデータタイプの使用	52
ActionScript によるストリーム長の取得	52
ストリームのバッファリング	53
ストリームの再生ステータスについて	53
SharedObject クラス	54
共有オブジェクトの同期化について	54
共有オブジェクトスロットの効果的な使用	54
リモート共有オブジェクトのフラッシュ	55
共有オブジェクトの同期化についての問題の回避	55
Stream クラス	57
System クラス	57
Video クラスについて	58
Video オブジェクトの動的な作成について	58
フレームレートについて	58
第 4 章: メディアファイルの使用	59
ビデオの操作	59
ライブ Web イベントの作成について	59
ストリーム配信のカスタマイズ	60
MP3 ファイルの操作	64
第 5 章: アプリケーションのデバッグと監視	67
Management Console を使用したアプリケーションのデバッグと監視	67
Management Console のサーバーへの接続	68

[View Applications] パネルについて	69
[Live Log] パネルについて	71
[Client] パネルについて	72
[Shared Objects] パネル	73
[Streams] パネル	75
[Performance] パネルについて	76
デバッグ接続の使用	78
onStatus イベントハンドラ	78
スクリプト内で onStatus ハンドラを記述する場所	79
onStatus ハンドラの無効化について	79
System.onStatus ハンドラについて	80
NetConnection.Connect.Failed メッセージのデバッグについて	80
オブジェクトのプロパティのトレース	81
第 6 章: アプリケーション開発のヒント	83
ファイルタイプとパスについて	83
共有オブジェクトファイルについて	84
サーバー間での移植性	85
クライアントサーバーのスクリプトの相互依存性	86
複数のスクリプトファイルの使用	87
サーバーサイドのスクリプトファイルのアーカイブとコンパイル	88
スクリプトのバイトコードへのコンパイル	91
[Macromedia Flash Player 設定] パネルの強制表示	93
帯域幅の管理	94
ダブルバイト言語のアプリケーションの記述	95
アプリケーションのアンロードと再ロード	96
ダイナミックアクセスコントロールの実装	96
セキュアアプリケーションの開発	97
サードパーティーのコードからのスクリプトの保護	98
簡単なシステム呼び出しの例	100
同期システム呼び出し	101
非同期システム呼び出し	102
コーディング規則	103
命名ガイドラインの順守	103
コードヒントに対応するための変数命名法	104
コードのコメント記述	105
アプリケーションの初期化	106
ローカル変数としての var の使用	106
索引	109

このマニュアルについて

Macromedia Flash Media Server 2 の世界へようこそ。このパワフルなサーバープラットフォームを使用して、Macromedia Flash でリッチなメディアアプリケーションを作成し、Flash クライアントにオーディオやビデオをストリーミングすることができます。Flash Media Server (FMS) では、複数のユーザーがテキスト、オーディオ、またはビデオを使用するリアルタイムの会話に参加できます。たとえば、FMS は、ミーティング、オンラインコミュニティインタラクション、カスタマサポート、セールスサポート、トレーニングなどに使用できます。FMS はビデオメッセージング、ビデオチャット、ビデオブログの優れた手段であると共に、ネットワークを通じてストリーミングライブデータと魅力的な Flash ビデオ体験をインターネットに配布するためのプラットフォームでもあります。FMS は Macromedia の完全なソリューションの一部で、データベース接続、ディレクトリシステム、およびプレゼンスサービスを提供します。また、広く使用される Flash Player にオーディオとビデオを配信可能な唯一のサーバーです。

対象となる読者

このマニュアルは、Flash 開発者に、開発環境の設定と、デバッグやテストを含む Flash Media Server アプリケーション作成の手順を説明しています。

このマニュアルは、Flash でのオーサリング、ActionScript、および Flash Player に習熟していることを前提として記載されています。また、サーバーサイド ActionScript の基礎である JavaScript、クライアントサーバーモデル、およびネットワーキングの概念についてもある程度の知識が必要です。

Flash Media Server ドキュメントは、Macromedia Flash がすでにインストールされていて、その使用方法をご存知であることを前提としています。

Flash Media Server ドキュメントについて

Flash Media Server ドキュメントは、Flash ドキュメント『Flash ユーザーガイド』および『ActionScript 2.0 リファレンスガイド』と連動しています。

Flash Media Server ドキュメント (英語) はすべて、PDF 形式 (Adobe Reader で表示と印刷が可能) で Flash Media Server CD に格納されており、 [Macromedia Web サイト](#) と Macromedia Flash アプリケーションの Flash ヘルプ ([ヘルプ]-[Flash ヘルプ] を選択) から参照できます。また、LiveDoc 形式の Flash Media Server ドキュメントは、 [Macromedia Web サイト](#) から入手できます。

- 『Flash Media Server インストールガイド』。Flash Media Server と Flash オーサリング拡張機能のインストール方法を説明します。インストール時にインストールされるファイルの一覧も記載されています。
- 『Flash Media Server ファーストステップガイド』。Flash Media Server の簡単な概要と、バージョン 2 の新機能について紹介し、システム要件とインストール時の指示も記載されています。Flash Media Server の CD ケースには、このマニュアル (英語) が同梱されています。
- 『メディアアプリケーション開発ガイド』 (このマニュアル)。開発環境のセットアップ方法、および Flash オーサリング環境と Flash Media Server API (アプリケーションプログラミングインタフェース) を使用してメディアアプリケーションを作成する手順について説明します。
- 『Flash Media Server 管理ガイド』。サーバーの設定・保守管理と Management Console の使い方について説明しています。
- 『Flash Media Server 2 クライアントサイド ActionScript リファレンスガイド』。クライアント側の機能を作成するための ActionScript について説明しています。
- 『サーバーサイド ActionScript リファレンスガイド』。サーバー側の機能を作成するための ActionScript について説明しています。
- 『サーバー管理 ActionScript リファレンスガイド』。拡張 ActionScript メソッドを説明しています。このメソッドを使用して、Flash Media Server Console の機能拡張、または独自の管理および監視ツールを作成できます。

Flash Media Server サポート

Flash Media Server の習得にあたっては、ほかに次のようなサポートがあります。

- [Flash Media Server サポートセンター](#) では、Flash Media Server についてのテクニカルノートと最新の情報を提供しています。
- [Flash Media Server デベロッパーセンター](#) では、Flash Media Server アプリケーションを作成する上で役に立つ情報を紹介しています。
- Flash Media Server オンラインフォーラム (www.macromedia.com/go/flashmediaserver_forum_en) では、他の Flash Media Server ユーザーと意見交換する場を提供しています。
- 最新情報、および未解決の問題一覧については、[Flash Media Server リリースノート](#)を参照照してください。

Flash Media Server に関するサードパーティーのリソースを参照できる Web サイトもあります。

- Macromedia Flash コミュニティサイト
- Macromedia Flash 関連書籍
- オブジェクト指向プログラミングの概念

表記規則

このマニュアルでは、以下の表記規則を使用します。

- `Code font` は例で使用される ActionScript ステートメントおよびリテラルテキストを示します。
- *Code font italic* は、コード内のプレースホルダー要素を示します。
たとえば、`attachAudio(source)` は、*source* パラメータの値を自分で指定する必要があることを意味します。
- *Italic* は、パスのプレースホルダー要素を示します。たとえば、`/settings/myPrinter/` は、*myPrinter* の場所を自分で指定する必要があることを意味します。
- ディレクトリパスは、前にスラッシュ (/) を付加して示します。Flash Media Server を Microsoft Windows オペレーティングシステム上で実行する場合、スラッシュをバックスラッシュに置き換えます。パスが Microsoft Windows オペレーティングシステムに固有の場合は、バックスラッシュ (\) を使用します。

ファーストステップ

この章では、Macromedia Flash Media Server 2 を使用してアプリケーション開発を開始する前に行うべき手初期手順を説明します。開発環境のセットアップ、アプリケーションの配置について解説し、サーバーに接続する簡単なアプリケーションを作成する方法についても説明します。

インストールとシステム要件

Flash Media Server のインストール手順とシステム要件に関する情報については、『Flash Media Server インストールガイド』を参照してください。

ハードウェアおよびソフトウェアのセットアップ

Flash Media Server のアプリケーションを記述するには、Macromedia Flash オーサリングソフトウェア、Flash Media Server、および最新の Flash Player をインストールしておく必要があります。オーディオやビデオをキャプチャするアプリケーションを記述する場合には、マイクやカメラをインストールする必要があります。さらに、Flash Media Server アプリケーション用のサーバーサイドスクリプトを必要とするアプリケーションでは、Flash Professional のような UTF-8 JavaScript エディタも必要です。このセクションでは、開発環境のこれらの要素について説明します。

Flash オーサリングソフトウェア Macromedia Flash がまだインストールされていない場合は、Flash のドキュメントを参照してください。

Flash Media Server ソフトウェア Flash Media Server がまだインストールされていない場合は、製品 CD に PDF 形式で格納されている、『Flash Media Server インストールガイド』(英語)を参照してください。

Flash Player 最新バージョンの Flash Player を必ず使用してください。Flash Player は、Macromedia の Web サイト www.macromedia.com/go/getflashplayer からダウンロードできます。

メモ

Flash Media Server を Linux 上で実行する場合は、メディアアプリケーションを作成するための Flash がインストールされている Windows または Macintosh のコンピュータが必要であり、また、オーサリング用のコンピュータにはオーサリング拡張機能をインストールする必要があります。また、UNIX サーバーコンピュータで管理ツールを使用するためにも、Flash Player をインストールする必要があります。

カメラとマイク カメラまたはマイクをインストールする場合は、使用するデバイスに付属している説明書の手順に従ってください。Flash Media Server と互換性があるカメラの一覧については、Macromedia の Web サイトに掲載されている、[カメラの互換性に関するドキュメント \(英文 \)](#) を参照してください。この一覧に載っていないカメラであっても Flash Media Server と互換性がある可能性もありますが、それらについて Macromedia は動作確認を行っていません。

多くのカメラにはマイクが内蔵されています。マイクを別にインストールすることもできますが、マイクとヘッドセットが組み合わされているものの方がよい結果が得られるでしょう。

デバイスのインストールが完了したら、Flash がどのカメラあるいはマイクをデフォルトとして使用するのかを指定できます。Flash アプリケーションの再生中に右クリック (Windows の場合) または Control + クリック (Macintosh の場合) でコンテキストメニューを表示し、[設定] を選択して [マイク] パネルまたは [カメラ] パネルをクリックし、ポップアップメニューの中から使用するデバイスを選択します。

JavaScript エディタの使用 サーバーサイドの ActionScript コードの記述には、どのようなエディタを使用することもできます。コードは、.asc または .js という拡張子のファイルに格納します。Flash Professional や Dreamweaver を始めとする、Web ベースのアプリケーションの設計に特化したソフトウェアを使用する場合もあるでしょう。Flash で .asc ファイルを作成するには、[ファイル]-[新規]-[AS コミュニケーションファイル] を選択します。

サーバーサイドスクリプトの中にアジア地域のいくつかの言語で使用されているダブルバイトキャラクターのような ASCII 以外のテキストを記述する場合は、UTF-8 エンコードをサポートしているエディタを使用する必要があります。クライアント間でダブルバイトキャラクタを受け渡す場合、Flash Media Server には UTF-8 でエンコードされた ASC ファイルが必須です。詳細については、[95 ページの「ダブルバイト言語のアプリケーションの記述」](#)を参照してください。

開発環境の作成

このセクションでは、Flash Media Server アプリケーションを作成する前に知っておくべきことを説明します。

サーバーが稼働していることを確認する。アプリケーションのパブリッシュとテストを行うためには、Flash Media Server が稼働している必要があります。詳細については、[20 ページの「サービスを開始する」](#)を参照してください。

サーバー URI を指定する。Flash Media Server が稼働しているコンピュータ上にある Flash オーサリング環境を使用してください。そうでない場合は、ここに記載されているすべての connect コマンドにサーバー名を付加してください。たとえば、サーバーが `myServer.myDomain.com` で稼働している場合であれば、

```
new_nc.connect("rtmp:/doc_record/room_01");
```

という記述を以下のように変更してください。

```
new_nc.connect("rtmp://myServer.myDomain.com/doc_record/room_01");
```

メモ

変更後のコードでは、rtmp: の後に必ずスラッシュを 2 つ (//) 付けてください。スラッシュ 1 つをサポートするのは、Flash Media Server が稼働しているコンピュータ上で SWF アプリケーションのサービスが実行される場合だけです。

パブリッシュ形式を指定する。SWF ファイルと HTML ファイルの両方をパブリッシュするよう、Flash を設定しておく必要があります。パブリッシュによって作成される形式を指定するには、Flash オーサリング環境で [ファイル] - [パブリッシュ設定] を選択します。

クライアントサイド ActionScript コードを記述する。特に指定がない限り、クライアントサイド ActionScript コードは、各オブジェクトに対してではなく、FLA ファイルの最初のキーフレーム上のレイヤーに追加してください。Flash Player 7 以降用にパブリッシュされたファイル内のコードでは、大文字と小文字が区別されます。

サーバーサイド ActionScript コードを記述する。サーバーサイド ActionScript コードを使用するアプリケーションでは、サーバーサイドスクリプトファイルの中にコードを記述してください。ファイル名は `main.asc` (または `registered_app_name.asc` というような名前) とし、Flash ActionScript Editor または JavaScript エディタを使用して記述します。サーバーサイドのコードでは、大文字と小文字が区別されます。

`components.asc` をロードする。どのアプリケーションにおいても、メディアコンポーネントを使用する場合には、`scriptlib` ディレクトリにある `components.asc` ファイルをロードする必要があります。このファイルをロードするには、そのアプリケーション用のサーバーサイドスクリプトファイルが作成済みでなければ作成し、正しい名前 (`main.asc` など) を付けて、先頭に次のコードを追加してください。

```
load("components.asc");
```

ヒント

このファイルは、メディアコンポーネントを使用している他のいずれかのアプリケーションのディレクトリからコピーすると便利です。ただし、正しく名前を付けるようにしてください。

プライバシーに対するユーザーの権利を認識する。誰かの画像や声を記録またはブロードキャストする場合は、事前にその人にその旨を知らせた上で承認あるいは合意を得ることが重要です。

アプリケーションの実行を監視する。サーバーを管理する権限をもっていれば、アプリケーションのテスト中にアプリケーションが生成するログメッセージや共有オブジェクトの値など、そのアプリケーションに関する詳細情報を確認できます。そのためには、Management Console (管理コンソール)を開いて、Flash Media Server に接続します。詳細については、[第5章の「アプリケーションのデバッグと監視」](#)を参照してください。

開発タスクの概要

以下のチェックリストは、Flash Media Server を使用するすべてのアプリケーションで行うべきタスクの概要を示したものです。この章の中で各タスクについて詳しく説明しますので、この章全体を飛ばさずに読んでください。

Flash Media Server アプリケーションを作成し、デプロイするには、次の作業を完了する必要があります。

1. 新しいアプリケーションの名前を決め (たとえば、my_app)、そのアプリケーションを Flash Media Server に登録するために、Flash Media Server のアプリケーションディレクトリの中に、アプリケーションの名前のディレクトリを新たに作成します。

この名前が登録アプリケーション名です。そして、このディレクトリが登録アプリケーションディレクトリです。

2. Flash を使って、FLA ファイルを作成します。このファイルには、以下のように、登録アプリケーション名を使った新しい NetConnection ステートメントと、指定するアプリケーションインスタンスがあれば URI を使って記述します。

たとえば、次のようになります。

```
my_nc = new NetConnection();  
my_nc.connect("rtmp://myDomain.com/registered_app_name");
```

このステートメントによって、クライアントが registered_app_name アプリケーションに接続されます。

3. 登録アプリケーション名でこの FLA ファイルを保存します。

FLA ファイルはどこに置いてもかまいません。この FLA ファイルは SWF ファイルを作成するためのソースファイルであり、アプリケーションの一部としてデプロイされるものではありません。

4. サーバーサイド ActionScript を含むスクリプトファイルがある場合は、そのファイルを Flash Media Server アプリケーションディレクトリの登録アプリケーションディレクトリか /scripts ディレクトリに入れます。/scripts ディレクトリは、登録アプリケーションディレクトリの中にユーザーが作成します。

サーバーサイドスクリプトファイルの名前は、main.asc または *registered_app_name.asc* のいずれでもかまいません。

5. クライアントからはアクセスでき、Flash Media Server は使用しないディレクトリにアプリケーション用の SWF ファイルをパブリッシュします。

たとえば、SWF ファイルを Web パブリッシュ用のディレクトリに置き、アプリケーションがクライアントへのサービスを提供するようにできます。または、SWF ファイルを電子メールでクライアントに送り、その SWF ファイルをどこか別のディレクトリに格納することもできます。

アプリケーションによっては、これ以外の手順が必要になる場合もありますが、ここで示した手順はアプリケーションの機能に関係なく、すべてのアプリケーションで必須となるものです。

ヒント	Flash Media Server を使用するディレクトリやファイル、およびそのアプリケーションに対して名前を付ける場合は、小文字のみを使用し、間にスペースを入れない方法をお勧めします。そうすることで、開発中に異なるプラットフォームの異なるコンピュータに移動することになっても、アプリケーションを正常に動作させることができるからです。
-----	--

この章のこれ以降の部分では、アプリケーションのセットアップに含まれる手順を説明し、Flash Media Server アプリケーションの中で使用されるオブジェクトとファイルを紹介し、最初の Flash Media Server アプリケーションを作成し、接続する方法を説明します。

アプリケーションと アプリケーションインスタンスのデプロイメント

このセクションでは、サーバーが検索するアプリケーションデータをどこに置く必要があるのかと、アプリケーションインスタンスを実行する方法とその理由を説明します。

サーバーサイドとクライアントサイドのファイルの格納

サーバーサイドアプリケーションファイルのデフォルトの場所は、Windows の場合は C:\Program files\Macromedia\Flash Media Server 2\applications、UNIX の場合は /opt/macromedia/fms/applications です。そして、これをアプリケーションディレクトリと呼びます。メディアアプリケーションを作成する場合に登録アプリケーションディレクトリを作成する必要がありますが、これはアプリケーションディレクトリの中のサブディレクトリとなります。

クライアントサイドファイル (SWF と HTML) はどこに置いてかまいませんが、一般的には Web サーバーのパブリッシュ用ディレクトリに置く場合が多いでしょう。アプリケーション用の FLA ファイルは配置までの間は SWF ファイルや HTML ファイルと一緒に残しておいてもかまいませんが、本番運用時には FLA ファイルを削除し、安全な場所に移動しておいてください。

開発環境では、各種ファイルの場所を把握しやすくするために、クライアントとサーバーのすべてのアプリケーションファイル (FLA、SWF、HTML、および ASC) を1つのサブディレクトリに置いておく方が便利かもしれません。アプリケーションのデプロイメントにあたっては、SWF ファイルと HTML ファイルをどこに置いてもかまいません。ただし、登録アプリケーションディレクトリは、サーバー上に置いておく必要があり、アプリケーションが使用する ASC ファイル、FSO ファイル、および FLV ファイルはいずれもそこに置いておく必要があります。サーバーサイドのファイル (ASC ファイル、FLV ファイル、および FSO ファイル) と FLA ソースファイルは、Web のルートディレクトリには置かないでください。Web パブリッシュ用ディレクトリに残しておくのは、SWF ファイルと HTML ファイルだけです。

いずれの場合にも、`NetConnection.connect()` コマンドを発行して接続するアプリケーションと同じ名前の登録アプリケーションディレクトリを作成する必要があります。

たとえば、`chat_App` という名前のアプリケーションがあるとすると、次のようになります。

```
NetConnection.connect("rtmp://myServer.myDomain.com/chat_App")
```

この場合、アプリケーションディレクトリに `chat_App` という名前のサブディレクトリを作成する必要があります。さらに、アプリケーションがサーバーサイドスクリプトを使用していて、それが `chat_App.asc` という名前のファイルに格納されていたとすると、`chat_App.asc` ファイルも同じディレクトリに置く必要があります。

メモ

入れるべきサーバーサイドスクリプトファイルがない場合であっても、アプリケーション名と同じ名前の登録アプリケーションディレクトリを作成する必要があります。これは、Flash Media Server が、アプリケーションによって作成されたストリームや共有オブジェクトファイルを登録アプリケーションディレクトリに格納するためです ([103 ページの「コーディング規則」](#)を参照)。また、このディレクトリが存在することで、このアプリケーションが承認され、このアプリケーションのインスタンスにユーザーが接続できることを Flash Media Server に通知することになるからです。

アプリケーションのサーバーサイドスクリプトファイルの格納

多くのアプリケーションでは、サーバーサイドスクリプトを使用します。アプリケーション用のサーバーサイドスクリプトファイルは、JavaScript テキストエディタを使って作成します。このスクリプトファイルには、main.asc または main.js、もしくは registered_app_name.asc または registered_app_name.js という名前を付けますが、registered_app_name は、登録アプリケーションディレクトリの名前であり、アプリケーションそのものの名前でもあります。そして、スクリプトファイルを登録アプリケーションディレクトリに保存します。たとえば、自分のチャットアプリケーションのディレクトリが /applications/chat_app だとすると、サーバーサイドスクリプトファイルには chat_app.asc という名前を付け、この chat_app ディレクトリに置きます。あるいは、登録アプリケーションディレクトリの中の scripts という名前のサブディレクトリ (/applications/chat_App/scripts) にサーバーサイドスクリプトを置くこともできます。

メモ	Flash Media Server に用意されているアーカイブコンパイラユーティリティ (far.exe) を使用すると、複数のサーバーサイドスクリプトファイルを1つのアーカイブファイルにパッケージ化できるため、公開が簡単になります。また、このユーティリティによってスクリプトファイルをバイナリ形式にコンパイルできるため、ロード時間も短縮されます。詳細については、 88 ページの「サーバーサイドのスクリプトファイルのアーカイブとコンパイル」 を参照してください。
----	---

アプリケーションインスタンスの使用

アプリケーションインスタンスの作成によって、アプリケーションが実行されます。クライアントがアプリケーションに接続する場合、そのクライアントは実際にはアプリケーションインスタンスに接続されます。たとえば、クライアントが chat_app という名前のアプリケーションに接続する場合は、次のようになります。

```
nc.connect("rtmp://myDomain.com/chat_app");
```

この例ではインスタンスが指定されていないため、クライアントはアプリケーションのデフォルトのインスタンスである `_deflnst_` に接続されます。

クライアントは、`NetConnection.connect` コマンドの中の `instanceName` の値で定義された特定のアプリケーションインスタンスに接続することもできます。

```
nc.connect("rtmp://myDomain.com/chat_app/instance1");
```

この例では、クライアントは `instance1` という名前のアプリケーションインスタンスに接続されます。アプリケーションインスタンスを特定の目的のために使用したい場合もあるでしょう。たとえば、トピックごとに " ルーム " を設けたチャットアプリケーションであれば、いくつかのユーザーグループがあってみんなが同じアプリケーションにアクセスするものの、グループ同士がお互いにやり取りすることはありません。そのためには、次のような方法でチャットアプリケーションに同時に複数のインスタンスを用意します。

```
my_nc.connect("rtmp://myServer.myDomain.com/chatApp/room_01")
my_nc.connect("rtmp://myServer.myDomain.com/chatApp/room_02")
```

各アプリケーションインスタンスには一意の名前を付けます。親であるアプリケーションとは異なり、インスタンスの場合はサーバー上にそのインスタンス用のディレクトリを定義する必要はありません。ただし、ストリームや共有オブジェクトのようなアプリケーションリソースは各インスタンスからは独立しており、各インスタンスはアプリケーション用に構成された streams ディレクトリと sharedObjects ディレクトリの中に格納されます。

アプリケーションインスタンスを使用するもう1つの理由は、アプリケーションが作成する記録されたストリームや共有オブジェクトの衝突を防ぐためです。たとえば、上の例では、room_01 が作成したストリームや共有オブジェクトと room_02 が作成したストリームや共有オブジェクトは、たとえ同じ chat_App というアプリケーションの中でこれら2つのインスタンスが動作するとしても、互いに区別されます。

たとえば、次のコードの中でアプリケーションは CustomerInfo という名前の共有オブジェクトを2つ作成しますが、アプリケーションの各インスタンスはそれぞれのインスタンス自身の CustomerInfo オブジェクトにのみアクセスします。さらに、次の例で示すように、session1 が使用する CustomerInfo 中のデータは session2 が使用するものとは異なります。

```
// "support" アプリケーションの 1 つのインスタンス
first_nc = new NetConnection();
first_nc.connect("myserver.mydomain.com/support/session1");
first_so = SharedObject.getRemote("CustomerInfo", first_nc.URI, false);
first_so.connect(first_nc.URI);
// "support" アプリケーションのもう 1 つのインスタンス
second_nc = new NetConnection();
second_nc.connect("myserver.mydomain.com/support/session2");
second_so = SharedObject.getRemote("CustomerInfo", second_nc.URI, false);
second_so.connect(second_nc.URI);
```

このマニュアルに掲載されている多くの例では、room_01 というインスタンス名を使用していますが、アプリケーションにおいてインスタンスに付ける名前には、そのインスタンスの意味がよくわかるような任意の文字を利用できます。

アプリケーションインスタンスに関連するいくつかの項目は、設定可能です。Application.xml ファイルでは、インスタンスがサーバーにアンロードされるまでの最大アイドル時間を設定できます。Vhost.xml ファイルでは、アプリケーションをホスティングする仮想ホストに接続できるクライアントの数と、仮想ホストがロードできるインスタンスの数を設定できます。

インスタンス名の使用に関する詳細については、『Flash Media Server 2 クライアントサイド ActionScript リファレンスガイド』の「NetStream.publish()」の項目を参照してください。

複数のアプリケーションがリモート共有オブジェクトを利用できるようにする方法についての情報は、『Flash Media Server 2 クライアントサイド ActionScript リファレンスガイド』の「SharedObject.getRemote()」の項目を参照してください。

Flash Media Server で使用されているファイル形式

Flash が作成、使用するファイル形式 (FLA、SWF、および SWD) に加えて、Flash Media Server では、次のファイル形式を使用または作成します。

ASC ファイルと JS ファイルは、サーバーサイドスクリプトファイルで、自分で書くことができますが、Flash Media Server が提供しているものもあります。サーバーサイドスクリプトは、Flash ActionScript エディタか JavaScript エディタを使用して作成し、アプリケーション用に作成したサブディレクトリである登録アプリケーションディレクトリ /applications/chat_app、またはそのスクリプトサブディレクトリ /applications/chat_app/scripts に置きます。

Flash Media Server のスクリプトライブラリは /scriptlib ディレクトリの中にあり、コンポーネントと Flash Remoting サービスのためのサーバーサイドスクリプトが含まれています。コンポーネントまたは Flash Remoting を Flash Media Server と一緒に使用する場合は、/scriptlib の該当するスクリプトをアプリケーションのサーバーサイドスクリプトファイルにインクルードまたはロードします。scriptlib ディレクトリの場所は、Application.xml ファイルの <ScriptLibPath> タグで指定します。components.asc ファイルをロードしてメディアコンポーネントを使用する方法については、[13 ページの「開発環境の作成」](#)を参照してください。

ASE ファイルは、バイナリファイルとしてコンパイルされた、サーバーサイドスクリプトファイルです。[88 ページの「サーバーサイドのスクリプトファイルのアーカイブとコンパイル」](#)を参照してください。

FAR ファイルは、必要な機能を1つのファイルにすべて備えた、サーバーサイドスクリプトファイルのアーカイブです。[88 ページの「サーバーサイドのスクリプトファイルのアーカイブとコンパイル」](#)を参照してください。

FLV ファイルと IDX ファイルは、記録されたストリームとそれに対するインデックスファイルです。サーバーはストリームを生成するときに、登録アプリケーションディレクトリの中に streams ディレクトリを生成し、アプリケーションインスタンス用のサブディレクトリ (たとえば、/applications/chat_app/streams/instance2) の中に FLV ファイルと IDX ファイルを格納します。

SOL ファイル、SOR ファイル、および FSO ファイルは、共有オブジェクトファイルです。SOL ファイルはクライアント上に常に存在します。FSO ファイルはサーバー上に常に存在します。SOR ファイルはクライアント上に常に存在し、それに一致する FSO ファイルがサーバー上に常に存在します。共有オブジェクトファイルの格納場所は、その共有オブジェクトの種類によって異なります。

Flash Media Server がストリームと共有オブジェクトファイルをどこに置くかについては、[103 ページの「コーディング規則」](#)を参照してください。

共有オブジェクトに関する詳細については、[84 ページの「共有オブジェクトファイルについて」](#)を参照してください。

サーバーへの接続

Flash Media Server のインスタンスに接続するには、まずサービスを開始してから、クライアントサイドスクリプトの中で `new NetConnection` コマンドと `NetConnection.connect()` コマンドを発行します。この作業については、このセクションの中で説明します。

メモ	アプリケーションは、RTMP (Real-Time Messaging Protocol) または RTMPS (Real-Time Messaging Protocol over SSL (Secure Socket Layer)) を使用して Flash Media Server に接続できます。
----	---

ヒント	コンポーネントを使用してサーバーに接続することはできますが、このセクションの説明にあるような ActionScript を使った接続方法をまず学んでください。
-----	---

サービスを開始する

サーバーが稼動していなければ、手動で開始します。Windows の [スタート] メニューから、[すべてのプログラム] - [Macromedia Flash Media Server 2] - [Start Flash Media Admin Server and Flash Media Server] を選択します。Windows の場合、サーバーが稼動していることを確認するために、タスクマネージャを開いて、FMSAdmin.exe と FMSMaster.exe の両方が [プロセス] タブに表示されていることを確認します。

Linux の場合、シェルウィンドウを開いて Flash Media Server をインストールしたディレクトリに変更し、root ユーザーを使って、「`fmsmgr fms server start`」と入力します。

サーバーへの接続を開く

Flash オーサリング環境で新しくファイルを開き、サーバーに接続するためのクライアントサイド ActionScript コマンドを追加します。

Flash Media Server に接続するには：

1. 新しい Flash アプリケーションで、次のコマンドを発行して接続を開く準備を開始します。

```
my_nc = new NetConnection();
```

2. `connect` コマンドを使った次のコマンドを続けます。

```
my_nc.connect(targetURI);
```

`NetConnection.connect()` (オプションのパラメータは省略) のこの基本的なシンタックスの中で、`targetURI` は、接続が確立されるときに稼動しているべき Flash Media Server 上のアプリケーションの URI (Uniform Resource Identifier) です。`targetURI` は、次のいずれかの形式で指定します (角カッコ [] で囲まれた項目はオプションです)。

```
rtmp://localhost[:port]/appName[/instanceName]
```

(localhost は、サーバーがローカルコンピュータ上で稼働していることを表します。)

```
rtmp://host[:port]/appName[/instanceName]
```

メモ	<i>targetURI</i> のシンタックス例はどれも、接続のプロトコルとして rtmp (Real-Time Messaging Protocol) を指定する必要があります。これを省略すると、Flash Player はアプリケーションサーバーに対して HTTP を使って接続を試行し、接続は失敗します。
----	--

デフォルトの仮想ホスト以外の仮想ホストに接続する場合は、*host* にその仮想ホストの名前を指定します。サーバーがローカルのコンピュータ上で稼働している場合は、URI のホスト名として "localhost" を使用できます。この方法は、アプリケーション開発中には便利です。

たとえば、次のコードでは、`new NetConnection` コンストラクタを使って新しい接続オブジェクトを作成します。次に、`my_nc.connect()` の呼び出しによって、オブジェクトがサーバーに接続されます。

```
// 新しい接続オブジェクトを作成する
my_nc = new NetConnection();

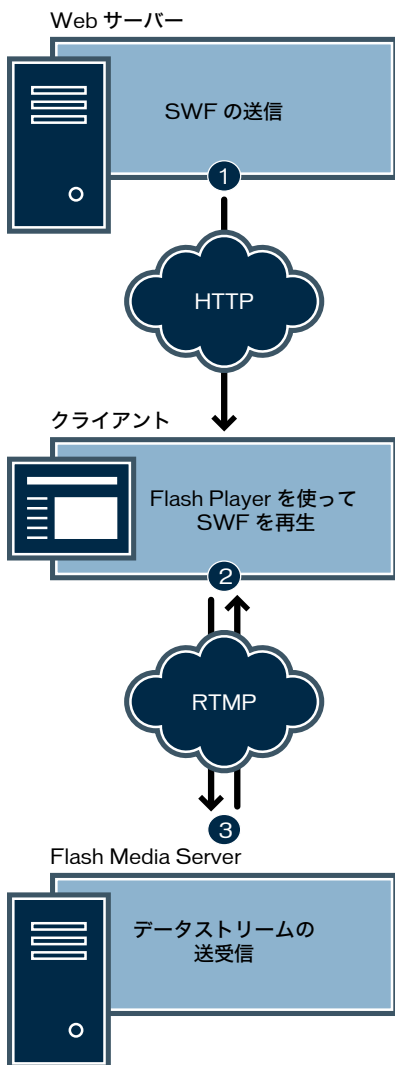
/* 仮想ホスト myServer.myDomain.com で稼働している
Flash Media Server 上の
appName という名前のアプリケーションの
/* appInstance という名前のインスタンスに接続する
my_nc.connect("rtmp://myServer.myDomain.com/appName/appInstance");
```


Flash Media Server アーキテクチャ

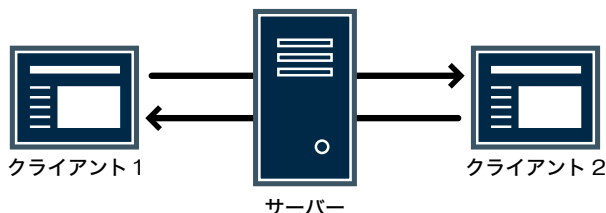
Macromedia Flash Media Server 2 プラットフォームは、サーバーと Macromedia Flash Player の 2 つで構成されています。このプラットフォーム上に構築されるアプリケーションは、Flash Player が実行するクライアント Macromedia Flash アプリケーション (SWF ファイル) と、このクライアントと通信するサーバーコンポーネントで構成されています。サーバーコンポーネントは、最低でも、Flash Media Server が稼動しているサーバー上にユーザーが作成した 1 つのアプリケーションフォルダで構成されます。このフォルダには、オプションで、メディアアプリケーションが使用するサーバーサイド ActionScript (ASC) ファイルとその他のリソースファイルを入れることができます。

Flash Media Server アーキテクチャの概要

サーバーと Flash クライアントアプリケーションは、RTMP (Real-Time Messaging Protocol) を使用した持続的な接続を介して通信します。一般的には、Web サーバーが HTTP を介して、Flash クライアントを Flash Player に配信します。次に、Flash クライアントが RTMP を使用して Flash Media Server への持続的な接続を確立し、これによって、クライアントとサーバーの間で、中断のないデータストリーム処理が可能になります。



複数のユーザー (Flash クライアント) が、Flash Media Server 上で実行中の同じアプリケーションに接続することができます。Flash Media Server は、接続されたユーザー間の通信チャンネルとして動作します。



Flash Media Server は、クライアントのための通信チャンネルを提供します。

メモ	セキュアな RTMPS 接続を作成することもできます。詳細については、クライアントサイドとサーバーサイドの ActionScript リファレンスガイドの「 <code>NetConnection.connect()</code> 」を参照してください。
----	---

ストリームと共有オブジェクトについて

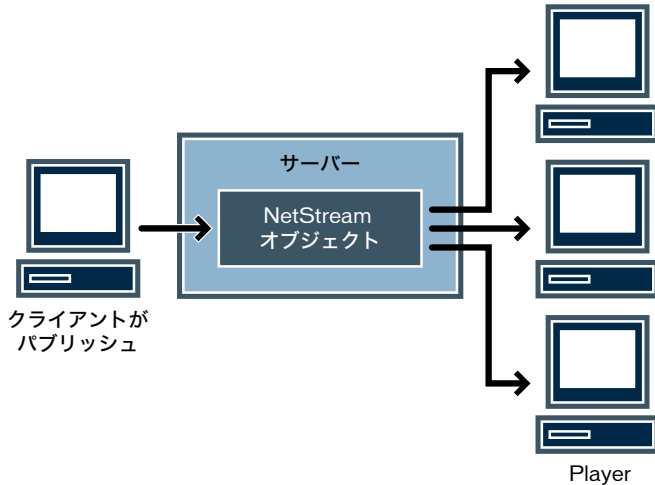
従来のクライアント / サーバーアプリケーションでは、サーバーは何らかの種類のトランザクション (クライアントの要求に応じて、サーバーがデータベースを参照したり、何らかのリソースに基づく計算を行って、結果をクライアントに返すというような処理) を実行するために使用されます。クライアントとサーバーの間の接続は、トランザクションを完了するのに必要な時間のみ持続します。

Flash Media Server を使用してトランザクションを実行できますが、Flash Media Server の主な使用方法は、インタラクションの処理、つまり、接続された複数のユーザー、すなわちクライアントの調整と、サーバーサイドデータの転送にあります。Flash Media Server は、ユーザーインタラクションの処理を単純化する 2 つの通信モデルである、ストリームと共有オブジェクトを提供します。

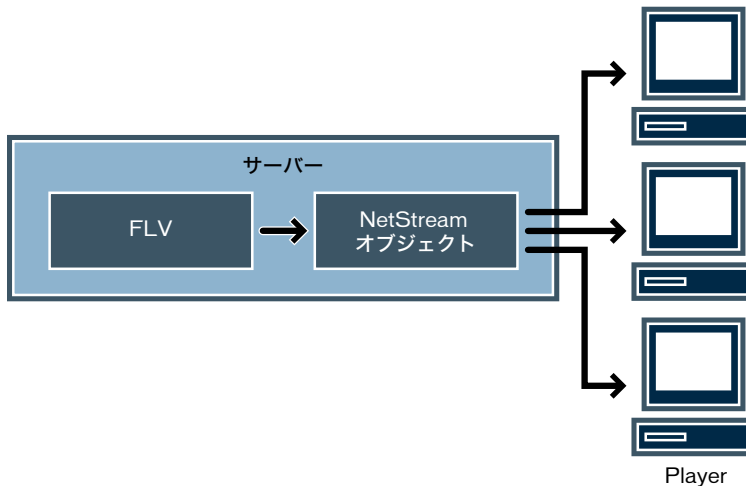
ストリームの概要

ストリームとは、同期化されたオーディオ、ビデオ、クライアントからサーバーにあるいはサーバーからクライアントに送られるデータメッセージのいずれかまたは組み合わせの、時間ベースのフローを意味します。ストリームはパブリッシュ / サブスクライブモデルを使用しているため、ストリームを使用するアプリケーションの開発はシンプルです。パブリッシュしたストリームは、リアルタイムで再生することも (例 : ビデオチャットアプリケーション)、記録しておいて後で再生することもできます。

記録したストリームは、Flash Video (FLV) 形式で保存され、ビデオだけでなく、データメッセージも記録し、保存できます。さらに、On2 や Sorenson Squeeze のようなサードパーティーのビデオエンコードユーティリティを使用するか、Flash から書き出すことで、既存のデジタルビデオファイルまたはオーディオファイルから FLV ファイルを作成することもできます。この方法で、Flash Media Server を使用して、あらかじめ記録したコンテンツをストリームすることができます。



ライブストリーム



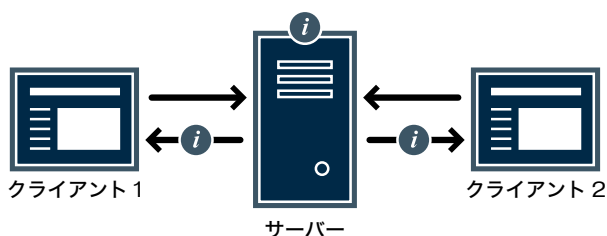
FlashVideo (FLV) ファイルとして記録されたストリームの再生

共有オブジェクトの概要

メディアアプリケーションを作成する場合に使用できる共有オブジェクトには、ローカルとリモートの2つの基本タイプがあります。

ローカル共有オブジェクトは、Flash の cookie、つまり、オフラインアクセスのために、あるいは個人の嗜好を保存するためにユーザーのコンピュータにデータを保存しておくことができるものであると考えることができます。ローカル共有オブジェクトは Flash Player の機能であり、その動作に Flash Media Server は必ずしも必要ではありません。

リモート共有オブジェクト(サーバー上にファイルとして存在)は Flash Media Server によって管理され、メッセージング、データ同期化、およびデータストレージのサービスを提供します。Flash クライアントは、リモート共有オブジェクトに接続またはサブスクライブし、共有オブジェクトに変更があった場合には更新を受け取ります。また、リモート共有オブジェクトに接続しているすべてのクライアントにメッセージを送信できます。リモート共有オブジェクトは、複数のアプリケーションセッション間で持続させることも、一時的なものにすることもできます。

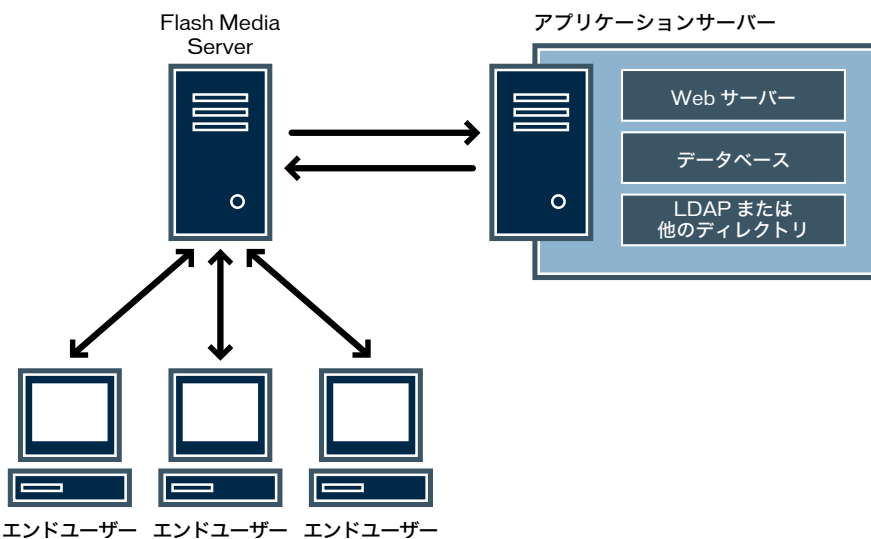


共有オブジェクトは、クライアントのためのデータの保管と同期化のサービスを提供します。

共有オブジェクトの動作に関する詳細については、[36 ページの「共有オブジェクトフローの概要」](#)、[52 ページの「SharedObject クラス」](#)、および [84 ページの「共有オブジェクトファイルについて」](#)を参照してください。

外部データソースへの接続

ストリームおよび共有オブジェクトによって提供される通信モデルに加えて、Flash Media Server は、Web サービスやリレーショナルデータベースのような外部データソースや、他の Flash Media Server アプリケーションとのやり取りを行うこともできます。たとえば、Web サービス (WebService クラスを使用) または ColdFusion アプリケーションに接続する ActionScript を記述して、氏名や電話番号のリストを取り出すことができます。そして、このクエリーの結果を共有オブジェクトの中に入れることができます。Web サービスに関する詳細については、[37 ページの「Flash Media Server のクラスについて」](#)を参照してください。



エンドユーザー エンドユーザー エンドユーザー

Flash Media Server は、外部データソースとやり取りすることができます。

Macromedia Flash Remoting は、メディアアプリケーションを J2EE アプリケーションサーバーや Microsoft Windows .NET サーバーに接続するゲートウェイです。クライアントサイドまたはサーバーサイドのスクリプトから Flash Remoting に接続でき、Flash Remoting は、NetServices と呼ばれる ActionScript ライブラリを活用して、Macromedia Flash アプリケーションをサーバーサイドゲートウェイにリンクします。NetServices API の説明が記載されている『Using Flash Remoting』は、[Macromedia の Web サイト](#)に PDF 形式で掲載されています。Flash Media Server を Flash Remoting と併せて使用する方法については、[Macromedia Flash Media Server デベロッパーセンター](#)を参照してください。

アプリケーションの作成とデプロイメントのワークフロー

Flash オーサリングツールを使用して、アプリケーションの中の、Flash Player で実行するクライアントコンポーネントを作成します。Flash SWF ファイルは、アプリケーションのユーザーインターフェイスを提供するだけでなく、Flash Media Server への接続とやり取りを管理する ActionScript も含みます。クライアントサイドの Media ActionScript に関する詳細については、『Flash Media Server 2 クライアントサイド ActionScript リファレンスガイド』を参照してください。

サーバーコンポーネントは、Flash Media Server が稼働しているコンピュータ上にユーザーが作成した、最低1つのアプリケーションフォルダで構成されます。このフォルダには、必要に応じて、サーバーサイド ActionScript (ASC) ファイルも入れることができ、これにより、共有状態の情報の制御、複数ユーザー間のリアルタイムでのやり取りを仲介するロジックの提供、外部リソースとの通信の実行が可能になります。

メモ

また、secure.asc ファイルまたは secure.js ファイルを入れることもできます。このファイルは、他のどのサーバーサイドスクリプトファイルよりも前に自動的にロードされるため、セキュアなシステム呼び出しを実現できます。詳細については、[93 ページの「\[Macromedia Flash Player 設定\] パネルの強制表示」](#)を参照してください。

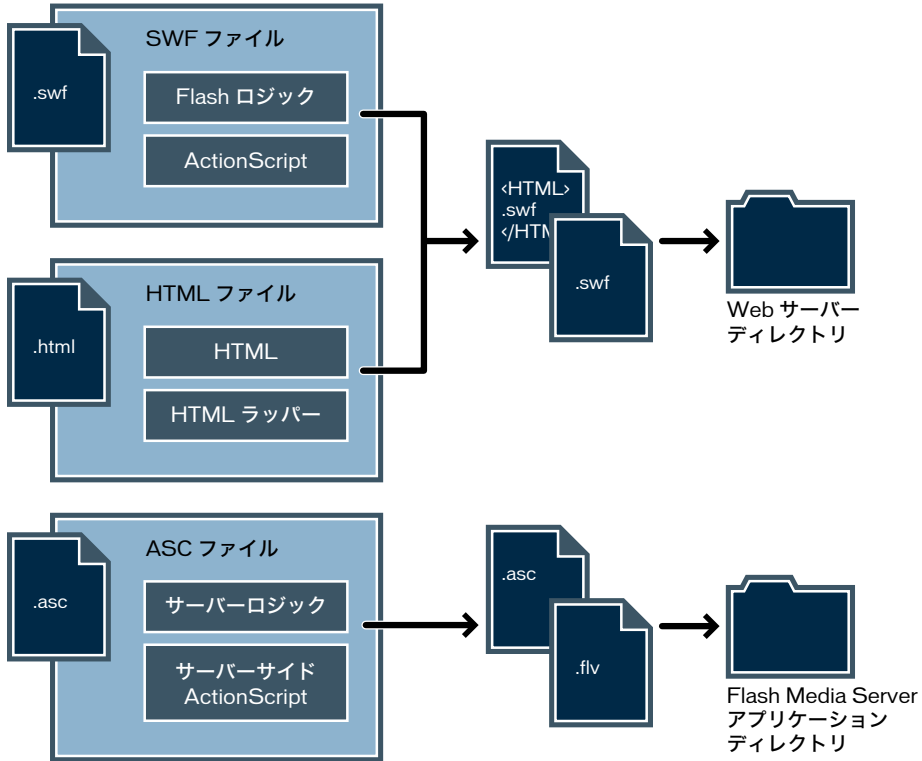
ASC ファイルの記述には、Flash Professional、テキストエディタ、または JavaScript エディタを使用できます。サーバーサイド ActionScript に関する詳細については、『サーバーサイド ActionScript リファレンスガイド』を参照してください。

アプリケーションをデプロイするには、クライアントとサーバーのファイルを正しい場所にパブリッシュする必要があります。Web にデプロイするアプリケーションの場合は、クライアントファイル、すなわち Flash アプリケーション (SWF ファイル) と、必要となる HTML ラッパーコードがあればそれを Web サーバー上のディレクトリにパブリッシュする必要があります。

メモ

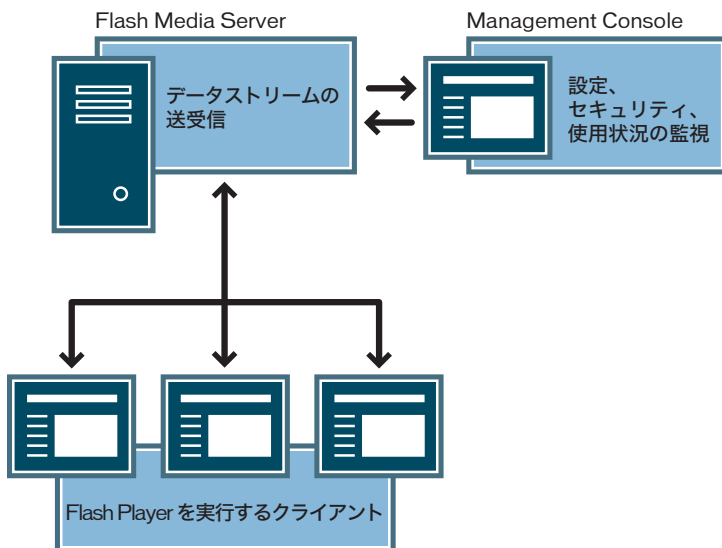
SWF ファイルはローカルで直接実行しテストすることができます。Flash Media Server と通信するために Web サーバー上に置く必要はありません。

ASC ファイル、記録ストリーム (FLV) ファイル、およびその他のサーバーサイドのリソースファイルを含むサーバーファイルは、サーバー上に定義した登録アプリケーションディレクトリにパブリッシュします。



アプリケーションディレクトリの登録に関する詳細については、第 1 章の「ファーストステップ」を参照してください。

管理者は、Management Console を使用して、Flash Media Server の設定、システムセキュリティの設定、使用状況の監視、サーバーの開始と停止、ユーザーの追加を行います。Flash Media Server の管理作業に関する詳細については、『Flash Media Server 管理ガイド』を参照してください。



アプリケーションフローの概要

このセクションでは、Flash クライアントをサーバーに接続する方法、共有オブジェクトにアクセスする方法、およびリモートメソッドの呼び出し方法をさらに詳細に説明します。

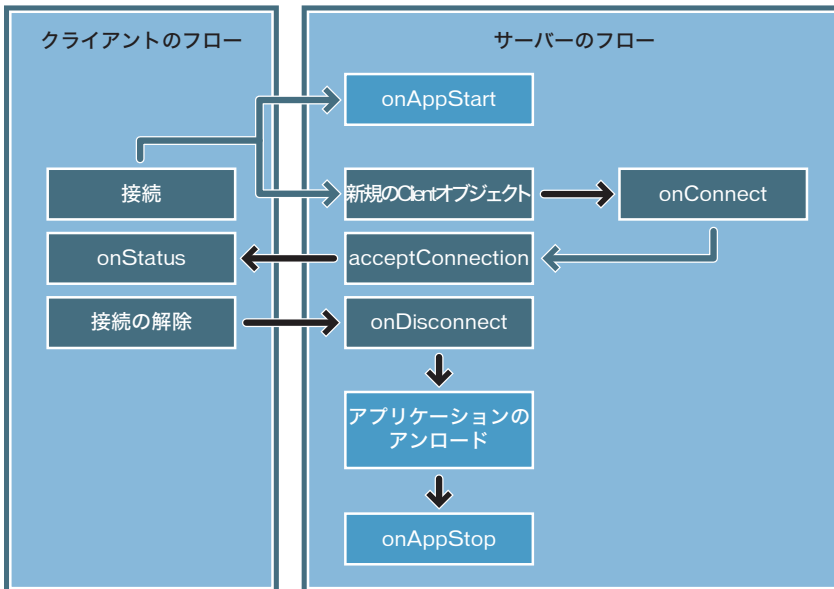
ユーザーが Flash SWF ファイルを実行し、その SWF ファイルがサーバーに接続すると、サーバーはアプリケーションをロードし、まだ稼動していなければ、アプリケーションインスタンスを作成します。サーバーは、接続を受け付け、サーバー上のそのクライアントアプリケーションを表す新しいサーバーサイド Client オブジェクトを作成し、ユーザーが指定したサーバーサイドスクリプトを実行します。サーバーサイド Client オブジェクトはサーバーへの接続に使用されたクライアントサイド NetConnection オブジェクトに対応しています。そして、この Client オブジェクトには、サーバーサイドスクリプトからのみアクセスできます。

クライアントは、ストリームの開始、オブジェクトの共有といった処理を実行します。一連のこの処理については、次のセクションでさらに詳しく説明します。

接続フロー

クライアントがサーバーに接続すると、アプリケーションインスタンスのロード時にサーバーは `onAppStart` を呼び出します (アプリケーションが実行中で、`onAppStart` が呼び出されていない場合)。次に、新しく作成された Client オブジェクトでサーバーサイド `onConnect` ハンドラが呼び出されます。このメソッドのロジックによって、接続を許可するか拒否するかが決定されます。クライアントサイドでは、`onStatus` ハンドラが呼び出され、接続が許可されたのか拒否されたのかが通知されます。クライアントが接続を閉じると、サーバーサイドの `onDisconnect` ハンドラが呼び出されます。アプリケーションがアンロードされると、`onAppStop` が呼び出されます。

`onConnect` ハンドラで接続を受諾するには、`true` を返すか、`application.acceptConnection()` を呼び出します。接続を拒否するには、`false` を返すか、`application.rejectConnection()` を呼び出します。`onConnect` ハンドラで `true` または `false` が返されないと、クライアントは、`acceptConnection()` または `rejectConnection()` が呼び出されるまでの間、保留のままになります。



接続フロー

メモ	通信セッションを開始できるのは、クライアントアプリケーションだけです。ステータスメッセージの送受信、ストリームの開閉、データの保存と共有、およびネットワーク接続の終了は、クライアントとサーバーのどちらからでも実行できます。
----	---

リモートメソッドの呼び出し

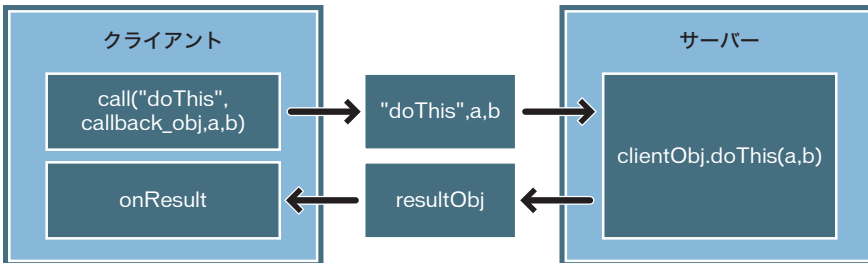
接続が成功すれば、クライアントは、サーバーコンポーネントによって定義されたリモートメソッドを呼び出せます。また、サーバーアプリケーションも、リモートで、Flash クライアントアプリケーションに定義されている ActionScript メソッドを呼び出すことができます。

クライアントからのサーバーメソッドの呼び出し

サーバーアプリケーションに接続する各 Flash アプリケーションは、サーバーサイド Client クラスのインスタンスである Client オブジェクトによって表されます。さらに、各 Client オブジェクトには、そのオブジェクトに関連し、リモートでアクセスされるメソッドを開発者が定義し、入れることができます。(prototype プロパティを使用して、すべてのクライアントが利用できるリモートメソッドを作成することもできます。その例については、『サーバーサイド ActionScript リファレンスガイド』の Client クラスの項目を参照してください。

次に、Flash アプリケーションは、`NetConnection.call()` メソッドを使用して、リモートに定義されたメソッドを呼び出し、オプションでコールバックオブジェクトを指定して、サーバーから返される結果を処理します。サーバーでは、クライアント呼び出しに対応するメソッドが呼び出され、結果がクライアントに返されます。

次の図は、サーバーが定義したメソッドをクライアントから呼び出している例です。この図では、クライアントが、リモートメソッド `doThis` を `nc` という名前の `NetConnection` オブジェクトを介して呼び出しています。`doThis` メソッドには、この例では `clientObj` という名前の Client オブジェクトに関連付けられていて、このクライアントを表しています。



クライアントからサーバーへのリモートメソッド呼び出しのフローおよびクライアントへの結果の戻り

次の表は、Flash Media Server のクライアントステータスメッセージの一覧です。

Code プロパティ	Level プロパティ	説明
NetStream.Clear.Success	Status	記録ストリームが正常に削除されました。
NetStream.Clear.Failed	Error	記録ストリームの削除に失敗しました。
NetStream.Publish.Start	Status	パブリッシュの試行が成功しました。
NetStream.Publish.BadName	Error	すでに別のユーザーによってパブリッシュされているストリームをパブリッシュしようとして失敗しました。
NetStream.Failed	Error	Stream メソッドを使用する試行が失敗しました。
NetStream.Unpublish.Success	Status	パブリッシュ解除の試行が成功しました。
NetStream.Record.Start	Status	記録が開始されました。
NetStream.Record.NoAccess	Error	読み取り専用ストリームを記録しようとして失敗しました。
NetStream.Record.Stop	Status	記録が停止しました。
NetStream.Record.Failed	Error	ストリームを記録する試行が失敗しました。
NetStream.Play.Start	Status	再生が開始されました。
NetStream.Play.StreamNotFound	Error	存在しないストリームを再生しようとして失敗しました。
NetStream.Play.Stop	Status	再生が停止しました。
NetStream.Play.Failed	Error	ストリームを再生する試行が失敗しました。
NetStream.Play.Reset	Status	再生リストがリセットされました。
NetStream.Play.PublishNotify	Status	ストリームへの初期パブリッシュが正常に行われました。このメッセージはすべてのサブスクライバ側に送信されます。
NetStream.Play.UnpublishNotify	Status	ストリームからのパブリッシュ解除が正常に行われました。このメッセージはすべてのサブスクライバ側に送信されます。
NetStream.Play.InsufficientBW	Status	データが通常速度より低速で再生されています。

情報オブジェクト NetStream.Failed および NetStream.Play.Failed には、description プロパティもあります。このプロパティは、エラーの原因を具体的に説明する文字列です。

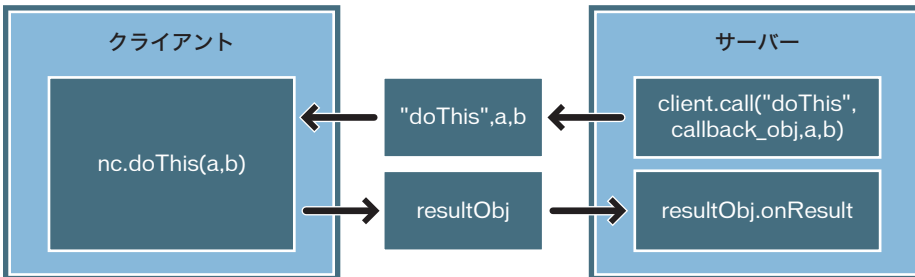
情報オブジェクト NetStream.Play.Start および NetStream.Play.Failed には、description プロパティもあります。このプロパティは、再生されるストリームの名前を表す文字列です。複数のストリームを再生する場合に利用すると便利です。再生リスト上のエレメントが次のエレメントに切り替わるたびに、details プロパティにストリーム名が表示されます。

サーバーメソッドのリモートでの呼び出しに関する詳細については、『サーバーサイド ActionScript リファレンスガイド』の Client."commandName" の項目を参照してください。

サーバーからのクライアントメソッドの呼び出し

クライアントが定義したメソッドをサーバーから呼び出すためのプロセスは、その逆の場合と似ています。クライアントでは、ユーザーが定義したメソッドを、サーバーへの接続に使用する NetConnection オブジェクトに結合できます。次に、サーバーは、`Client.call()` メソッドを使用して、そのメソッドをリモートで呼び出せます。

次の図は、クライアントメソッドをサーバーから呼び出している例です。この図では、クライアントサイドメソッド `doThis` は、`nc` という名前の NetConnection オブジェクトに関連付けられています。サーバーは、`Client.call()` メソッドを使用して `doThis` メソッドを呼び出します。サーバーサイドスクリプトは、オプションで、結果を処理するオブジェクトを指定できます。再び、クライアントが結果を返し、クライアントに送られたコールバックオブジェクト上でサーバーの `onResult` ハンドラが呼び出されます。



サーバーからクライアントへの呼び出しのフローおよびサーバーへの結果の戻り

次の表は、Flash Media Server ステータスメッセージの一覧です。

Code プロパティ	Level プロパティ	説明
<code>NetStream.Clear.Success</code>	Status	記録ストリームが正常に削除されました。
<code>NetStream.Clear.Failed</code>	Error	記録ストリームの削除に失敗しました。
<code>NetStream.Publish.Start</code>	Status	パブリッシュの試行が成功しました。
<code>NetStream.Publish.BadName</code>	Error	すでに別のユーザーによってパブリッシュされているストリームをパブリッシュしようとして失敗しました。
<code>NetStream.Failed</code>	Error	Stream メソッドを使用する試行が失敗しました。
<code>NetStream.Unpublish.Success</code>	Status	パブリッシュ解除の試行が成功しました。
<code>NetStream.Record.Start</code>	Status	記録が開始されました。
<code>NetStream.Record.NoAccess</code>	Error	読み取り専用ストリームを記録しようとして失敗しました。
<code>NetStream.Record.Stop</code>	Status	記録が停止しました。
<code>NetStream.Record.Failed</code>	Error	ストリームを記録する試行が失敗しました。

Code プロパティ	Level プロパティ	説明
<code>NetStream.Play.Start</code>	Status	再生が開始されました。
<code>NetStream.Play.StreamNotFound</code>	Error	存在しないストリームを再生しようとしてしました。
<code>NetStream.Play.Stop</code>	Status	再生が停止しました。
<code>NetStream.Play.Failed</code>	Error	ストリームを再生する試行が失敗しました。
<code>NetStream.Play.Reset</code>	Status	再生リストがリセットされました。
<code>NetStream.Play.PublishNotify</code>	Status	ストリームへの初期パブリッシュが正常に行われました。このメッセージはすべてのサブスクライブ側に送信されます。
<code>NetStream.Play.UnpublishNotify</code>	Status	ストリームからのパブリッシュ解除が正常に行われました。このメッセージはすべてのサブスクライブ側に送信されます。

情報オブジェクト `NetStream.Failed` および `NetStream.Play.Failed` には、`description` プロパティもあります。このプロパティは、エラーの原因を具体的に説明する文字列です。

情報オブジェクト `NetStream.Play.Start` および `NetStream.Play.Failed` には、`description` プロパティもあります。このプロパティは、再生されるストリームの名前を表す文字列です。複数のストリームを再生する場合に利用すると便利です。再生リスト上のエレメントが次のエレメントに切り替わるたびに、`details` プロパティにストリーム名が表示されます。

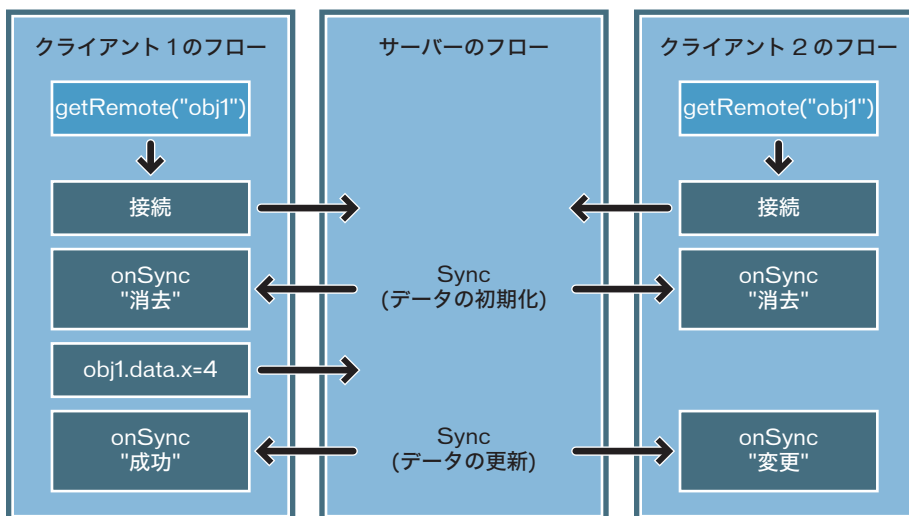
共有オブジェクトフローの概要

共有オブジェクトは、複数ユーザー間でのデータ共有のための開発者の作業を簡素化します。Flash クライアントアプリケーションは、`SharedObject.getRemote()` コマンドの発行によってリモート共有オブジェクトをサブスクライブし、それによって、リモート共有オブジェクトの参照が返されます。次に、クライアントが `SharedObject.connect()` コマンドを発行することでリモート共有オブジェクトをサーバーに接続します。 `SharedObject.connect()` の呼び出しによって、`SharedObject` が `NetConnection` オブジェクトに関連付けられ、それによって、サーバーとの送受信にこの接続を使用できるようになるため、共有オブジェクトがサーバーとの間で更新を送受信できるようになります。

クライアントが共有オブジェクトに接続されると、サーバーは共有オブジェクトへの同期メッセージを送信し、クライアント上に定義された `SharedObject.onSync` イベントハンドラがこのメッセージを処理します。クライアント、サーバー、または他のムービーインスタンスによって共有オブジェクトが変更されると、サーバーは再びその共有オブジェクトへの同期メッセージを送信します。

各同期メッセージには、共有オブジェクトに対する変更の特性を表すコードが含まれています。このコードは状況によって異なります。たとえば、クライアントが最初に共有オブジェクトに接続すると、同期メッセージのこのコードの値は `clear` となり、また、値が `success` であれば、クライアントによる共有オブジェクトの内容の変更が成功したことを表します。このコードとその説明の全一覧については、『Flash Media Server 2 クライアントサイド ActionScript リファレンスガイド』の「SharedObject.onSync」の項目を参照してください。

次の図は、クライアント1とクライアント2という2つのクライアントによる、同じ共有オブジェクト `obj1` への接続のシナリオを示したものです。各クライアントが最初に共有オブジェクトに接続すると、`clear` 同期メッセージがサーバーから返されます。次に、クライアント1が `obj1` のデータベースである `x` を変更して `4` を設定します。すると、クライアント1は、共有オブジェクトの変更が成功したことを示す `success` 同期メッセージを受け取り、クライアント2は、`obj1` 共有オブジェクトが変更されたことを示す `change` 同期メッセージを受け取ります。



共有オブジェクトフロー

メディアクラスの使用

Macromedia Flash Media Server には、クライアントサイド API とサーバーサイド API という 2 つの API (アプリケーションプログラムインターフェイス) が提供されています。この章では、クライアントサイドとサーバーサイドのクラスについて説明し、それらが対になってどのように通信中にオブジェクトが作成されるのかを解説します。また、共有オブジェクトについても取り上げ、ユーザー間、アプリケーションインスタンス間、およびアプリケーション間で共有するアプリケーションやユーザーの情報を格納することについても説明します。

Flash Media Server の中核となるこれらのクラスの説明に続いて、いくつかのクラスの実装に関する重要な詳細情報として、Application、Camera、Microphone、NetStream、Stream、System、Video といったオブジェクトの最適化についての推奨事項と、Client オブジェクトの保護と NetConnection オブジェクトのセキュリティ設定などについても解説します。

Flash Media Server のクラスについて

クライアントサイド API には、Camera、Microphone、NetConnection、NetStream、SharedObject、Video というクラスがあります。これらのクラスの使用に関する詳細については、『Flash Media Server 2 クライアントサイド ActionScript リファレンスガイド』を参照してください。サーバーサイド API には、Application、Client、File、LoadVars、NetConnection、SharedObject、Stream、WebService、XML、XMLSocket、XMLStreams というクラスがあります。これらのクラスの使用に関する詳細については、『サーバーサイド ActionScript リファレンスガイド』を参照してください。

NetConnection などのように、両方の API でクラスの名前が同じものもいくつかありますが、クライアントサイドのクラスとサーバーサイドのクラスでは、名前が同じであっても機能は異なります。また、クライアントサイドのいくつかのクラスには、それに対応するサーバーサイドのクラスがあるものもありますが、ないものもあります。次のセクションでは、クライアントサイドとサーバーサイドの各クラスについて簡単に解説し、クライアントとサーバーが通信を行うにはどのような組み合わせで使用すればいいのかを説明します。

クライアントサイドクラスについて

これらのクラスは、クライアントサイド ActionScript のみで使用されます。これらのクラスに関する詳細については、『Flash Media Server 2 クライアントサイド ActionScript リファレンスガイド』を参照してください。

Camera クラス クライアントサイドの Camera クラスにより、Macromedia Flash Player を実行中の任意のコンピュータに接続されているビデオカメラからビデオをキャプチャすることができます。Flash Media Server と併せて使用すると、キャプチャするビデオの送信と表示ができ、さらにオプションで記録することもできます。これらの機能を使用することで、テレビ会議やインスタントメッセージングなどのメディアアプリケーションを開発することができます。

Microphone クラス クライアントサイドの Microphone クラスによって、Flash Player を実行中の任意のコンピュータに接続されているマイクからオーディオをキャプチャすることができます。Flash Media Server と併せて使用すると、キャプチャするオーディオの送信と再生ができ、さらにオプションで記録することもできます。これらの機能を使用することで、オーディオによるインスタントメッセージングや後で他者が再生できるプレゼンテーションの録音などのメディアアプリケーションを開発することができます。

Microphone オブジェクトはサーバーがなくても使用できます。たとえば、ローカルシステムでマイクからスピーカーにサウンドを送信する場合に使用します。

NetConnection クラス クライアントサイドの NetConnection クラスによって、Flash クライアントが Flash Media Server 上の TCP ソケットを開き、RTMP (Real-Time Messaging Protocol) を使用してデータを継続的に交換することができます。また、NetConnection クラスを使用するとアプリケーションサーバーに接続することもできますが、この章では、Flash Media Server との通信のための NetConnection クラスの使用に的を絞ります。

NetStream クラス クライアントサイドの NetStream クラスは、クライアントサイドの NetConnection オブジェクトによって使用できるようになった接続を通して、Flash Player と Flash Media Server の間の一方通行のストリーミング接続を開きます。NetStream オブジェクトは、NetConnection 内の 1 つのチャンネルのようなもので、このチャンネルでは、NetStream.publish() メソッドを使用してオーディオ、ビデオ、およびデータをパブリッシュすること、あるいは、NetStream.play() メソッドを使用してストリームをサブスクライブし、データを受け取ることができます。ライブ (リアルタイム) のデータのパブリッシュまたは再生と、以前に記録されたデータの再生が可能です。複数のクライアントが 1 つのストリームを再生することが可能ですが、パブリッシュをできるのは 1 度に 1 つのクライアントだけです。サーバー上での記録されたストリームの作成と格納に関する情報については、[103 ページの「コーディング規則」](#)を参照してください。

SharedObject クラス クライアントサイドのローカルの共有オブジェクトを使用すると、ユーザーのコンピュータ上にゲームのハイスコアのような情報を格納できるため、同じアプリケーション、あるいはそのコンピュータ上で実行中の他のアプリケーションを後で使用することができます。ローカルの共有オブジェクトに関する詳細については、[43 ページの「共有オブジェクトの概要」](#)を参照してください。

Video クラス クライアントサイドの Video クラスによって、ストリーミングビデオをステージ上に表示することができます。再生またはキャプチャされるビデオフィードを NetStream.play() コマンドまたは Camera.get() コマンドを使用して表示するには、ステージ上に Video オブジェクトを置き、Video.attachVideo() メソッドを使用してフィードをオブジェクトに割り当てます。

Flash で、ステージ上に Video オブジェクトを置くには：

1. Flash で、[ライブラリ] パネルが開いていなければ、[ウィンドウ]-[ライブラリ] を選択して表示します。
2. [ライブラリ] パネルの右上部にあるオプションメニューをクリックして [新規ビデオ] を選択し、埋め込み Video オブジェクトをライブラリに追加します。
3. Video オブジェクトをステージにドラッグし、プロパティインスペクタを使用して一意の名前を設定します。

サーバーサイドクラスについて

これらのクラスは、サーバーサイド ActionScript のみで使用されます。これらのクラスの使用に関する詳細については、『サーバーサイド ActionScript リファレンスガイド』を参照してください。

Application クラス サーバーサイドの Application クラスによって、Flash Media Server のアプリケーションインスタンスに関する情報が含まれるオブジェクトを作成することができます。このオブジェクトは、そのアプリケーションインスタンスがアンロードされるまで持続します。Application クラスを使用すると、クライアント接続試行の受諾および拒否、クラスやプロキシの登録および登録解除、アプリケーションの開始時または終了時やクライアントの接続時または接続解除時に呼び出される関数の作成が可能です。

Client クラス サーバーサイドの Client クラスによって、Flash Media Server のアプリケーションインスタンスへの各ユーザーの接続を表すオブジェクトを作成することができます。Client オブジェクトは、クライアントサイド NetConnection.call() コマンドから送信されたメッセージを受け取り、クライアントサイドの NetConnection オブジェクトのメソッドを呼び出すことができます。Client オブジェクトのプロパティを使用することで、各クライアントのバージョン、プラットフォーム、および IP アドレスを判断することができます。また、Client クラスを使用することで、Stream オブジェクトや SharedObject オブジェクトのような多様なアプリケーションリソースに対して読み取りと書き込みの権限を個別に設定することもできます。詳細については、[96 ページの「ダイナミックアクセスコントロールの実装」](#)を参照してください。

File クラス サーバーサイドの File クラスによって、アプリケーションによるサーバーのファイルシステムへの書き込みが可能になります。

LoadVars クラス サーバーサイドの LoadVars クラスによって、リモートまたはローカルにあるサーバーサイドスクリプトに変数をロードすることができます。

NetConnection クラス サーバーサイドの NetConnection クラスでは、Flash Media Server アプリケーションインスタンスとアプリケーションサーバー間、別の Flash Media Server 間、または同一サーバー上の別の Flash Media Server アプリケーションインスタンス間の双方向の接続を作成することができます。さらには、サーバーサイドの NetConnection オブジェクトによって、たとえば、アプリケーションサーバーから気象情報を取得する、あるいは、アプリケーションロードを他の Flash Media Servers やアプリケーションインスタンスと共有するというような、より強力なアプリケーションを作成することができます。

NetConnection オブジェクトを使用すると、アプリケーションサーバーに接続し、標準プロトコル (HTTP など) を使用してサーバー同士がやり取りすることや、他の Flash Media Server に接続し、RTMP (Real-Time Messaging Protocol) を使用してオーディオ、ビデオ、およびデータを共有することができます。

Macromedia Flash Remoting を Flash Media Server と併せて使用すれば、Macromedia ColdFusion、.NET、および J2EE のサーバーとの通信が可能です。詳細については、Macromedia の Web サイト www.macromedia.com/jp/software/flashremoting/ に掲載されている、Flash Remoting の情報を参照してください。

SharedObject クラス サーバーサイドの共有オブジェクトによって、クライアントサイドの共有オブジェクトおよび他の Flash Media Server 上のオブジェクトと通信できます。サーバーサイドの共有オブジェクトに関する詳細については、[43 ページの「共有オブジェクトの概要」](#)を参照してください。

Stream クラス サーバーサイドの Stream クラスによって、Flash Media Server アプリケーション内の各ストリームを処理することができます。Flash Media Server は、クライアントサイドスクリプトで NetStream.play() メソッドまたは NetStream.publish() メソッドが呼び出されると、Stream オブジェクトを自動的に作成します。Stream.get() メソッドを呼び出すことで、サーバーサイド ActionScript でストリームを作成することもできます。ユーザーは複数のストリームに同時にアクセスでき、複数の Stream オブジェクトが同時にアクティブになります。

WebService クラス サーバーサイドの Webservice クラスによって、WSDL/SOAP Web サービスを作成し、アクセスすることができます。WebService クラスと併せて使用できるサーバーサイドクラスは、Log クラス、SOAPFault クラス、SOAPCall クラスです。

XML クラス サーバーサイドの XML クラスによって、XML ドキュメントツリーをロード、削除、送信、作成、操作することができます。

XMLSocket クラス サーバーサイドの XMLSocket クラスはクライアントソケットを実装しており、Flash Player はこのソケットを使用して、IP アドレスまたはドメイン名で識別されるサーバーコンピュータと通信することができます。

XMLStreams クラス サーバーサイドの XMLStreams クラスは、XMLSocket クラスのバリエーションで、メソッドとプロパティは同じですが、データを断片的に送受信します。

クライアントとサーバーの通信

次の " オブジェクトのペア " は、クライアントサイドとサーバーサイドのオブジェクトの間で確立できる接続を示します。たとえば、クライアントサイドの NetConnection オブジェクトがサーバーに接続すると、サーバーサイドの Client オブジェクトが作成され、それによって、この Client オブジェクトはそれに対応する NetConnection オブジェクトのメソッドを呼び出すことができます。

次の表は、クライアントサイドとサーバーサイドでオブジェクトが互いに関連付けられているものをまとめたものです。

クライアントサイドオブジェクト	対応するサーバーサイドオブジェクト
my_nc (NetConnection オブジェクト)	my_client (Client オブジェクトまたは application.clients オブジェクト)
my_ns (NetStream オブジェクト)	my_server_stream (Stream オブジェクト)
my_so (リモート共有オブジェクト)	my_server_so (サーバーサイド共有オブジェクト)

次の表では、左側のクライアントサイド呼び出しによって、右側のサーバーサイド呼び出しが開始されます。

クライアントサイド呼び出し	サーバーサイド呼び出し
my_nc.connect	application.onConnect
my_nc.close	application.onDisconnect
my_nc.call("doThing", myCallbackFcn, 1, "foo")	my_client.doThing(1, "foo")
my_so.send("doThing", 1, "foo")	my_server_so.doThing(1, "foo")

次の表では、左側のサーバーサイド呼び出しによって、クライアントサイド呼び出しが開始されます。

サーバーサイド呼び出し	クライアントサイド呼び出し
my_client.call("doThing", myCallbackFcn, 1, "foo")	my_nc.doThing(1, "foo")
my_server_stream.send("doThing", 1, "foo")	my_ns.doThing(1, "foo")
my_server_so.send("doThing", 1, "foo")	my_so.doThing(1, "foo")

共有オブジェクトの概要

共有オブジェクトとは、複数の異なるクライアント間、Flash Media Server 上で稼動しているアプリケーションの異なるインスタンス間、および複数の Flash Media Server 上で稼動しているアプリケーション間でデータを共有するための手段です。Flash Media Server は、3 種類の共有オブジェクトをサポートしています。ローカル、リモート、およびサーバーサイドこれらのオブジェクトについて、以下で簡単に説明します。共有オブジェクトの処理の詳細については、[84 ページの「共有オブジェクトファイルについて」](#)を参照してください。

ローカル共有オブジェクトについて

共有オブジェクトの使用方法の1つに、ローカルでのデータの保存と取得があります。データはエンドユーザーのコンピュータに保存され、1つ以上の Flash アプリケーションからアクセスされます。ローカル共有オブジェクトは、非永続的に、つまり、アプリケーションの実行中だけ利用可能にすることもできます。

ローカル共有オブジェクトは Flash Media Server への接続を必要としないので、このマニュアルでは詳しく説明しません。永続的なローカル共有オブジェクトの保存場所については、[84 ページの「共有オブジェクトファイルについて」](#)を参照してください。ローカル共有オブジェクトに関する詳細については、『Flash Media Server 2 クライアントサイド ActionScript リファレンスガイド』の「`SharedObject.getLocal()`」の項目を参照してください。

リモート共有オブジェクトについて

クライアントサイド ActionScript を使用すると、同じあるいは異なるクライアント上で実行中の他の Flash Media Server アプリケーションインスタンスが利用できる共有オブジェクトを作成し、参照することができます。ローカル共有オブジェクトと同様、リモート共有オブジェクトもコンピュータ上に永続的に存在させることができます。ただし、リモート共有オブジェクトはサーバー上で永続的に存在するため、共有オブジェクトに接続するすべてのユーザーが同じ情報にアクセスできるようになります。

たとえば、サーバー上に複数のユーザーが使用する共有のアドレス帳などを保存することができます。あるクライアントが、サーバー上のデータを変更すると、現在そのリモート共有オブジェクトに接続しているほかのクライアント、およびそれ以降に接続する全クライアントに伝わります。データがローカルにも保存されていて、サーバーに接続しない状態で変更された場合は、そのクライアントが次回リモート共有オブジェクトに接続したときに、同期化されます。

共有オブジェクトの中で、Flash Media Server アプリケーションで最も使用頻度が高いのは、リモート共有オブジェクトです。永続リモート共有オブジェクトのデータの保存場所については、[84 ページの「共有オブジェクトファイルについて」](#)を参照してください。リモート共有オブジェクトに関する詳細については、『Flash Media Server 2 クライアントサイド ActionScript リファレンスガイド』の「`SharedObject.getRemote()`」の項目を参照してください。

プロキシ共有オブジェクトについて

プロキシ共有オブジェクトは、リモート共有オブジェクトの1つで、クライアントとサーバーアプリケーションの間で共有されるのではなく、2つの異なる Flash Media Server アプリケーション間、または同じアプリケーションのインスタンス間で共有されます。たとえば、同じチャットアプリケーションに `/chat_01` と `/chat_02` という2つのインスタンスがあるとします。サーバー上で `/chat_01` アプリケーションインスタンスが `/chat_02` に定義されている共有オブジェクトに接続すると、その共有オブジェクトの中の情報を、`/chat_01` に定義されている場合と同じように使用できます。

プロキシ共有オブジェクトの詳細は、『サーバーサイド ActionScript リファレンスガイド』の「`SharedObject.get()`」の項目を参照してください。

Application クラスについて

Application クラスを使用すると、クライアント接続試行の受諾および拒否、クラスやプロキシの登録および登録解除、アプリケーションの開始時または終了時やクライアントの接続時または接続解除時に呼び出される関数の作成が可能です。このセクションでは、`application.OnConnect` 関数および `application.OnDisconnect` 関数をセットアップする方法と理由や、メディアコンポーネントを使用するアプリケーションで `Application.onConnectAccept` ハンドラおよび `Application.onConnectReject` ハンドラを使用する方法を紹介します。

Application クラスの詳細は、『サーバーサイド ActionScript リファレンスガイド』の「Application class」の項目を参照してください。

application.onConnect ハンドラの使用

クライアントが `NetConnection.connect()` メソッドを呼び出すと、サーバー上で `application.onConnect` イベントハンドラが呼び出されます。onConnect ハンドラには、接続しようとするクライアントを表すサーバーサイド Client オブジェクトへの参照が自動的に渡され、開発者が定義したパラメータがあれば、`NetConnection.onConnect` ハンドラに渡されます。

`application.onConnect` ハンドラに割り当てられている関数の中にメソッドを設定する方法は、ユーザーのログイン情報によって異なるメソッドを設定することが必要な場合に効率的です。ただし、すべてのクライアントが利用できるメソッドを設定する場合には、次の例のように、Client オブジェクトの ActionScript 1.0 prototype プロパティを使用できます。

```
// ユーザーは接続済み
application.onConnect = function(newClient, userName)
{
    if (userName == "admin")
    {
        newClient.adminFunc= function(param)
        {
            // admin のみに有効なコード
            newClient.myAdminProperty = param;
        }
    } else
    {
        // 多くの場合に有効なコード
    }

    // ログインを許可
    application.acceptConnection(newClient);
}

// この部分は別ファイルに入れることも可能

// 各クライアント (admin を含む) にこの関数が必要になる
// ("prototype" の動作による)
Client.prototype.commonFunction = function (myColor)
{
    // 何らかのコード

    // クライアントを参照するために newClient の代わりに使用
    this.color = myColor;
}
```

また、接続しているクライアントオブジェクトに `call()` メソッドを使用することが必要な場合には、さらにコマンドを発行する前に `application.acceptConnection()` を呼び出して、クライアントが接続されていることを確認しておく必要があります。この処理をセットアップするコードは次のとおりです。

```
application.onConnect = function(clientObj,name,passwd)
{
  // まず、接続を受け付ける
  application.acceptConnection(clientObj);
  // クライアントがアプリケーションインスタンスに登録されれば
  // "call" メソッドを使用できる
  clientObj.call("onWelcome", "You are now connected!!!");
  return;
  // acceptConnection() または
  // rejectConnection() を onConnect で呼び出すと、戻り値は無視される
}
```

ヒント

コンポーネントを使用する場合は、48 ページの「コンポーネントベースのアプリケーションにおける イベントの処理」を参照してください。

application.onDisconnect ハンドラの使用

クライアントがアプリケーションから切断されると、サーバーは `application.onDisconnect` ハンドラを呼び出します。このハンドラに、クライアントが切断されたというこのイベントを他のすべてのクライアントに通知する次のようなコードを追加することもできます。

```
// サーバーサイドに追加するコード
application.onConnect = function(newClient, name)
{
    newClient.name = name;
    return true;
}

application.onDisconnect = function(client)
{
    for (var i = 0; i < application.clients.length; i++)
    {
        application.clients[i].call("userDisconnects", client,name);
    }
}

// クライアントサイドに追加するコード

nc = new NetConnection();
nc.userDisconnects= function (name) {
    trace(name + "quits");
}
nc.connect ("rtmp:/app_name", userName);
```

ヒント

コンポーネントを使用する場合は、[48 ページの「コンポーネントベースのアプリケーションにおける イベントの処理」](#)を参照してください。

コンポーネントベースのアプリケーションにおけるイベントの処理

コンポーネントを利用してアプリケーションを開発する場合は明示的な `onConnectAccept` イベントと `onConnectReject` イベントが発生するため、これらのイベントを処理するコードを記述する必要があります。サーバーサイドコードの `application.onConnect` ステートメントに、`application.onConnectAccept` イベントハンドラと `application.onConnectReject` イベントハンドラを含める必要があります。`onConnect` ハンドラにおいて、実行順で最後のステートメントは、`application.acceptConnection()` か、`application.rejectConnection()` のどちらかとします。アプリケーションへのアクセスの許可または拒否のメッセージをユーザーに告知する場合など、明示的な `acceptConnection()` メソッドあるいは `rejectConnection()` メソッドの後に追加のコードを実行する必要があるときは、`application.onConnectAccept` または `application.onConnectReject` ステートメント中にそのようなコードを追加します。

ヒント

メディアコンポーネントを使用しない場合には、`application.onConnectAccept` と `application.onConnectReject` は使用できません。

各コンポーネントは、独自に `application.onConnect` ハンドラを使用できます。`onConnectAccept` と `onConnectReject` を使用すれば、コンポーネントのカスタマイズや新しいコンポーネントの作成により、認証を組み込むことができます。たとえば、コンポーネントの中のコードでデータベースにクエリーを実行し、データベースに保存されているユーザー名とパスワードをもとにユーザーの許可または拒否を決定することも可能です。

`application.onConnectAccept` と `application.onConnectReject` の詳細については、『サーバーサイド ActionScript リファレンスガイド』を参照してください。

Camera クラス

このセクションでは、Camera クラスを効率的に使用するために役立つ情報として、記録後にカメラをオフにする方法、カメラの設定を利用可能な帯域幅に合わせる方法、複数のアプリケーションで1台のカメラを使用する方法などを紹介します。

カメラをオフにする

アプリケーションの中で Camera オブジェクトを使用し、NetStream オブジェクトに接続してデータを記録すると、記録が終了してもそのカメラはオンのままになります。記録が終了した後にカメラをオフにするには、`NetStream.attachVideo(false)` メソッドを使用します。

帯域幅速度に合わせた設定

デフォルトのカメラ設定では、どのような帯域幅設定でもよく写るようになっていますが、帯域幅に合わせて設定を変えて試してみることもできます。

帯域幅速度を設定するためのコードは次のとおりです。

```
my_cam = Camera.get();  
my_cam.setQuality(bandwidthSpeed,quality)
```

帯域幅速度に合わせてカメラ設定を変える場合は、次の表を参考にしてください。

帯域幅	効果	コード
モデム	イメージの品質:低、モーションの品質:高	<code>my_cam.setQuality(4000,0)</code>
	イメージの品質:高、モーションの品質:低	<code>my_cam.setQuality(0,65)</code>
DSL	イメージの品質:低、モーションの品質:高	<code>my_cam.setQuality(12000,0)</code>
	イメージの品質:高、モーションの品質:低	<code>my_cam.setQuality(0,90)</code>
LAN	イメージの品質:低、モーションの品質:高	<code>my_cam.setQuality(400000,0)</code>
	イメージの品質:高、モーションの品質:低	<code>my_cam.setQuality(0,100)</code>

複数のアプリケーションによる1台のカメラの使用

複数のアプリケーション (SWF ファイル) が同じプロセスの中で実行中である場合に限り、それらのアプリケーションは同時に1台のカメラを使用できます。一般的に、複数のブラウザウィンドウはすべて同じプロセスの中にあるため、ブラウザ環境においてはこの機能はうまく動作します。

けれども、たとえば、一方がブラウザで、他方がスタンドアロンのプレーヤーの中でというように、2つのプロセスで実行中のアプリケーション間で1台のカメラを共有することはできません。

Client クラス

サーバーサイドスクリプトの中で Client オブジェクトにメソッドを割り当てている場合には、その Client オブジェクトに対するすべてのメソッドはクライアント上の SWF ファイル内のスクリプトから呼び出せるので注意してください。リモートコンピュータから呼び出されないようにしたいメソッドについては、Client オブジェクトに含めないでください。たとえば、アプリケーションを切断するメソッドは、クライアントから呼び出せるようにはしたくないでしょう。

Microphone クラス

このセクションでは、Microphone クラスを効果的に使用するために役立つ情報として、オーディオフィードバックを回避するためのヒントなどを紹介します。

オーディオフィードバックの回避

外部スピーカーが付属していて、比較的ハイゲインのマイクを使用すると、オーディオフィードバックの問題が発生することがあります。スピーカーからのオーディオフィードバックを減らすために、Flash Media Server にはエコー抑制の機能が用意されており、次のコマンドで使用することができます。

```
my_mic.useEchoSuppression(true);
```

このコマンドによって、スピーカーから過度なエコーが発生しない程度の適度な入力レベルに保たれます。

エコー抑制はユーザーの入力から出力信号の一部分だけを取り除くため、使用しているマイクとスピーカーが近すぎると、オーディオフィードバックが残ることがあります。オーディオフィードバックを回避するためには、次のガイドラインに従ってください。

- スピーカーの音量を下げる
- マイクとスピーカーを離す
- 使用しているハードウェアを正しい設置方法と設定値に直す
- ヘッドセットを使う

マイクのオン状態の維持

帯域幅を節約するために、Flash Media Server は、デフォルトでは、使用中でなければマイクをオフにしますが、たとえば、マイクをオンにするための遅延が許されないような状況では、アプリケーションの中でマイクをオン状態に維持できます。マイクをオンのままにしておくには、

```
my_mic.setSilenceLevel(0) コマンドを使用します。
```

NetConnection クラス (クライアントサイド)

Flash アプリケーション (SWF ファイル) が含まれている HTML ページがあり、その SWF ファイルが Flash Media Server へのアクセスに使用するドメイン名とは異なるドメイン名でその HTML ページがアクセスされると、Flash Media Server への接続は失敗します。これは Flash Player のセキュリティ機能によるものですが、状況によってはこの機能が不便になることもあります。

たとえば、Flash アプリケーションが含まれている Web ページがイントラネット上の `http://deptserver.mycorp.com` にあって、`http://deptserver` でもアクセスできるようになっているとします。ところが、そのページが URL `http://deptServer/tcpage.htm` を使ってアクセスされ、SWF ファイルで `NetConnection.connect()` メソッドの `targetURI` パラメータに `deptServer.mycorp.com` と指定していたとすると、SWF ファイルから Flash Media Server への接続は成功しません。同様に、その Web ページと SWF ファイルが URL `http://deptserver.mycorp.com/tcpage.htm` を使ってアクセスされ、SWF ファイルで `targetURI` パラメータに URL `rtmp://deptserver` を使用していたとすると、SWF ファイルは Flash Media Server に接続されません。

このようなセキュリティポリシーによってユーザーが不便を感じることをないようにする方法はいくつかあります。最も簡単な方法の1つが、`targetURI` パラメータにサーバー名を入れないことです (この方法は、Flash Media Server と Web サーバーが同じマシン上で稼動している場合にのみ有効です)。そのためには、`targetURI` の2つ目のスラッシュを省略します。たとえば、次のコマンドを使えば、Flash Player は SWF ファイルが取り出される Web サーバーと同じホストとドメインへの接続を試行するようになります。

```
nc = new NetConnection();
nc.connect("rtmp:/myApp");
```

2つ目は、HTML ページで JavaScript を使用することで、セキュリティのこの問題を回避する方法です。SWF ファイルが完全修飾ドメイン名 URL を使用して Flash Media Server にアクセスする場合には、次の JavaScript によって、Web ページを明示的に指定した完全名の URL にリダイレクトします。

```
<SCRIPT language="javascript">
//URL にドメインがない場合
if (document.URL.indexOf("mycorp.com") == -1) {
    // ドメインがある URL にリダイレクトする
    document.URL="http://deptServer.mycorp.com/tcpage.htm";
}
</script>
```

最後は、Flash Media Server を稼動させるコンピュータに、独自の IP アドレスとホスト名を与えてしまう方法です。これは、独自のドメイン名を持っていてそのドメイン名の DNS レコードにアクセスでき、Flash Media Server を稼動させるコンピュータに、静的な IP アドレスを割り当てられる場合にのみ可能です。その IP アドレスのホスト名をポイントするアドレス ("A") レコードを作成します。たとえば、`flashcom.mycorp.com` とすることで、Flash Media Server が稼動していて、IT 部門か ISP が割り当てた IP アドレスのコンピュータにマッピングします。これで Web ページは、現在使用中の手段でホスティングを続行できます。(DNS アクセスが可能なホスト名があり、静的な IP アドレスがあるかどうか不明のサーバーにトラフィックを転送する必要がある場合は、"CNAME" レコードを推奨します。)

NetStream クラス

このセクションでは、NetStream クラスの効果的な使い方として、ストリームへのデータの組み込み、ストリームのバッファの管理、ストリームの再生終了後のコードの実行などのヒントを紹介します。

ストリームにおける複数のデータタイプの使用

ストリームの中には、ストリーミングオーディオとビデオに加え、テキストメッセージのようなデータを入れることもできます。ストリームに複数のデータタイプを追加するには、

`NetStream.send()` メソッドを使用します。

ActionScript によるストリーム長の取得

ストリームをバッファリングする場合は、`NetStream.bufferLength` プロパティを使用してバッファ内の現在の秒数を取得することができますが、ストリーム全体の長さを取得したいこともあるでしょう。Flash Player にはこの情報はわかりませんが、サーバーにはわかります。サーバーには `Stream.length` プロパティがあり、クライアントはサーバーへのメッセージを介してこのプロパティを要求できます。

ストリームの長さを要求する次のようなクライアントサイド ActionScript を記述します。

```
function getInfo(){
    nc.call("sendInfo", new MyResultSetName(), myStream);
}
function MyResultSetName(){
    this.onResult = function (retVal){
        _root.streamlength = retVal;
    };
    this.onStatus = function(info){
        trace("Level:" + info.level + "    Code:" + info.code);
        // エラーオブジェクトの処理
    };
}
```

さらに、対応するサーバーサイド ActionScript では、main.asc ファイルに次のように記述します。

```
application.onAppStart = function(){
    trace("::: Application has started :::");
}
application.onConnect = function(client){
    application.acceptConnection(client);
    // メソッドの追加
    client.prototype.sendInfo = function(name) {
        var slen = Stream.length(name);
        trace("slen: " + slen);
        return slen;
    };
}
```

ストリームのバッファリング

再生ストリームでは、NetStream.bufferTime プロパティから返される値は、Flash Player がストリームの再生を開始する前に Flash Media Server がそのストリームをバッファリングするのに要する時間を表します。たとえば、記録したストリームの長さが 50 秒で、再生速度がダウンロード速度の 2 倍速かったとすると、この値には 25 秒を設定すると考えればいいでしょう。そうすることで、滞ることなく、すべてのストリームを再生できます。

ストリームのパブリッシュでは、この値は、Flash Player がメッセージのドロップを実行し始めるまで、どの位の長さまで現行のキューを増やせるかを表します。現在、どの位のデータがキューに蓄積されているかを調べるには、NetStream.bufferLength を使用します。高速の接続においては、NetStream.bufferLength から返される値が NetStream.bufferTime に近づくことはありませんが、低速の接続ではこの 2 つの値が近づく可能性があります。

そのため、作成するアプリケーションが低速の接続を使って実行されることがわかっていて、メッセージのドロップあるいは再生の停滞を最小限に抑えたい場合は、NetStream.setBufferTime() メソッドを使用して NetStream.bufferTime の値を大きくするという方法が考えられます。

ストリームの再生ステータスについて

ストリームが最後まで再生されると、NetStream.onPlayStatus イベントハンドラが呼び出されます。コールバック関数に渡される情報オブジェクトによって、NetStream オブジェクトが再生リスト内のあるストリームから別のストリームに切り替わったのか (NetStream.Play.Switch)、NetStream オブジェクトが最後まで再生されたのか (NetStream.Play.Complete) を判断するための追加情報が提供されます。このイベントハンドラに応答するには、引数として与えられた情報オブジェクトを処理する必要があります。

SharedObject クラス

共有オブジェクトは多くの目的に使用でき、さまざまな方法で設計し、使用することができます。このセクションでは、アプリケーションの中で共有オブジェクトを使う前に考慮すべき点を説明します。

共有オブジェクトの同期化について

リモート共有オブジェクトを使って作業を始める前に、まず、`SharedObject.connect()` メソッドから接続が成功したことを示す値 `true` が返されたことを検査し、その後に、`SharedObject.onSync` ハンドラに割り当てた関数から結果を受け取るまで待ちます。この処理が失敗した場合は、`SharedObject.onSync` が呼び出される前にローカルでオブジェクトに対して行った変更は失われる可能性があります。

リモート共有オブジェクトの `SharedObject.data` プロパティのいずれかが変更されると、`SharedObject.onSync` ハンドラが呼び出されます。また、クライアントが最初に `SharedObject.getRemote()` メソッドを使用してローカルまたはサーバー上に永続するリモート共有オブジェクトに接続した場合にも、このハンドラが呼び出されます。後者の場合は、オブジェクトのすべてのプロパティに空の文字列がセットされ、`code` プロパティには "clear" という値がセットされて、`SharedObject.onSync` が呼び出されます。次に、再び `SharedObject.onSync` が呼び出されますが、今回は、`code` の値は "change" にセットされ、リモート共有オブジェクトのクライアント側のインスタンスのプロパティにはサーバーと一致する値がセットされます。

共有オブジェクトがどのように動作するのがわからない場合は、`SharedObject.onSync` ハンドラに次の例のようなデバッグコードを入れておくといいでしょう。

```
my_so.onSync = function(list) {
  for (var k in list) {
    trace("name = " + list[k].name + ", event = " + list[k].code);
  }
  // 他に行うべき処理があればここに記述
}
```

共有オブジェクトスロットの効果的な使用

共有オブジェクトのデータを1つのスロットにカプセル化するか(`SharedObject.data` プロパティ)、あるいは複数のスロットに分散させるかを決定する場合に、アプリケーションがどのようにオブジェクトを変更するのかを考慮するという方法があります。たとえば、アプリケーションが文字列全体を特定の間隔で接続しているすべてのクライアントに送信する必要がある場合であれば、その文字列全体を1つのスロットに格納するといいいでしょう。

一方、アプリケーションは変更された情報だけを送信すればいいのであれば、データを複数のスロットに分割するといいいでしょう。この方法では、ネットワークトラフィックが少なくなるため、アプリケーションのパフォーマンスが向上します。さらには、データが衝突することなく、複数のスロットを同時に更新できるため、コンフリクトを解決するために必要となるコードを最小限に抑えることができます。

リモート共有オブジェクトのフラッシュ

すべてのユーザーが共有オブジェクトから切断された場合、あるいはサーバーが停止した場合には、Flash Media Server はクライアントとサーバーの両方で自動的にリモート共有オブジェクトをディスクにフラッシュ (その時の状態を書き込み) します。それ以外のタイミングで共有オブジェクトの変更をディスクに反映させるには、クライアントサイド `SharedObject.flush()` メソッドを明示的に呼び出す必要があります。

ただし、クライアントサイド `ActionScript` で `SharedObject.flush()` を呼び出した場合は、共有オブジェクトのローカルのコピーだけがフラッシュされます。共有オブジェクトのサーバー上のコピーを手動でフラッシュするには、サーバーサイドスクリプトで次のように `SharedObject.flush()` を呼び出す必要があります。

```
// 永続共有オブジェクトをサーバーにフラッシュするための
// サーバーサイドのコード例

// アプリケーションのロード時に共有オブジェクトを取得
application.onAppStart = function()
{
    application.mySO = SharedObject.get("SharedObjName", true);
}

// ユーザーの接続解除時に、共有オブジェクトをフラッシュ
application.onDisconnect = function(client)
{
    application.mySO.flush();
}
```

共有オブジェクトの同期化についての問題の回避

複数のクライアント (またはサーバーアプリケーション) が共有オブジェクトの1つのスロット内のデータを同時に変更できる場合には、コンフリクトを解決するための手法を開発者が組み込む必要があります。ここでは、次のようないくつかの例を紹介します。

異なるスロットを使用する 最も簡単な手法は、オブジェクト内のデータを変更する可能性があるユーザーごとに異なるスロットを使用する方法です。たとえばチャットルームでは、ユーザーごとにスロットを用意し、ユーザーが自分のスロットしか変更できないようにします。

所有者を割り当てる 少し複雑な手法としては、時間を限定してある特定のクライアントを共有オブジェクト内のプロパティの所有者として定義する方法があります。これを実現させるためには、「ロック」オブジェクトを生成するサーバーサイドコードを記述する方法があります。クライアントはこのオブジェクトの所有権をサーバーにリクエストし、もし許可された場合はその共有オブジェクト内のデータを変更できる唯一のクライアントとなる、という仕組みです。

次に示すサーバーサイド ActionScript では、ゲームから返されたハイスコアが正確であることを保証するために、共有オブジェクトをロックし、アンロックしています。直前のハイスコアが95で、プレイヤー1のスコアが105に、プレイヤー2のスコアが110になったとします。ロックをかけておかないと、両プレイヤーのスコアを直前のハイスコア95と比較できるかもしれませんが、両方のクライアントが同時に `updateHighScore` を呼び出したとすると衝突が発生します。直前のハイスコアか他のすべてのクライアントから入ってくる最新のスコアであるかに関係なく、各プレイヤーのスコアを確実に最高のスコアと比較するようにしたければ、最高のスコアを格納するために使用している共有オブジェクトをロックし、アンロックします。そうすることで、各スコアを順番に比較できるため、比較が失敗することはありません。

```
application.onAppStart = function()
{
    application.scoreS0 = SharedObject.get("high_score_so", true);
    application.scoreS0.onSync = function(listVal)
    {
        trace("got an onSync on scoreS0");
    }
}

application.onConnect = function(newClient,name,passwd)
{
    newClient.updateHighScore = function(final_score)
    {
        application.scoreS0.lock();
        if (application.scoreS0.getProperty("high_score_so") < final_score)
        {
            application.scoreS0.setProperty("high_score_so", final_score);
        }
        application.scoreS0.unlock();
    }
}
```

クライアントに通知する 共有オブジェクトのプロパティに対してクライアントから要求された変更をサーバーが却下すると、`SharedObject.onSync` イベントハンドラがクライアントに対して、変更が却下されたことを通知します。それを使って、ユーザーにコンフリクトが生じて失敗に終わったことを通知し、時間を置いて再度アクセスしてもらうよう促します。この方法は、サーバー上で共有するアドレス帳のように、データの変更が頻繁でない場合に適しています。同期化においてコンフリクトが生じた場合は、ユーザーはその変更を許可するか、拒否するかを選択できます。

先着順処理 アプリケーションによっては、"先着順処理"が適しているものがあります。ほかのユーザーが変更中だったときにはそれを優先し、その後で変更を適用することに問題がなければ、この方法が適しています。

SharedObject.send() メソッドを使って変更に対する制御レベルを向上させる

SharedObject.onSync だけに頼って同時実行を管理するのではなく、必要に応じて SharedObject.send() メソッドを使用します。SharedObject.send() メソッドは、メッセージを送ったクライアントを含め、リモート共有オブジェクトに接続しているすべてのクライアントにメッセージを配信します。

Stream クラス

記録されたストリームに関連する FLV ファイルと IDX ファイルを削除したい場合は、次のコードをサーバーサイドで使用する必要があります。

```
s = Stream.get("foo");
if (s)
{
    s.clear();
}
```

System クラス

Macromedia Flash Media Server は、テキストを UTF-8 形式で送信します。ActionScript コードの中で System.useCodepage に true という値をセットすると、メディアアプリケーションが正しく動作しないことがあります。true という値によって、Flash Player はエンドユーザーのコードページに基づいてすべてのテキストを扱うようになります。これは、Flash Player の以前のバージョンと同じ動作です。ところが、Flash Player は UTF-8 でエンコードされたテキストを Flash Media Server に送信し、サーバーから受け取るテキストもすべて UTF-8 でエンコードされています。Flash Media Server アプリケーションで System.useCodepage = true にセットすると予期せぬ動作や期待に反する結果を招くおそれがあるため、このように設定しないでください。たとえば、サーバーとの間でテキストの送受信が正しく行われられない可能性があります。System.useCodepage の詳細については、『ActionScript 2.0 リファレンスガイド』または Macromedia の Web サイトに掲載されている記事 (英文のみ) 『Unicode in Macromedia Flash』を参照してください。

Video クラスについて

このセクションでは、Microphone クラスを効果的に使用するために役立つ情報として、Video オブジェクトの動的な作成についてのヒントを紹介します。

ビデオの詳細については、[第 4 章の「メディアファイルの使用」](#)を参照してください。

Video オブジェクトの動的な作成について

アプリケーションに Video オブジェクトを追加するには、ライブラリパネルから埋め込み Video オブジェクトをステージにドラッグします。この作業は Flash オーサリング環境内でのみ可能です。ただし、ActionScript コード内から Video オブジェクトを実装したければ、ムービークリップ内に Video オブジェクトを埋め込む方法があります。この方法によって、`duplicateMovieClip()` メソッドと `removeMovieClip()` メソッドを使用して Video オブジェクトを動的に作成および削除することができます。

フレームレートについて

静的な SWF ファイル内の Flash アプリケーションに FLV ファイルを埋め込むと、そのフレームレートはタイムライン上でのフレームの再生レートと同じになります。しかし、FLV ファイル内でデータをストリームする場合、つまり Flash Media Server を通じて再生されるビデオの場合は、そのビデオが含まれている SWF ファイルとは異なるフレームレートであっても構いません。

たとえば、別のアプリケーションを使用してフレームレートが 15 fps のブロードバンド FLV ファイルに出力しているとします。この FLV ファイルを 6 fps の Flash アプリケーション (SWF ファイル) に読み込み、タイムラインに埋め込むと、2 つのフレームレートが異なるため、ビデオは同期化されません (SWF ファイルのフレームレートに合わせられてしまいます)。ところが、代わりに Flash Media Server を使用して FLV ファイルを同じ 6 fps の SWF ファイルにストリームすると、ストリーミングファイルはフレームを基準とした再生を使用していないため、15 fps で再生されます。

Macromedia Flash Media Server を使用して、広範なメディア体験を実現できます。この章では、ライブ Web イベントのブロードキャスト、個々のクライアントに対するストリーム配信のカスタマイズ、および MP3 のパブリッシュと再生に関する情報を紹介します。

ビデオの操作

Flash Media Server では、Flash Player クライアントへのライブおよび記録されたビデオのストリーミングが可能で、パワフルなビデオ配信を実現する、多様な機能を装備しています。

エッジサーバーを使用すると、サーバーの負荷が分散され、大規模なライブ Web イベントの作成が可能になります。

カスタムストリーム配信機能を利用すれば、クライアントを個々に処理することができます。Flash Player のバージョンに基づくストリーム配信が可能になるほか、わずかなコードの追加だけで、クライアントの帯域幅に応じたストリーム配信も実現されます。

ライブ Web イベントの作成について

Flash Media Server アプリケーションクライアントはライブビデオストリームをサーバーにパブリッシュでき、複数のクライアントがこのブロードキャストにサブスクライブして、受信することができます。このため、ライブ Web イベントの作成が可能です。

大規模なライブ Web イベントで多くの加入者がそのライブブロードキャストを受信する場合、加入者はプロキシサーバー（エッジサーバーとも呼びます）経由でそのライブストリームに接続することができます。エッジサーバーがオリジンサーバーからのデータトラフィックを処理するため、オリジンサーバーは多数の加入者に対して配信することが可能になります。エッジサーバーの詳細については、『Flash Media Server エッジサーバーユーザーガイド』を参照してください。

ストリーム配信のカスタマイズ

Flash Media Server のカスタムストリーム配信機能を使用すると、同じストリームでもクライアントに応じたバージョンで配信できるため、それぞれのクライアントに最適なストリーム体験を提供できます。たとえば、Flash Player 7 を使用するクライアントには Sorenson Spark コーデックでエンコードされたビデオを配信し、Flash Player 8 を使用するクライアントには On2 VP6 コーデックでエンコードされたビデオを配信することができます。また、クライアントの帯域幅に基づいて異なるビットレートでエンコードされたビデオを配信することもできます。

Flash Media Server には、併せて使用することでカスタムストリーム配信を可能にする、仮想ディレクトリと仮想キーという 2 つの機能があります。

Flash Media Server に複数のディレクトリを作成し、各ディレクトリに同じビデオコンテンツの異なるバージョン（たとえば、異なるコーデックあるいは異なるビットレートでエンコードされたビデオ）を置いておくことができます。Flash Media Server に接続するクライアントごとに仮想キーが割り当てられます（サーバーサイド `Client.virtualKey` プロパティに保存されます）。デフォルトでは、クライアントの Flash Player のバージョンを示す値に基づいて仮想キーが設定されます。`vhost.xml` 設定ファイルの `VirtualKeys` セクションと `VirtualDirectory` セクションのディレクトリにキーをマップすることで、個々のクライアントにカスタムコンテンツを配信できます。帯域幅に基づいてストリーム配信をカスタマイズする場合は、スクリプト内でクライアントの帯域幅をチェックし、それに従ってクライアントのキーを変更する必要があります。

キーとディレクトリの値は、`vhost.xml` ファイルに直接セットするか、サーバーサイド ActionScript (`Client.virtualKey` プロパティと `Stream.setVirtualPath()` メソッド) を使用して設定することもできます。`vhost.xml` ファイルに（またはプログラムで）マッピングを設定すると、接続するそれぞれのクライアントには、対応する適切なビデオコンテンツが自動的に配信されます。

`vhost.xml` 設定ファイルに関する詳細については、『Flash Media Server 管理ガイド』の「`Vhost.xml` ファイル」を参照してください。

Flash Player のバージョンに基づくストリーム配信

Flash Media Server では、On2 V6 でエンコードされたビデオと Sorenson Spark でエンコードされたビデオのエンコードと配信が可能です。Flash Player 8 はこの 2 つのコーデックをどちらもサポートしますが、Flash Player 7 以前のバージョンでサポートしているのは Sorenson Spark コーデックだけです。サーバー上に仮想ディレクトリを作成して各形式のビデオストリームのコピーを保存することで、Flash Player 8 を使用するクライアントに最高品質のコンテンツを配信できます。

ここでは、ストリームを再生しようと考えているユーザーがいて、コンピュータ上には Flash Player 8 がインストールされている例を取り上げます。Flash Player 8 では、On2 ビデオを再生できます。Flash Media Server が HappyStream.flv ファイルを要求します。サーバーへのコンタクト後に、Flash Player 8 はサーバーサイドの Client.virtualKey プロパティを調べます。virtualKey プロパティによって、デフォルトのストリームディレクトリではなく、On2 ストリームのディレクトリにマップされていて、On2 コーデックでエンコードされている HappyStream.flv ストリームが再生されます。

Flash Player のバージョンに基づく自動ストリーム配信のメカニズムを作成するには、vhost.xml ファイルを次のように編集します。

```
<VirtualKeys>
  <Key from="WIN 7,0,19,0" to="WIN 9,0,0,0">A</Key>
  <Key from="WIN 6,0,0,0" to="WIN 7,0,18,0">B</Key>
  <Key from="MAC 6,0,0,0" to="MAC 7,0,55,0">B</Key>
</VirtualKeys>

<VirtualDirectory>
  <Streams key="A">foo;c:\streams\on2</Streams>
  <Streams key="B">foo;c:\streams\sorenson</Streams>
  <Streams key="">foo;c:\streams</Streams>
</VirtualDirectory>
```

Key タグと Streams タグの編集に関する詳細については、『Flash Media Server 管理ガイド』の「Vhost.xml ファイル」を参照してください。

帯域幅に基づくビデオ配信

クライアントの帯域幅に基づいてビデオを配信するには、次の例にあるように、異なるビットレート（および必要であれば異なるコーデック）でビデオをエンコードし、そのストリームを格納するディレクトリを作成します。

- c:\streams\Sorenson22k\mystream.flv
- c:\streams\Sorenson150k\mystream.flv
- c:\streams\Sorenson300k\mystream.flv
- c:\streams\Vp622k\mystream.flv
- c:\streams\Vp6150k\mystream.flv
- c:\streams\Vp6300k\mystream.flv

次の例にあるように、vhost.xml ファイルの VirtualKeys タグで、Flash Player 7 ならびに以前のバージョンには仮想キー fp7 をセットし、Flash Player 7 (7.0.19.0) より新しい (On2 コーデックをサポートする) Player のバージョンには仮想キー fp8 を設定します。

```
<VirtualKeys>
  <Key from="WIN 8,0,0,0" to="WIN 9,0,0,0">fp8</Key>
  <Key from="MAC 8,0,0,0" to="MAC 9,0,0,0">fp8</Key>
  <Key from="WIN 6,0,0,0" to="WIN 7,0,55,0">fp7</Key>
  <Key from="MAC 6,0,0,0" to="MAC 7,0,55,0">fp7</Key>
</VirtualKeys>
```

次の例にあるように、“fp7slow”、“fp8slow”、というようにクライアントキーを検索するように仮想ディレクトリを設定します。

```
<VirtualDirectory>
  <Streams key="fp7slow">foo;c:\streams\Sorenson22k</Streams>
  <Streams key="fp7medium">foo;c:\streams\Sorenson150k</Streams>
  <Streams key="fp7fast">foo;c:\streams\Sorenson300k</Streams>
  <Streams key="fp8slow">foo;c:\streams\Vp622k</Streams>
  <Streams key="fp8medium">foo;c:\streams\Vp6150k</Streams>
  <Streams key="fp8fast">foo;c:\streams\Vp6300k</Streams>
  <Streams key="">foo;c:\streams</Streams>
</VirtualDirectory>
```

次の例にあるように、サーバーサイドスクリプトで、application.onConnect ハンドラには、帯域幅を検出するルーチンを記述し、各クライアントの仮想キー値に範囲を付加します。

```
client.virtualKey += "slow"
client.virtualKey += "medium"
client.virtualKey += "fast"
```

以上の設定によって、Flash Media Server はクライアントの Flash Player バージョンと帯域幅に基づいて個々のクライアントにコンテンツを配信するようになります。

仮想ディレクトリと仮想キー間のマッピング

仮想キーと仮想ディレクトリ間のマッピングは、vhost.xml ファイルで行います。vhost.xml ファイルの <VirtualKeys> セクションは、次の例にあるように、仮想キーを Flash Player バージョンにマップします。

```
<VirtualKeys>
  <Key from="WIN 7,0,19,0" to="WIN 9,0,0,0"></Key>
  <Key from="WIN 6,0,0,0" to="WIN 7,0,18,0"></Key>
  <Key from="MAC 6,0,0,0" to="MAC 7,0,55,0"></Key>
</VirtualKeys>
```

デフォルトでは、Key タグに値はありません。この機能を実装するには、vhost.xml ファイルに直接設定するか、`Stream.setVirtualPath()` メソッドを使用するかのどちらかの方法で、キー値を追加する必要があります。たとえば、次のコードで示すように、最初のキーには A を、次の 2 つのキーには B をセットすることで、この機能を実装します。

```
<VirtualKeys>
  <Key from="WIN 7,0,19,0" to="WIN 9,0,0,0">A</Key>
  <Key from="WIN 6,0,0,0" to="WIN 7,0,18,0">B</Key>
  <Key from="MAC 6,0,0,0" to="MAC 7,0,55,0">B</Key>
</VirtualKeys>
```

vhost.xml ファイルの `<VirtualDirectory>` セクションは、次のようになります (キー属性値と Streams タグ値はデフォルトではこのファイルに存在しません)。仮想キーは、セミコロンで区切られた仮想パスと物理ディレクトリ (たとえば、`foo;c:\streams`) にマップされています。Flash Player のバージョンごとに仮想ディレクトリをセットアップするには、次の例にあるように、Streams タグごとに異なる物理ディレクトリを設定した、同じ仮想パスを使用します。

```
<VirtualDirectory>
<!-- 記録されたストリームに対する仮想ディレクトリのマッピングを指定 -->
<!-- 1 つのストリームに複数の仮想ディレクトリを指定するには、 -->
<!-- <Streams> タグを仮想ディレクトリのマッピングごとに -->
<!-- さらに追加する。仮想ディレクトリの構文は、 -->
<!-- <Streams>foo;c:\data</Streams> で、こうすることで、名前が "foo/" で始まる -->
<!-- すべてのストリームが物理ディレクトリ c:\data にマッピングされる。 -->
<!-- たとえば、"foo/bar" という名前のストリームは -->
<!-- 物理ファイル "c:\data\bar.flv" にマッピングされる。同様に、"foo/bar/x" という名前の -->
<!-- ストリームがあれば、最初に "foo/bar" の仮想ディレクトリ -->
<!-- マッピングを探そうとする。それに失敗したら、次に "foo" の -->
<!-- 仮想マッピングをチェックする。該当するものがあるため、ストリーム -->
<!-- "foo/bar" はファイル "c:\data\bar\x.flv" に対応する。 -->
  <Streams key="A">foo;c:\streams\on2</Streams>
  <Streams key="B">foo;c:\streams\sorenson</Streams>
  <Streams key="">foo;c:\streams</Streams>
</VirtualDirectory>
```

vhost.xml ファイルに関する詳細については、『Flash Media Server 管理ガイド』の「Vhost.xml ファイル」を参照してください。

MP3 ファイルの操作

Flash Media Server アプリケーションでは、クライアントサイド ActionScript を使用することで MP3 オーディオファイルの再生と MP3 ファイルの ID3 タグの表示が可能であり、サーバーサイド ActionScript を使用することでストリームを介した MP3 ファイルのパブリッシュが可能です。

そのためには、アプリケーションが使用する MP3 ファイルを、登録アプリケーションディレクトリの /streams/application_instance サブディレクトリにアップロードします。(/streams サブディレクトリが存在しない場合、ストリームの記録時に Flash Media Server が自動的に作成しますが、自分で作成することもできます。)たとえば、CDPlayerApp という名前のアプリケーションが Flash Media Server のアプリケーションディレクトリにあるとすると、このアプリケーションが使用する MP3 ファイルを /applications/CDPlayerApp/streams/application_instance にアップロードします。

共有する MP3 ファイルをあるディレクトリに入れ、アプリケーションの Vhost.xml ファイルの Streams タグの virtual directory でこの共有ディレクトリの場所を指定することで、アプリケーションのすべてのインスタンス間で MP3 ファイルを共有できます。さらに、Stream.play() ステートメントでは、この仮想ディレクトリと再生したい MP3 ファイルを指定します。仮想ディレクトリと Streams タグについては、『Flash Media Server 管理ガイド』の「クライアントアプリケーションの登録」を参照してください。

MP3 ファイルを再生するには、NetStream オブジェクトを Video オブジェクトに割り当ててから play() メソッドを呼び出すか、NetStream オブジェクトを MovieClip オブジェクトに割り当てて attachAudio() メソッドを呼び出します。streamname、すなわち、何を再生するかを指定するパラメータでは、MP3 ファイル名の前には mp3: を付けてください。次のコード例では、mystream というネットワークストリーム内の bolero.mp3 というファイルを再生する 2 種類の方法を紹介します。

```
// すでに開かれている mystream_ns ストリームの中の bolero.mp3 を Video オブジェクトを使って再生
vidObj_video.attachVideo(mystream);
mystream_ns.play("mp3:bolero");
```

```
// すでに開かれている mystream2_ns ストリームの中の bolero.mp3 を MovieClip オブジェクトを使って再生
// bolero.mp3 は Flash Media Server コンピュータ上の C:\mp3_files ディレクトリに置かれていて、
// これは Vhost.xml の中で仮想ディレクトリ mp3dir にマップされている
movieObj_mc.attachAudio(mystream2_ns);
mystream2_ns.play("mp3:mp3dir/bolero");
```

ヒント	NetStream.play() ステートメントの中で、ビデオおよびオーディオファイルのデフォルトの形式である FLV は指定しなくても構いませんが、MP3 ファイルを再生する場合は、必ず MP3 形式を指定する必要があります。つまり、"flv:granada" と "granada" はどちらを指定しても granada.flv が再生されますが、"mp3:bolero" と指定しないと bolero.mp3 ファイルは再生されません。
-----	---

MP3 ファイルの ID3 タグを表示するには、ストリーム名の前に `id3:` と指定し、ID3 データをキャプチャするためのコールバック関数を定義します。たとえば、`bolero.mp3` の ID3 タグを表示するには、次のように記述します。

```
// bolero.mp3 の ID3 タグを表示
mystream_ns.play("id3:bolero");

// ID3 データをキャプチャするためのコールバック関数。タグからのデータには
// "info" が前に付加されて表示される。例: info.songtitle
mystream_ns.onId3 = function(info){
    for (i in info){
        trace(i + ":" + info[i]);
    }
}
```

サポートする ID3 タグのバージョンについて

Flash Media Server は、UTF-8、UTF-16、および ISO-8859-1 の各形式の再生をサポートしており、ID3 のバージョン 1.0、2.3、および 2.4 をサポートしています。曲タイトル、アーティスト名、コメント、録音年のような、テキストデータが含まれているタグのみをサポートしています。

サーバーサイド ActionScript による MP3 ファイルの制御

サーバーサイドの Stream オブジェクトのメソッドを使用することで、MP3 ファイルの再生 (`Stream.play()`) と MP3 ファイルの長さの取得 (`Stream.length()`) が可能です。MP3 ファイルとこれらのメソッドの使用に関する詳細については、『サーバーサイド ActionScript リファレンスガイド』の「`Stream.play()`」と「`Stream.length()`」を参照してください。

サーバーサイド ActionScript を使用して MP3 ファイルを削除するには、`Application.clearStreams()` メソッドを使用します。

ストリームまたは MP3 ファイルに関連付けられている ID3 タグ情報を介して MP3 ファイルをパブリッシュするには、次の例にあるように、`Stream.play()` メソッドを使用します。

```
// サーバーストリームをセットアップ
application.myStream = Stream.get("music");
if (application.myStream)
{
    // bolero.mp3 という MP3 ファイルを "music" というストリームにパブリッシュ
    // ストリームの前には mp3: という接頭辞を付け、startTime パラメータには 0 を指定して、記録されている bolero.mp3 というストリームをサーバーが再生するよう指示する
    application.myStream.play("mp3:bolero", 0, -1);
}
```

Stream.play() メソッドを使用して ID3 タグのテキストのキャプチャし、再生する方法は、次の例を参照してください。

```
// サーバーストリームをセットアップ
application.myStream = Stream.get("description");
application.myStream.onId3 = function(info)
{
    for (i in info)
    {
        trace(i + ":" + info[i]);
    }
}
if (application.myStream)
{
    // bolero.mp3 の ID3 テキストデータを "description" というストリームにパブリッシュ
    // ストリーム名の前には id3: という接頭辞を付け、startTime パラメータには 0 を指定
    application.myStream.play("id3:bolero", 0, -1);
}
```

Macromedia Flash Media Server 2 の Management Console を使って、アプリケーション、ユーザー、およびサーバーの制御を行うことができます。この章では、アプリケーションの使用状況、ストリームデータ、および共有オブジェクトデータを表示する方法を紹介します。サーバー管理者および vhost 管理者の追加と削除、サーバーのパフォーマンス統計の表示、およびサーバーの起動と停止に関する詳細については、『Flash Media Server 管理ガイド』を参照してください。

Management Console を使用したアプリケーションのデバッグと監視

Flash Media Server の Management Console を使用して、現在、サーバー上で実行中のアプリケーションをデバッグ、監視することができます。選択したインスタンスに関して、次のような詳細情報やステータスを表示することができます。

- サーバー上のそのアプリケーションインスタンスによって生成されるログメッセージ
- そのアプリケーションインスタンスに接続しているクライアントの一覧
- そのアプリケーションインスタンスのアクティブな共有オブジェクトの一覧
- そのアプリケーションインスタンスのアクティブなストリームの一覧
- インスタンスの合計実行時間やユーザー数など、選択したアプリケーションの全体的な状態に関する情報

たとえばデザイナーは、アプリケーションをデバッグするのに、特定ストリームのコンテンツを表示させる必要があります。この場合、管理者として Management Console から Flash Media Admin Server にログインし、アプリケーションを選択した上で [Streams] タブを選択します。コンソールペインにストリーミングコンテンツが表示されます。

Management Console を開くには、Macromedia Flash で [ウィンドウ]-[他のパネル]-[Management Console] を選択します。Flash がインストールされていないサーバーコンピュータでは、//Flash Media Server 2/fms_help/html/admin ディレクトリにある Management Console の HTML ファイルを開きます。

Management Console を使用するには、完全なサーバー管理者権限または仮想ホスト管理者権限が必要です。

Management Console のサーバーへの接続

Management Console を開いたら、ログイン画面を使用して、管理対象とするアプリケーションが実行中のサーバーに管理者として接続します。



Management Console のログイン画面を使用すると、サーバーに接続することができます。

サーバーに接続する手順は以下のとおりです。

1. Windows の場合は、Windows の [スタート] メニューから、[すべてのプログラム] - [Macromedia] - [Flash Media Server] - [Management Console] を選択します。Linux の場合は、Flash Player がインストールされているコンピュータ上の Web ブラウザから、fmsconsole.swf ファイルを開きます。
2. [Server Name] テキストボックスにサーバーの名前を入力するか、ポップアップメニューを使用して、正常に接続されているいずれかのサーバーの名前を選択します。(このサーバー名はユーザーが使用するための識別子なので、任意のサーバー名を付けることができます。)
3. [Server Address] テキストフィールドにサーバーの URL を入力するか、サーバーと Management Console が同じコンピュータ上で稼働している場合には **localhost** と入力します。サーバーが 1111 (デフォルト) 以外のポートにインストールされている場合は、たとえば **localhost:1234** というように、ポート番号も入力します。
4. 管理者のユーザー名とパスワードを入力します。

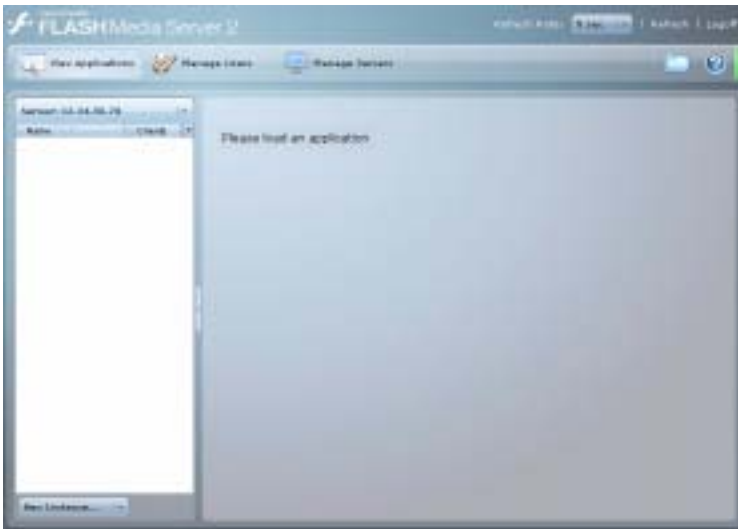
5. どのセッションでもこのユーザー名とパスワードを使用する場合には、[Remember my password] を選択します。
6. Management Console を開くたびにこのサーバーに自動的に接続するには、[Automatically Connect Me] を選択します。
7. [Login] をクリックします。

ヒント	Management Console のこの初期画面には、デベロッパーセンター、サポートセンター、リソースノート、ドキュメント、オンラインヘルプなどの Flash Media Server リソースへのリンクも数多く設定されています。
-----	---

[View Applications] パネルについて

サーバーまたは仮想ホストに接続されると、Management Console は、現在実行中のアプリケーションインスタンスを一覧表示するパネルを表示します。

ヒント	ヘルプボタンの疑問符の右にある表示灯のアイコンは、Management Console がサーバーに接続されているかどうかを示します。最初に Management Console を起動したとき、この表示灯は赤になっています。赤は、サーバーに接続していないことを示します。サーバーへの接続が成功すると、この表示灯が緑になります。
-----	--



[View Applications] パネルを使用して、アプリケーションインスタンスの表示、ロード、およびアンロードを行います。

[View Applications] パネルでは、次の作業を実行できます。

- アプリケーションインスタンスを手動でロードするには、[New Instance] をクリックし、ポップアップメニューからアプリケーションを選択します。そのアプリケーションは、サーバー上に既に設定されている必要があります。ロードするアプリケーションを選択したら、アプリケーションインスタンスの名前を入力することができます。
- 実行中のアプリケーションインスタンスに関する情報を表示するには、この一覧からアプリケーションを選択します。[Live Log] パネルが表示されます (71 ページの「[Live Log] パネルについて」を参照してください)。
- アプリケーションインスタンスを終了させるには、そのインスタンスのすべてのユーザーの接続を解除し、そのインスタンスが使用しているリソースがあればそれも解放してから、一覧からアプリケーションを選択し、[Unload this application] (パネル名の右にある小さいボタン) をクリックします。
- アプリケーションを名前順に並べるには、[Name] という列ヘッダーをクリックします。クライアント順に並べ替えるには、[Clients] という列ヘッダーをクリックします。
- Management Console とサーバー間の接続を解除するには、[Logoff] をクリックします。ログオン画面に戻ります (68 ページの「Management Console のサーバーへの接続」を参照してください)。

更新間隔の変更または一時停止

Management Console のパネルの情報は、デフォルトでは 5 秒ごとに更新されます。この更新間隔はいつでも、(1 ~ 60 秒の任意の間隔に) 変更でき、更新オプションを一時停止することもできます。

更新間隔を変更する手順は次のとおりです。

- [Refresh Rate] の隣にあるポップアップメニューをクリックし、たとえば 10 秒というように新しい間隔を選択します。
パネルに表示されている情報は 10 秒ごとに更新されます。

更新オプションを一時停止する手順は以下のとおりです。

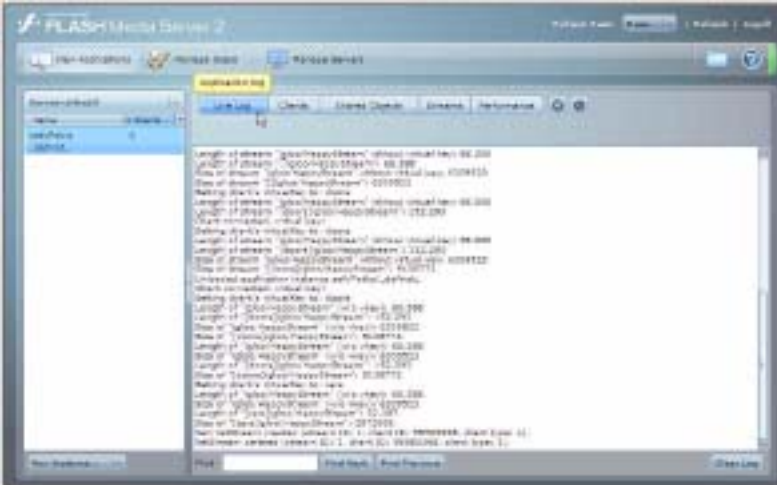
1. [Refresh Rate] の隣にあるポップアップメニューをクリックし、[Pause] を選択します。
操作を確認するメッセージが表示されます。
2. [Pause Refresh] をクリックして、続行します。

Management Console (管理コンソール) のパネルの回りに赤い境界線が表示され、これは更新の機能が一時停止していることを示します。

情報の更新を再開するには、さきほどのポップアップメニューをクリックし、更新間隔を選択します。

[Live Log] パネルについて

[Live Log] パネルには、サーバー上の選択したアプリケーションインスタンスがアプリケーション内の `trace()` ステートメントなどによって生成したログメッセージが表示されます。このパネルの情報は、パネルが更新されるときではなく、対象のアプリケーションインスタンスがログメッセージを生成するたびに更新されます。(コンソールを更新する機能を一時停止している場合であっても、ログメッセージは受け取ります。)



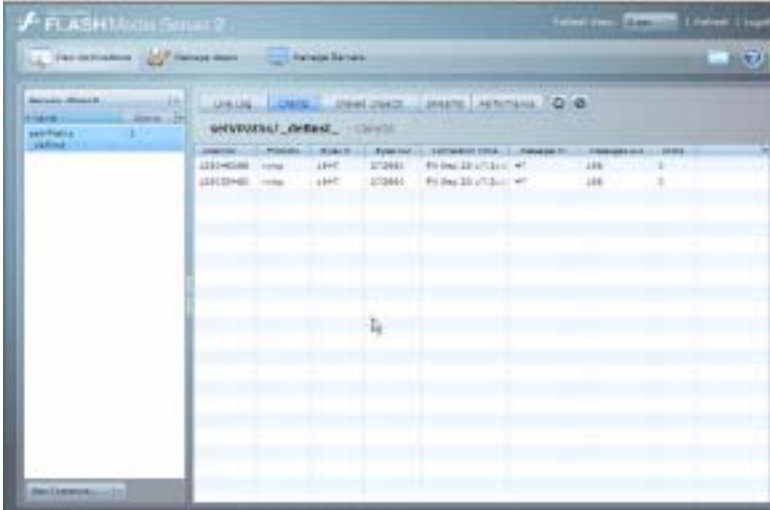
[Live Log] パネルには、選択したアプリケーションインスタンスのログメッセージが表示されます。

[Live Log] パネルでは、次の作業を実行できます。

- 特定の文字列を検索するには、[Find] テキストボックスに検索する文字列 (またはその一部) を入力し、[Find Next] をクリックします。[Find Next] をクリックすると、文字列の検索を続行できます。
- ログの前方に出現する文字列を検索するには、[Find Previous] をクリックします。
- ログのすべてのメッセージを削除するには、[Clear Log] をクリックします。
- アプリケーションが使用するサーバーサイドスクリプトの 1 つを変更したなどの理由からアプリケーションインスタンスをリロードする必要がある場合、あるいはすべてのユーザー接続を解除する場合には、[Reload this application] (パネル名の右側にある小さなボタン) をクリックします。
- アプリケーションインスタンスを削除するには、[Unload this application] (パネル名の右側にある小さなボタン) をクリックします。
- Management Console とサーバー間の接続を解除するには、[Logoff] をクリックします。
- 別のパネルを表示するには、該当するパネルの名前 (Clients、Shared Objects、Streams、または Performance) をクリックします。

[Client] パネルについて

[Clients] パネルには、アプリケーションに接続しているすべてのクライアントの詳細情報として、サーバーが生成した Client ID、接続プロトコル、送受信されたバイト数、送受信されたメッセージ数、欠落したメッセージ数などが表示されます。



[Clients] パネルには、選択したアプリケーションインスタンスに接続しているクライアントに関する情報が表示されます。

[Clients] パネルに表示される情報は次のとおりです。

Client ID は、クライアントの内部 ID を表します。この ID は、サーバーが生成した数値で、Flash Media Server が各クライアントを識別するために使用します。

Protocol は、RTMP などの、クライアントが使用している接続プロトコルです。

Bytes In and Bytes Out は、サーバーとの間で 1 秒間に送受信されたバイト数の平均です。Management Console は、直前の 15 秒間に受け取った合計バイト数を 15 で割ることで、この値を算出します。パネルが最初に表示されたときには、開始点からのデータが 1 つあるだけなので、ここには `ending` と表示され、パネルを開いた 15 秒後に数値が表示されます。

Connection Time は、クライアントが接続した日時です。

Messages In and Messages Out は、クライアントの間に送受信されたメッセージの数です。"messages in" にはクライアントからサーバーに送られた更新要求が反映され、"messages out" にはサーバーから接続しているクライアントに送られた更新成功の通知が反映されます。

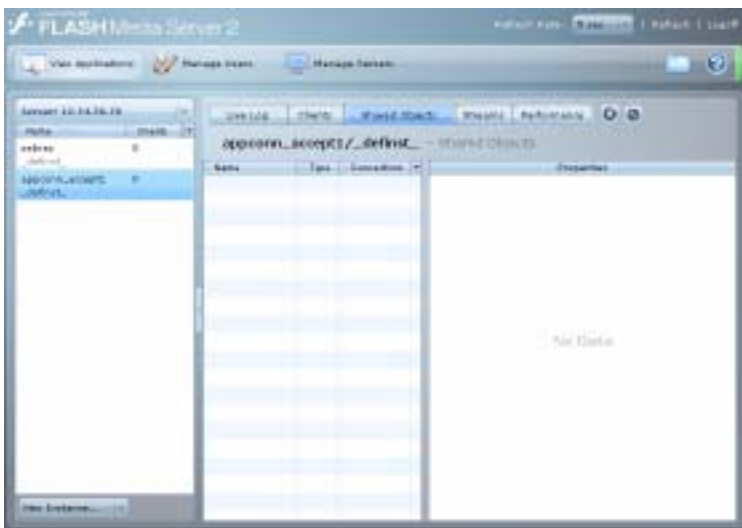
Drops は、クライアントが接続した以降に欠落したメッセージの数です。ライブのストリーム、オーディオ、およびビデオの場合にはメッセージが欠落する可能性があり、記録されたストリームの場合にはビデオメッセージだけが欠落します。コマンドメッセージが欠落することはありません。

このパネルでは、次の作業を実行できます。

- このパネルをカスタマイズするには、右端の列のポップアップメニューをクリックし、表示する列を選択します。
- アプリケーションが使用するサーバーサイドスクリプトの 1 つを変更したなどの理由からアプリケーションインスタンスをリロードする必要がある場合、あるいはすべてのユーザー接続を解除する場合には、[Reload this application] (パネル名の右側にある小さなボタン) をクリックします。
- アプリケーションインスタンスを削除するには、[Unload this application] (パネル名の右側にある小さなボタン) をクリックします。
- Management Console とサーバー間の接続を解除するには、[Logoff] をクリックします。
- 別のパネルを表示するには、該当するパネルの名前 (Live Log、Shared Objects、Streams、または Performance) をクリックします。

[Shared Objects] パネル

[Shared Objects] パネルには、アプリケーションインスタンスが使用しているすべてのアクティブな共有オブジェクトに関する情報が表示されます。この情報は、5 秒ごとに自動的に更新されます。即時更新するには [Refresh] をクリックします。



[Shared Objects] パネルには、アプリケーションインスタンスが使用しているアクティブな共有オブジェクトに関する詳細情報の一覧が表示されます。

共有オブジェクトには、リモートで永続的に存在するもの（サーバー上に保管され、アプリケーションインスタンスまたはサーバーが停止して再起動した後も使用可能なもの）と、アプリケーションインスタンスが有効である間だけ使用可能なもの（一時的に存在し、アプリケーションインスタンスの実行が停止すると削除されるもの）があります。[Shared Objects] パネルには、アプリケーションインスタンスが使用している両方のタイプの共有オブジェクトに関する情報が表示されます。

特定の共有オブジェクトに関する情報を表示するには、そのオブジェクトをクリックして選択します。この操作によって、パネルには次の情報が表示されます。

Name は、共有オブジェクトの名前です。

Type は、共有オブジェクトのカテゴリで、Stored あるいは Temp というように表示されます。

Connections は、現在、この共有オブジェクトに接続し、使用しているユーザーの数です。

Properties は、共有オブジェクトに割り当てられている data プロパティを表します。

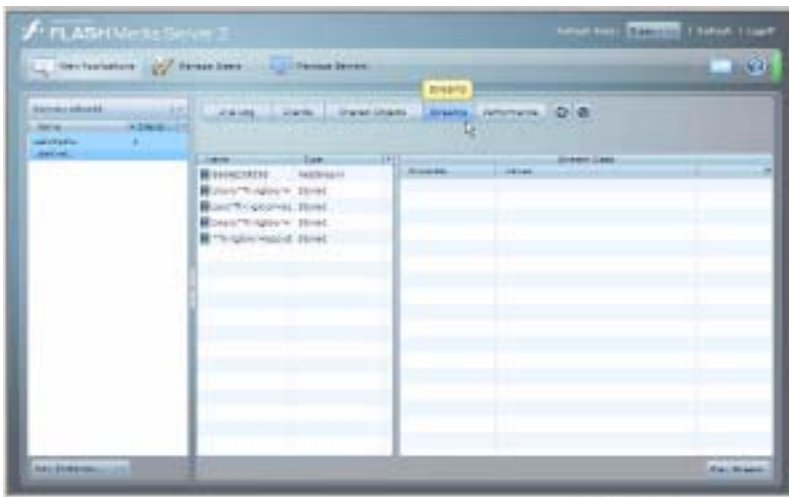
詳細については、『サーバーサイド ActionScript リファレンスガイド』の「SharedObject.resyncDepth」の項目を参照してください。

[Shared Objects] パネルでは、次の作業を実行できます。

- このパネルをカスタマイズするには、左の表の右端の列のポップアップメニューをクリックし、表示する列を選択または選択解除します。
- 共有オブジェクトからデータを取得するには、デバッグ接続を確立する必要があります。詳細については、[78 ページの「デバッグ接続の使用」](#)を参照してください。)
- アプリケーションが使用するサーバーサイドスクリプトの 1 つを変更したなどの理由でアプリケーションインスタンスをリロードする必要がある場合、あるいはすべてのユーザー接続を解除する場合には、[Reload this application] (パネル名の右側にある小さなボタン) をクリックします。
- アプリケーションインスタンスを削除するには、[Unload this application] (パネル名の右側にある小さなボタン) をクリックします。
- Management Console とサーバー間の接続を解除するには、[Logoff] をクリックします。
- 別のパネルを表示するには、該当するパネルの名前 (Live Log、Clients、Streams、または Performance) をクリックします。

[Streams] パネル

[Streams] パネルを使用すると、ストリームに関する情報の表示とストリームの再生を行えます。この情報は、5 秒ごとに自動的に更新されます。即時更新するには [Refresh] をクリックします。



[Streams] パネルでは、ストリームに関する詳細情報を表示し、ストリームを再生することができます。

[Streams] パネルには、情報が次のように表示されます。

Name は、ストリーム名または NetStream ID です。NetStream ストリームの場合には、この名前は NetStream ID (サーバーが生成する番号) です。パブリッシュされるライブストリームの場合には、ライブストリーム名が表示されます。記録されたストリームの場合には、flv/mp3 のファイル名が、たとえば flv:stream2.flv や mp3:sound.mp3 というように表示されます。クライアントからの stream2.flv の要求では、2 つのエントリ、すなわち、flv:stream2.flv (Stored) 用のエントリと、クライアントに送られる実際のネットストリームで netstream ID 名とタイプ (NetStream) であるエントリがあります。

Type は、サーバーから提供された、このストリームで何が起こるかを表す文字列で、Stored、Live、NetStream のいずれかです。

Properties は、Name、Status、Client、および Time です。

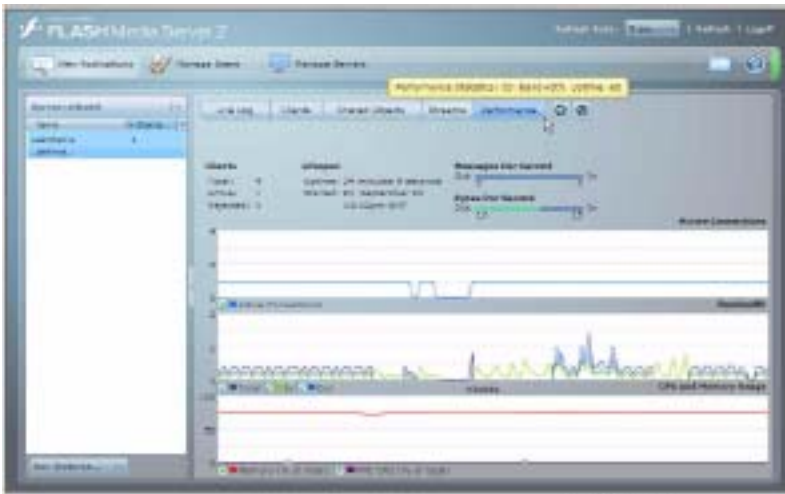
Values は、各プロパティに対する値で、Name (streamID ではなく、実際のストリーム名)、Status (publishing、playing live、playing recorded のいずれか)、Client (ストリームを再生するクライアント ID)、および Time (クライアントがストリームの再生を開始した時刻) です。

[Strames] パネルでは、次の作業を実行できます。

- このパネルをカスタマイズするには、ポップアップメニューをクリックし、表示する列を選択あるいは選択を解除します。
- [Play Stream] をクリックすると、選択したストリームが、そのストリームのサイズの別ウィンドウで再生されます。(選択したストリームを再生するには、デバッグ接続を確立する必要があります。詳細については、78 ページの「デバッグ接続の使用」を参照してください。)再生できるのは、名前付きのストリームだけです。
- アプリケーションが使用するサーバーサイドスクリプトの 1 つを変更したなどの理由でアプリケーションインスタンスをリロードする必要がある場合、あるいはすべてのユーザー接続を解除する場合には、[Reload this application] (パネル名の右側にある小さなボタン) をクリックします。
- アプリケーションインスタンスを削除するには、[Unload this application] (パネル名の右側にある小さなボタン) をクリックします。
- Management Console とサーバー間の接続を解除するには、[Logoff] をクリックします。
- 別のパネルを表示するには、該当するパネルの名前 (Live Log、Clients、Shared Objects、または Performance) をクリックします。

[Performance] パネルについて

[Performance] パネルには、選択したアプリケーションインスタンス全体の状態に関する情報が表示されます。この情報は、5 秒ごとに自動的に更新されます。即時更新するには [Refresh] をクリックします。



[Performance] パネルには、選択したアプリケーションインスタンスに関する詳細情報が表示されます。

[Performance] パネルに表示される情報は次のとおりです。

Clients は、このアプリケーションインスタンスに接続しているクライアントに関する情報として、開始時以降にこのアプリケーションインスタンスに接続したクライアントの合計数、アクティブなクライアント、このアプリケーションインスタンスに接続しようとして拒否されたユーザー数などが表示されます。(接続が失敗した理由を特定するには、Management Console の [Live Log] パネルを調べます。)

Lifespan は、アプリケーションインスタンスのこれまでの実行時間と実行が開始した日時です。

Messages Per Second は、サーバーとの間で送受信されたメッセージ (カメラのビデオフレーム、オーディオパケット、およびコマンドのメッセージ) の平均数です。

Bytes per second は、このアプリケーションインスタンスのためのサーバーとの間で送受信された平均バイト数です。Management Console は、直前の 15 秒間に受け取った合計バイト数を 15 で割ることで、この値を算出します。パネルが最初に表示されたときには、開始点からのデータが 1 つあるだけなので、ここには “pending” と表示され、パネルを開いた 15 秒後に数値が表示されます。

Active Connections は、現在、このアプリケーションインスタンスに接続しているユーザー数です。

Bandwidth は、アプリケーションインスタンスが処理したデータ量で、これには、送信したデータ、受信したデータ、およびデータトラフィックの合計などが表示されます。

CPU and Memory Usage は、Flash Media Server に割り当てられた CPU とメモリの比率です。

[Performance] パネルでは、次の作業を実行できます。

- チェックボックスの選択および選択解除によって、グラフに表示される情報をカスタマイズできます。たとえば、Bandwidth グラフでは、[Total] を選択して [In] と [Out] の選択を解除することで、使用された帯域幅の合計だけが表示されます。
- アプリケーションが使用するサーバーサイドスクリプトの 1 つを変更したなどの理由でアプリケーションインスタンスをリロードする必要がある場合、あるいはすべてのユーザー接続を解除する場合には、[Reload this application] (パネル名の右側にある小さなボタン) をクリックします。
- アプリケーションインスタンスを削除するには、[Unload this application] (パネル名の右側にある小さなボタン) をクリックします。
- Management Console (管理コンソール) とサーバー間の接続を解除するには、[Logoff] をクリックします。
- 別のパネルを表示するには、該当するパネルの名前 (Live Log、Clients、Shared Objects、または Streams) をクリックします。

デバッグ接続の使用

ストリームの再生と共有オブジェクトからのデータの取得では、Management Console からサーバーへの特別なデバッグ接続が必要になります。デフォルトでは、サーバーはこの接続を許可しないため、Application.xml を変更する必要があります。

メモ	デバッグ接続は、一般のユーザーとして、ライセンス数を上限としてカウントされます。
----	--

デバッグ接続を有効にする手順は次のとおりです。

- Application.xml ファイルで、`<allowDebugDefault>true</allowDebugDefault>` をセットします。

Application.xml にこの変更を加えると、[Shared Object] パネルで共有オブジェクトをクリックするか、[Streams] パネルで [Play Stream] をクリックしたときに、Management Console はサーバーに対してデバッグ接続を確立します。デバッグ接続が確立されている間は、ユーザーがクリックした共有オブジェクトは Management Console にサブスクライブされるため、データは直ちに更新されます。また、デバッグ接続が確立されると、[Close Debug] という名前のボタンが表示され、このボタンをクリックするとデバッグ接続を閉じることができます。

onStatus イベントハンドラ

クライアントサイドの Camera、Microphone、NetConnection、NetStream、および SharedObject の各クラスと、サーバーサイドの Application、NetConnection、Stream、および SharedObject の各クラスには、情報、ステータス、またはエラーメッセージの提供のために情報オブジェクトを使用する onStatus イベントハンドラが用意されています。このイベントハンドラを活用するためには、引数として与えられる情報オブジェクトの形式と内容を把握し、それを処理する必要があります。

デフォルトでは、各情報オブジェクトには、onStatus メソッドの結果を表す code プロパティと、"status"、"warning"、"error" のいずれかが格納される level プロパティがあります。これ以外にもデフォルトのプロパティがある情報オブジェクトもあり、それらのプロパティには、onStatus が呼び出された理由に関する詳細情報が格納されます。

onStatus ハンドラが返す値に関する詳細については、『Flash Media Server 2 クライアントサイド ActionScript リファレンスガイド』を参照してください。次の表は、特定のサーバーサイド呼び出しが返す情報オブジェクトをまとめたもので、左側のサーバーサイド呼び出しによって、右側のコード値でクライアントサイドの NetConnection.onStatus ハンドラが呼び出されます。(この情報は、『Flash Media Server 2 クライアントサイド ActionScript リファレンスガイド』には記載されていません。)

サーバーサイド呼び出し	クライアントサイド NetConnection.onStatus ハンドラに対するコード値
application.acceptConnection()	NetConnection.Connect.Success
application.rejectConnection()	NetConnection.Connect.Rejected
application.disconnect()	NetConnection.Connect.Closed

次のセクションでは、この機能を最も効率的に実装するための推奨事項をさらに紹介します。

スクリプト内で onStatus ハンドラを記述する場所

ネットワークとスレッドのタイミングの問題から、onStatus ハンドラを記述する最適な場所は、スクリプト内の connect メソッドの前です。そうしておかないと、スクリプトが onStatus ハンドラの初期化を実行する前に完了してしまうことがあります。また、セキュリティのチェックはすべて connect メソッドの中で行われ、onStatus ハンドラがまだセットアップされていない場合には通知が失われます。

onStatus ハンドラの無効化について

必要ではない部分が onStatus ハンドラに含まれている場合であっても、オブジェクトを作成するたびに、通常の onStatus ハンドラを無効化するのがいいでしょう。その方法については、次の例を参照してください。

```
// すべてのステータス情報をトレースする関数を作成
function traceStatus(info) {
    trace("Level:" + info.level + " Code:" + info.code);
}
// この関数を、オブジェクトの作成時に onStatus ハンドラに割り当てる
var my_nc:NetConnection = new NetConnection();
my_nc.onStatus = traceStatus;
var my_ns:NetStream = new NetStream(my_nc);
my_ns.onStatus = traceStatus;
```

アプリケーションを開発し、特定の目的のために実際にハンドラを無効にする必要があると判断したら、traceStatus() 関数を onStatus ハンドラに割り当てるコードを変更することができますが、その場合であっても、少なくともアプリケーションに関連するすべてのメッセージが表示されます。クライアントとサーバーの両方でのステータスメッセージを必ずチェックします。

System.onStatus ハンドラについて

78 ページの「onStatus イベントハンドラ」のリストに記載されているクラスに提供される固有の onStatus メソッドに加えて、Flash には、System.onStatus と呼ばれる "super ハンドラ" もあります。特定のオブジェクトに対して level プロパティが Error で onStatus が呼び出され、それに応答する関数が割り当てられていない場合は、System.onStatus に割り当てられた関数があれば、その関数が実行されます。

次のコード例では、onStatus ハンドラに割り当てられていないオブジェクトに対してエラーメッセージが発生すると、[Output] パネルにメッセージを送信します。

```
System.onStatus = function(genericError) {  
    // 実際のスクリプトではもっと内容のあることを実行します。  
    trace("An error has occurred.Please try again.");  
}
```

NetConnection.Connect.Failed メッセージのデバッグについて

NetConnection.connect() コマンドから NetConnection.Connect.Failed の code 値で情報オブジェクトが返されると、サーバーとの接続を確立することはできません。このエラーを受け取るたびに、次のようなトラブルシューティングのための標準的な事項を確認してください。

- 接続しようとするサーバーは正しいか？
- サーバーは稼働しているか？
- サーバーの接続のためのプロトコル (rhmp:) を指定しているか？
- ファイアウォールの裏側から接続する場合には、正しいプロトコル (rtmpt:) を指定しているか？

クライアントまたはサーバーで許可されているソケット接続の数が上限に達した場合にも、サーバーへの接続の試行が失敗する可能性があります。この上限は、サーバー上の物理メモリの大きさや、接続のビジー状況によって異なります。Windows システムにおいては、ソケット接続のためのメモリはページングされるメモリの外に割り当てられるため、ページファイルにスワップアウトできません。稼働中の他のプログラム (オペレーティングシステムのコアサービスなど) との間でページングされないメモリアールの領域の競合が発生すると、上限値は時間の経過につれて異なる値になります。

オブジェクトのプロパティのトレース

特定のオブジェクトで問題が発生する理由を突き止めるために、次のように、プロパティを繰り返して、調べることができます。

```
for (i in my_obj) {  
    trace(i + " = " + my_obj[i]);  
}
```


この章では、Macromedia Flash Media Server に特有の、アプリケーション開発に関する役立つ情報を紹介します。アプリケーション設計に関する一般的な説明のセクションと、コーディング規則に関するセクションから構成されています。第 1 章と第 2 章に説明されている概念を理解してから、この章の説明を読んでください。

ファイルタイプとパスについて

オーディオ、ビデオ、またはデータストリームを記録するためのメソッド（たとえば、`NetStream.publish()`）を使用すると、Flash Media Server は、`filename.flv` (`filename` はストリームを記録したメソッドから渡された文字列）という名前のファイルを作成します。これらのファイルは、記録されたストリームとそのインデックスです。たとえば、コマンド `NetStream.publish("me", "record")` を実行すると、`me.flv` という名前のファイルが作成されます。

Flash Media Server は、アプリケーション用のストリームファイルの生成にあたって、アプリケーションインスタンスのサブディレクトリを持つストリームディレクトリを作成します。たとえば、アプリケーションインスタンス `ChatApp/MondayChat` が `chat` という名前のストリームを記録する場合は、`chat.flv` ファイルが `/applications/ChatApp/streams/MondayChat` に作成されます。`ChatApp` のインスタンスが `TuesdayChat` という名前であるとすると、そのファイルは `/applications/ChatApp/streams/TuesdayChat` に保存されます。

Sorenson Squeeze や On2 のような特別なビデオアプリケーションによって作成された FLV ファイルを再生するには、Flash Media Server が探しにくいディレクトリ、すなわち、前の段落で説明した `/streams` ディレクトリにファイルを置きます。アプリケーションインスタンスの正しいサブディレクトリの中にさらにサブディレクトリを作成して、FLV ファイルをさらに整理することができます。たとえば、`/streams/TuesdayChat` の中にさらに `/streams/TuesdayChat/morning` と `streams/TuesdayChat/afternoon` というサブディレクトリを作成して、それらの中に FLV ファイルを格納できます。

MP3 オーディオファイルの再生については、[第 4 章の「メディアファイルの使用」](#)を参照してください。

ストリームの上書きを防止するために、ユーザーやストリームなどには一意の名前を使用します。たとえば、新しいストリームを記録する場合は、増分させた値を使用するといでしょう。

```
outStream.publish("myRecording" + numSnaps, "record");
```

記録されたストリームファイルの削除については、[57ページの「Stream クラス」](#)を参照してください。インスタンス名とファイルの格納に関する詳細については、『Flash Media Server 2 クライアントサイド ActionScript リファレンスガイド』の「[NetConnection.connect\(\)](#)」と「[NetStream.publish\(\)](#)」の項目を参照してください。

メモ	このセクションでは、アプリケーション名と一緒にインスタンス名を渡すことを前提にしています (15 ページの「アプリケーションとアプリケーションインスタンスのデプロイメント」 を参照)。インスタンス名を渡さないと、Flash Media Server はファイルを登録アプリケーションのディレクトリ内の <code>_definst_ (for "default instance")</code> というサブディレクトリに格納します。
----	---

共有オブジェクトファイルについて

ローカルあるいはリモートのどちらであっても、共有オブジェクトは (アプリケーションインスタンスが存在する間だけ) 動的に存在することも、永続的なデータとして使用するために保存することもできます。このセクションでは、ユーザーが作成できる 3 種類の永続的な共有オブジェクト、すなわち、永続的なローカル共有オブジェクト、サーバー上にのみ永続的に存在するリモート共有オブジェクト、およびクライアント上とサーバー上に永続的に存在するリモート共有オブジェクトについて説明します。(共有オブジェクトに関する全般的な情報については、[43ページの「共有オブジェクトの概要」](#)を参照してください。)

ローカル共有オブジェクトは、メモリとディスクの領域が利用できる限り、クライアント上に常に永続的に存在します。ただし、デフォルトでは、Flash がローカルに保存できるリモート共有オブジェクトのサイズの上限は 100 K です。それより大きいオブジェクトを保存しようとすると、Flash Player から [ローカル記憶領域] ダイアログボックスが表示され、アクセスが要求されているドメインのローカルへのオブジェクトの保存をユーザーが許可または拒否することができます。また、ユーザーはこのダイアログボックスを使用して、ローカルに永続的に存在するすべてのリモート共有オブジェクトを削除することもできます。詳細については、『クライアントサイド ActionScript リファレンスガイド』の「SharedObject」の項目にある「ローカルのディスクスペースに関して」の説明を参照してください。

永続的なローカル共有オブジェクト

永続的なローカル共有オブジェクトは、クライアントサイドの `SharedObject.getLocal()` コマンドを使用して作成します。永続的なローカル共有オブジェクトの拡張子は `.sol` です。`SharedObject.getLocal()` コマンドの `localPath` パラメータに値を渡すことで、オブジェクトを保存するディレクトリを指定することができます。

リモートに存在する永続的な共有オブジェクト

サーバー上でのみ永続的に存在するリモート共有オブジェクトは、クライアントサイドの `SharedObject.getRemote()` コマンドまたはサーバーサイドの `SharedObject.get()` コマンドで `persistence` パラメータに値 `true` を渡すことで作成できます。これらの共有オブジェクトの拡張子は `.fso` で、サーバー上で、共有オブジェクトを作成したアプリケーションのサブディレクトリに保存されます。Flash Media Server がこれらのディレクトリを自動的に作成するため、ユーザーがそれぞれのインスタンス名のディレクトリを作成する必要はありません。

リモートとローカルに永続的に存在する共有オブジェクト

クライアント上とサーバー上に永続的に存在するリモート共有オブジェクトは、クライアントサイドの `SharedObject.getRemote()` コマンドの `persistence` パラメータにローカルパスを渡すことで作成します。ローカルに永続的に存在する共有オブジェクトの拡張子は `.sor` で、クライアント上の指定されたパスに保存されます。リモートに永続的に存在する共有オブジェクトの拡張子は `.fso` で、共有オブジェクトを作成したアプリケーションのサブディレクトリに保存されます。

ローカルまたはリモートに永続的に存在する共有オブジェクトに対するパスを一部だけ指定することで、同じドメインの複数のアプリケーションが同じ共有オブジェクトにアクセスすることができます。詳細については、『Flash Media Server 2 クライアントサイド ActionScript リファレンスガイド』の「`SharedObject.getRemote()`」を参照してください。

サーバー間での移植性

アプリケーションの開発にあたっては、開発用の設定と本番用の設定を考慮することが重要です。

アプリケーションエレメントには、小文字テキストのみの名前を付ける。Flash Media Server とそのアプリケーションが使用するディレクトリやファイルに名前を付ける場合は、小文字のみを使用し、間にスペースを入れないようにすることが推奨されます。そうすることで、開発中に異なるプラットフォームの異なるコンピュータに移動することになっても、アプリケーションを正常に動作させることができるからです。

相対パスと絶対パスのどちらを使用してサーバーに接続するか。SWF ファイルの `NetConnection.connect()` ステートメントでは、相対パスか絶対パスのどちらかを使用して、Flash Media Server ディレクトリの登録アプリケーションディレクトリに接続することができます。スラッシュ 1 つ (`/`) の後にアプリケーション名を記述すると、相対パスを表し、この方法では、コードを変更することなく、異なるサーバーにファイルを移動することができます。スラッシュ 2 つ (`//`) は絶対パスを表します。

Flash Media Server が稼動しているのと同じコンピュータ上に SWF ファイルがある場合には、`rtmp://server.domain.com/appName/instanceName` のショートカット版として `rtmp://appName/instanceName` を使用することができます。開発サーバーでアプリケーションをオーサリングし、SWF ファイルで相対パスを使用していない場合には、SWF ファイルの `NetConnection.connect()` ステートメントに記述されているすべての URI (Uniform Resource Identifier) を開発サーバーから本番サーバーへの接続に変更する必要があるため、さらに作業が必要です。

相対パスを使用していれば、Flash Player は、SWF ファイルをホスティングするコンピュータは Flash Media Server で、SWF ファイルが含まれているディレクトリは正しいアプリケーションディレクトリであると想定します。開発サーバーと本番サーバーの両方でアプリケーションが同じ名前になっている限り、URI を編集する必要はありません。

SWF ファイルが Flash Media Server と同じコンピュータ上に存在しない場合には絶対パスが必要になり、ファイルを本番環境に移動する場合には編集が必要になります。

クライアントサーバーのスキプトの相互依存性

クライアントサイドとサーバーサイドの ActionScript コードは同じアプリケーションの一部であるため、相互に関連し合いながら動作する必要があります。クライアントのコードとサーバーのコードの間の相互依存性を示す例として、サーバーサイドの ActionScript `call()` メソッドがありますが、このメソッドは、クライアントサイドの `NetConnection` オブジェクトとサーバーサイドの `Client` オブジェクトのどちらに関連付けられているかによって動作が異なります。

たとえば、次のコード例は、クライアントサイドでネットワーク接続を作成してから、サーバーサイドでそれを呼び出す方法を示しています。

```
// これは、FLA ファイル内のクライアントサイド ActionScript
var my_nc:NetConnection = new NetConnection();
my_nc.someClientMethod = function()
{
    // ここにコードを記述
}
my_nc.connect("rtmp://hostname:port/appname");
```

```
// これは、main.asc ファイル内のサーバーサイド ActionScript
clientObj.call("someClientMethod");
```

`clientObj.call()` に渡すパラメータは、クライアントサイドの `NetConnection` オブジェクトで前もって定義されたメソッドである必要があります。これは、サーバーから呼び出されるクライアントコード内のどのメソッドも、クライアントサイドの `NetConnection` オブジェクトのプロパティでなければならないためです。

反対に、クライアントで `call()` メソッドを使用してサーバーサイドのメソッドを呼び出す場合を考えてみましょう。

```
// これは、FLA ファイル内のクライアントサイド ActionScript である
NetConnection.call("someServerMethod");
// これは、main.asc ファイル内のサーバーサイド ActionScript である
client.prototype.someServerMethod = function(){
    // ここにコードを記述
}
// 次のコードも使用可能
application.onConnect = function(newClient){
    newClient.someServerMethod = function(){
        // コード
    }
    return true;
}
```

この場合は、`NetConnection.call()` に渡されるパラメータは、`main.asc` ファイルに前もって定義されている `Client` オブジェクトのメソッドである必要があります。これは、クライアントから呼び出されるサーバーコード内のどのメソッドも、サーバーサイドの `NetConnection` オブジェクトのプロパティでなければならないためです。

複数のスクリプトファイルの使用

アプリケーションが複雑になる場合には、1つのスクリプトですべてを処理するのではなく、機能を複数のスクリプトに分割し、各スクリプトに個別の機能を持たせるようにしてください。複数のファイルを使用する場合は、"一度だけのインクルード" 定義を使用することによってファイルが何回も評価されるのを回避し、パフォーマンスを最適化することができます。

たとえば、クライアントサイドで `my_file.as` という名前のファイルを作成し、アプリケーション内の他のファイルでも使用できるようにするには、`my_file.as` に次のコードを記述します。

```
if (_root._my_file_as == null) {
    _root._my_file_as = true;
    // myfile.as のすべてのコードをここに記述
}
```

他のファイルでは、次のコマンドを実行します。

```
#include "my_file.as";
```

メモ

`#include` は、実行時ではなく、コードのコンパイル時に処理されます。

サーバーサイドでは、多少異なるコードを実装する必要があります。サーバーサイド ActionScript には、グローバルオブジェクトは含まれません。次のコードでは、my_file.asc という名前のファイルをインクルードしています。

```
if (_my_file_asc == null) {  
    _my_file_asc = true;  
    // myfile.asc のすべてのコードをここに記述  
}
```

さらに、サーバーサイドのスクリプトでは、#include コマンドの代わりに load コマンドを次のように使用します。

```
load("my_file.asc");
```

サーバーサイドのスクリプトファイルのアーカイブとコンパイル

Flash Media Server には、コマンドラインのアーカイブコンパイラユーティリティ (far.exe) が含まれています。このユーティリティを使用すると、サーバーサイドのすべてのスクリプトを ZIP ファイルによく似た 1 つのアーカイブファイルにパッケージ化することができるため、デプロイメントが容易になります。また、このアーカイブコンパイラユーティリティ (FAR と呼びます) を使用して、サーバーサイドのスクリプトファイルをバイトコード (拡張子は .ase) にコンパイルできるため、アプリケーションインスタンスのロードに要する時間を短縮することができます。

サーバーサイドスクリプトのアーカイブ

大規模なアプリケーションでは、異なる場所に格納されている複数のサーバーサイドのスクリプトファイルを使用することができます。いくつかのファイルはアプリケーションディレクトリに置かれ、他のファイルはサーバーの設定ファイルに定義されているスクリプトライブラリのパスに置かれます。メディアアプリケーションのデプロイメントを簡素化するためには、サーバーサイドの .js、ファイル、.asc ファイル、および .ase ファイルを Flash Media Server アーカイブファイル (.far) にパッケージ化します。

FAR ファイルは、メインのスクリプトファイル (main.js、main.asc、main.ase、<appname>.js、<appname>.asc、<appname>.ase のいずれか) と、そのメインのスクリプトで参照されるスクリプトファイルがあればそれも含まれているパッケージです。

サーバーサイドのスクリプトを1つのアーカイブファイルにパッケージ化する手順は次のとおりです。

1. オペレーティングシステムのコマンドシェルを開きます。
2. プロンプトに、`far -package` と入力し、必要なオプションも併せて指定します。

メモ	メインのスクリプトがサブディレクトリ内のスクリプトを参照している場合は、アーカイブファイル内の階層を維持する必要があります。この階層を維持するには、メインのスクリプトが置かれているディレクトリで FAR コマンドを実行することをお勧めします。
----	---

アーカイブコンパイラユーティリティを実行してスクリプトパッケージを作成するためのシNTAXは、次のとおりです。

```
c:\> far -package -archive <archive> -files <file1> [<file2> ... <fileN>]
```

次の表は、`-package` コマンドで使用できるコマンドラインオプションの説明です。

オプション	説明
<code>-archive <archive></code>	アーカイブファイルのファイル名を指定します。アーカイブファイルの拡張子は <code>.far</code> です。
<code>-files <file1> [<file2> ... <fileN>]</code>	アーカイブファイルに入れるファイルのリストを指定します。このオプションには、1つ以上のファイルを指定します。

FAR ファイル

FAR ファイルは、アプリケーション内の 2 箇所、すなわち、アプリケーションディレクトリか `scriptlib` パスに置くことができます。

FAR ファイルをアプリケーションディレクトリに置く場合は、FAR ファイルの名前を `main.far` または `<appname>.far` (`<appname>` はアプリケーションディレクトリの名前) にする必要があります。

たとえば、アプリケーションが `C:/samples/foo` に置かれているとすると、このアプリケーションフォルダのすべての必要なファイルを `main.far` または `<appname>.far` にパッケージ化し、`C:/samples/foo` に置きます。サーバーは、アプリケーションのロード時に、`main.far` ファイルまたは `<appname>.far` ファイルをアプリケーションフォルダに探しにいきます。FAR ファイルが見つからないと、サーバーは、アプリケーションフォルダ内にメインのスクリプトを探しにいきます。

FAR ファイル内のスクリプトが `scriptlib` パス内の他のスクリプトを参照している場合は、このパスの必要とするすべてのファイルを FAR ファイルに入れておくことをお勧めします。このようにしておくことで、異なるコンピュータ上にある、同じ名前の異なるバージョンのアプリケーションがロードされるのを防ぐことができます。スクリプトが FAR ファイル内のスクリプトファイルを参照していると、サーバーは常に、FAR ファイル内を探してから、アプリケーションディレクトリ内を探します。メインのスクリプトで他のスクリプトを参照している場合は、FAR ファイルにおいてもそのパスを維持する必要があります。

アプリケーションでの FAR ファイルの使用方法について理解するために、testFAR という名前のアプリケーションを基準にした次の例を参考にしてください。次の例は、testFAR アプリケーションのディレクトリです。



メインのスクリプトは testFAR.asc という名前で、scripts ディレクトリに置かれています。



testFAR.asc ファイルは、スクリプトライブラリディレクトリ (<installdir>\scriptlib) に提供されている Web サービスを使用しています。testFAR の開発者は、アプリケーションがどこに配置されるかはわからないため、すべての ASC ファイルを webServices フォルダに格納します。そうすることで、メインのスクリプトはいつでも Web サービスを探すことができます。WebServices.asc ファイルがメインのスクリプトからロードされると、load("webServices/WebServices.asc") が呼び出されます。FAR ファイルと同じ階層を維持することが重要で、そのためには、webServices フォルダを <installdir>\scriptlib からコピーします。そうすることで、次の例にあるように、このフォルダがメインのスクリプトと同じレベルになります。

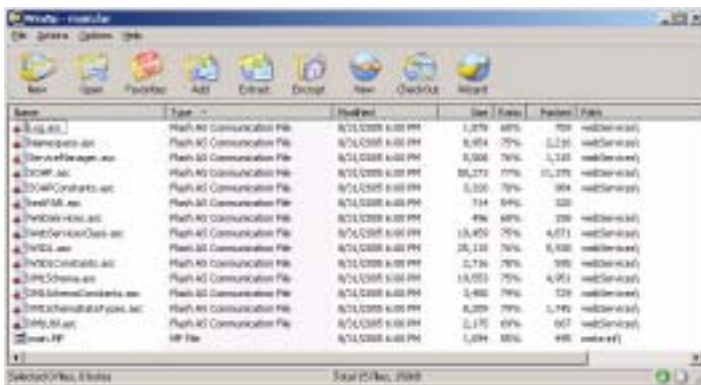


これで、FAR ファイルにパッケージ化する準備が整いました。コマンドシェルでアーカイブコンパイルユーティリティを開き、次のコマンドを実行します。

```
far -package -archive main.far -files testFAR.asc webServices\*.asc
```

最後に、main.far ファイルのアプリケーションディレクトリの scripts フォルダを置き換えます。

FAR ファイルの内容を調べるには、WinZip でこのファイルを開きます。重要なのは、FAR ファイル内のスクリプトのパスが、メインのスクリプトが正しくスクリプトを参照できるようにセットされていることを確認することです。testFAR の例では、解凍した FAR ファイルは次のようになります。



Flash Media Server の scriptlib パスにも FAR ファイルを入れることができます。scriptlib パスは、Flash Media Server のインストールディレクトリの下での fcs.ini ファイルに定義されています。この場合には、FAR ファイルは、ディレクトリからスクリプトファイルを探す代わりに、ディレクトリであるかのように動作します。Flash Media Server は、この FAR ファイル内にスクリプトファイルを探しにいきます。

scriptlib は、次の例にあるように、fcs.ini ファイルに定義されます。

```
APP.JS_SCRIPTLIBPATH = C:\Program Files\Macromedia\Flash Communication Server  
MX\scriptlib
```

scriptlib パスに FAR ファイルを入れるには、次のコードを使用します。

```
APP.JS_SCRIPTLIBPATH = C:\Program Files\Macromedia\Flash Communication Server  
MX\scriptlib;C:\FMS\myLibrary.far
```

スクリプトのバイトコードへのコンパイル

アプリケーションインスタンスのロードに必要な時間を短縮するには、FAR ユーティリティを使用して、1つ以上の ASC ファイルまたは JS ファイルをバイトコードにコンパイルします。

入力ファイルと同じ場所に同じ名前で作成ファイルが作成されますが、ファイル拡張子は .asc または .js から .ase に置き換わります。入力ファイルの拡張子が .asc または .js ではない場合は、ファイル名に .ase という拡張子が付加されます。

スクリプトをバイトコードにコンパイルする手順は次のとおりです。

1. オペレーティングシステムのコマンドシェルを開きます。
2. プロンプトに、`far -compile` と入力し、必要なオプションも併せて指定します。

メモ	FAR ユーティリティの場所を PATH 環境変数に追加します。こうすることで、実行可能ファイルの完全パスを指定することなく、FAR ユーティリティを実行することができます。
----	---

アーカイブコンパイラユーティリティを実行してスクリプトパッケージを作成するためのシンタックスは、次のとおりです。

```
c:\> far -compile [-v <version>] [-fv <version>] [-e <date>] [-hdr <header>]
      [-n] [-h [<command>]] <file1> [<file2> ... <fileN>]
```

次の表は、`-bytecode` コマンドで使用できるコマンドラインオプションの説明です。

オプション	説明
<code>-v <version></code>	オプション。12 バイトの署名の <code>version</code> フィールドにエンコードされるバージョンを指定します。指定しないと、このオプションのデフォルトは 1.0 になります。このオプションは、 <code>"version:1.0\n"</code> というように、オプションのヘッダーとしても追加されます。
<code>-fv <version></code>	オプション。このファイルで使用する Flash Media Server の推奨バージョンを指定します。このバージョン番号は、36 バイトのヘッダーの <code>fms_version</code> フィールドにエンコードされます。指定しないと、デフォルトは 1.6 になります。このオプションは、 <code>"fcsver:1.6\n"</code> というように、オプションのヘッダーとしても追加されます。 メモ：このオプションにより、ASE ファイルにはコンテンツが追加されますが、Flash Media Server がこの時点でこのコンテンツを使用することはありません。
<code>-e <date></code>	オプション。このファイルの有効期限を指定します。 <code>mm/dd/yyyy</code> という形式の日付で指定します。この日付を過ぎると、ファイルを読み取ることはできなくなります。指定しないと、このオプションのデフォルトは 0 で、有効期限は設定されません。試用期間をファイルに設けたい場合には、このオプションが有効です。(たとえば、 <code>12/31/2003</code> というように) 設定すると、この日付はオプションのヘッダーとして、 <code>"expires:Sat Dec 31 00:00:00 2003\n"</code> というような形式で追加されます。 メモ：このオプションにより、ASE ファイルにはコンテンツが追加されますが、Flash Media Server がこの時点でこのコンテンツを使用することはありません。

オプション	説明
-hdr <header>	<p>オプション。出力ファイルに埋め込む、ユーザー定義のヘッダーを指定します。ヘッダーは、Name:Value という形式で指定します。複数のヘッダーを指定する場合は、個別に -h <header> オプションを使用する必要があります。たとえば、casc -h name1:value1 -h name2:value2 と指定します。</p> <p>メモ：このオプションにより、ASE ファイルにはコンテンツが追加されますが、Flash Media Server がこの時点でこのコンテンツを使用することはありません。</p>
-n	<p>オプション。このユーティリティの実行時に、Macromedia の著作権ロゴを表示しません。このオプションをセットする例としては、自動化されたスクリプトからこのユーティリティを実行する場合があります。</p>
-h [<command>]	<p>オプション。このユーティリティの使用に関するヘルプを表示します。-help オプションの後にオプションでコマンドを指定すると、特定のコマンドオプションに関する詳細を表示することができます。</p>

[Macromedia Flash Player 設定] パネルの強制表示

アプリケーションがユーザーのカメラやマイクにアクセスを試みる場合、またはクライアントコンピュータ上にデータを保存しようとする場合に、Flash Player は、その操作に対する許可を得るために [Macromedia Flash Player 設定] ダイアログボックスを表示します。Flash Player のステージ上を右クリック (Windows) または Control + クリック (Macintosh) する方法でも、[Macromedia Flash Player 設定] パネルを開くことができます。[Macromedia Flash Player 設定] パネルを強制的に表示するには、次のコードを使用します。

```
// ユーザーが最後に表示したパネルを開く
System.ShowSettings();
// [ プライバシー ] パネルを開く
System.ShowSettings(0);
// [ ローカル記憶領域 ] パネルを開く
System.ShowSettings(1);
// [ マイク ] パネルを開く
System.ShowSettings(2);
// [ カメラ ] パネルを開く
System.ShowSettings(3);
```

たとえば、アプリケーションでカメラを使用する必要がある場合は、[Macromedia Flash Player 設定] パネルの [プライバシー] で [許可] を選択するようにユーザーに要求し、その後で System.showSettings(0) コマンドを実行することができます。(ステージのサイズが少なくとも 215 x 138 ピクセルより大きいことを確認してください。これがパネルの表示に必要な最小サイズです)。この XML オブジェクトの詳細については、『Flash Media Server 2 クライアントサイド ActionScript リファレンスガイド』の「System.showSettings()」の項目を参照してください。

帯域幅の管理

帯域幅のおよその容量を指定することで、サーバーが各クライアントに送信するデータ量を制御することができます。制御する方法はいくつかあります。1つ目は、設定ファイル (Config.xml) で Flash Media Server の容量を設定する方法です。この方法の詳細については、『Flash Media Server 管理ガイド』を参照してください。2つ目は、`NetStream.receiveVideo()` を使用して、受信ビデオの秒あたりのフレームレートを指定する方法です。3つ目の、データフローを容量に合わせる方法では、クライアントサイドで `Camera.setQuality()` を、サーバーサイドで `Client.setBandwidthLimit()` を使用することで、サーバーにクライアントの容量を知らせます。

帯域幅を管理するためのもう1つの方法は、Flash Player で `NetStream` クラスを使用し、Flash Media Server からの Flash Video (FLV) の再生のためのプロパティを提供することです。ステータスの変化または `NetStream` へのエラーの通知によって、イベントハンドラ (`onStatus()`) が呼び出されます。`NetStream.Play.InsufficientBW` というステータスメッセージは、`NetStream` を介したデータ再生が通常のを下回っていることを表します。

データが通常のを下回る理由はいくつかあります。たとえば、クライアントの帯域幅が十分でない、あるいは、サーバーがビジーであるために期待されているレートでデータを送信できないといった理由が考えられます。`NetStream.Play.InsufficientBW` というステータスメッセージによって、クライアントにフィードバックが提供されるため、クライアントがこのような状況に対処することができます。`NetStream.Play.InsufficientBW` メッセージは、`Info` オブジェクトとして `onStatus()` ハンドラに渡されます。

Code プロパティ	Level プロパティ
<code>NetStream.Play.InsufficientBW</code>	Warning

`Info` オブジェクトの詳細については、『Flash Media Server 2 クライアントサイド ActionScript リファレンスガイド』の付録 A の「クライアントサイドの情報オブジェクト」を参照してください。

次の例のように、警告メッセージを表示するというような簡単な記述になる可能性もあります。

```
ns = new NetStream(nc);
ns.onStatus = function(info){
    if (info.code == "NetStream.Play.InsufficientBW")
        trace("Warning: data is behind");
}
vid.attachVideo(ns);
ns.setBufferTime(2);
ns.play("foo", 0, -1, true);
```

あるいは、次の例のように、より複雑で、NetStream でコンテンツの切り替えを行うような記述にすることも考えられるでしょう。

```
ns = new NetStream(nc);
ns.onStatus = function(info){
    if (info.code == "NetStream.Play.InsufficientBW"){
        // 必要とする帯域幅がより少ないコンテンツの再生に切り替える
        curTime = this.time;
        this.play("foo_low_bandwidth", curTime, -1, true);
    }
}
vid.attachVideo(ns);
ns.setBufferTime(2);
ns.play("foo_high_bandwidth", 0, -1, true);
```

ダブルバイト言語のアプリケーションの記述

ダブルバイトテキスト(アジア言語の文字セットなど)を使用する開発環境でサーバーサイド ActionScript を使用する場合は、サーバーサイド ActionScript を UTF-8 でエンコードされている ASC ファイル内に置く必要があります。したがって、Flash Professional や Dreamweaver のコード用ウィンドウのような、ファイルを UTF-8 標準規格にエンコードする Java エディタが必要です。その上で Date.toLocaleString のようなビルトイン JavaScript メソッドを使用することで、文字列をロケールに変換し、そのシステムに適したエンコード処理を行うことができます。

ヒント	シンプルテキストエディタの中には、UTF-8 標準にファイルをエンコードしないものもあります。Windows XP と Windows 2000 で使用される Microsoft Windows のメモ帳には、[名前を付けて保存]に UTF-8 標準でファイルをエンコードするオプションがあります。
-----	---

Dreamweaver において UTF-8 でエンコードするには、ドキュメントのエンコーディング設定とインライン入力設定の 2 つをチェックする必要があります。

- ドキュメントのエンコーディング設定を変更するには、[修正]-[ページプロパティ]-[エンコーディング]を選択し、[UTF-8 (Unicode)]を選択します。
- インライン入力設定を変更するには、[編集]-[環境設定]または [Dreamweaver]-[環境設定](Mac OS X)を選択してから、[一般]をクリックします。[ダブルバイトのインライン入力可]を選択して、ダブルバイトを入力できるようにします。

ダブルバイト文字をメソッド名として使用するには、ドット演算子ではなく、オブジェクト配列演算子を使用して、メソッド名を割り当てる必要があります。

```
// ダブルバイトのメソッド名を作成する正しい方法
obj["Any_hi_byte_name"] = function(){}
```

```
// ダブルバイトのメソッド名を作成する正しくない方法
obj.Any_hi_byte_name = function() {}
```

アプリケーションのアンロードと再ロード

アプリケーションインスタンスがアンロードされるのは、通常は、ガベージコレクションのためです。すべてのクライアントがアプリケーションからの接続を切断した後に最初にガベージコレクションが実行されたときに、アプリケーションがアンロードされます。言い換えれば、すべてのクライアントの接続が切断されても、アプリケーションが直ちにアンロードされるわけではありません。アプリケーションの起動にはある程度の時間がかかるため、短時間であれば、アプリケーションを残しておいた方が、次に接続するクライアントで同じ遅延が発生しないですみます。

デフォルトでは、ガベージコレクションは 20 分ごと (使用されていない I/O スレッドに対しては 5 分ごと) に実行されます。これらの値は、任意の分 (0 よりも大きい数) に設定することができます。詳細は、『Flash Media Server 管理ガイド』を参照してください。

main.asc ファイルに変更を加えた場合にも、アプリケーションを再ロードして、その変更を反映させる必要があります。アプリケーションを再ロードすると、すべてのユーザーの接続は切断されます。Management Console を使用して、アプリケーションを再ロードすることができます (第 5 章の「[アプリケーションのデバッグと監視](#)」を参照)。

ダイナミックアクセスコントロールの実装

サーバーサイド ActionScript を使用して、共有オブジェクトとストリームに対してダイナミックアクセスコントロールリスト (ACL) の機能を実装することができます。デフォルトでは、すべての接続は、すべてのストリームと共有オブジェクトに対するすべてのアクセス権を持っています。開発者は、共有オブジェクトまたはストリームの作成、読み込み、または更新を行うアクセス権を誰に与えるのかを制御することができます。サーバーサイドのアプリケーションインスタンスに対する接続はすべて、サーバーサイドの Client オブジェクトによって表され、Client オブジェクトには `readAccess` と `writeAccess` という 2 つのプロパティがあります。これら 2 つのプロパティを使用することで、接続ごとにアクセスを制御することができます。

共有オブジェクトとストリームの名前は文字列であるため、いずれも URI エンコードされたデータの同じルールに従っており、名前に基づいてアクセスを定義することができます。`client.readAccess` コマンドと `client.writeAccess` コマンドは、文字列の値を受け付けます。これらの値には、複数の文字列トークンまたは制御の対象とするオブジェクト名の一意な識別子を、セミコロン (;) で区切って指定することができます。ここでは、次の 2 つの文字列を例として示します。

```
client.readAccess = "appStream;/appS0/"
client.writeAccess = "appStreams/public;/appS0/public/"
```

これらの呼び出しと文字列トークンの規則を考慮して、適切に定義したパターンに従って共有オブジェクトとストリームを作成することができます。たとえば、アプリケーションが作成するすべての共有オブジェクトを接頭辞 `appS0` で開始し、すべてのユーザーが使用できる共有オブジェクトを接頭辞 `appS0/public` で開始し、保護したい共有オブジェクトを接頭辞 `appS0/private` で開始するようにします。

読み取りアクセスを次のように設定したとします。

```
client.readAccess = "appS0/"
```

この場合は、サーバーは接続されているすべてのクライアントに対して、appS0 で始まる名前の共有オブジェクトへのサブスクリプトを許可します。

同様に、次のような呼び出しを作成することができます。

```
client.writeAccess= "appS0/public/"
```

この場合は、クライアントは、appS0/public/foo のように appS0/public で始まる名前の共有オブジェクトだけは作成できますが、appS0/private などへのアクセスは拒否されます。

これらの機能を使用してストリームと共有オブジェクトの命名規則を規定することで、ACL を実装することができます。詳細については、『サーバーサイド ActionScript リファレンスガイド』の「Client.readAccess」と「Client.writeAccess」の項目を参照してください。

セキュアアプリケーションの開発

SSL を始めとする方法を使用して、アプリケーションそのものとアプリケーションが使用するデータのセキュリティを確立することができます。

SSL を使用する。Flash Media Server 2 では、SSL を使用できます。SSL を使用するには、認証機関から SSL 証明書を取得するか、自分自身で証明書を作成する必要があります。Flash Media Server では、受信側と送信側の両方の SSL 接続をサポートしています。アプリケーション内で SSL を使用するには、アプリケーションと Flash Media Server の両方で設定が必要です。サーバーの設定に関する詳細については、『Flash Media Server 管理ガイド』を参照してください。

RTMPS (Real-Time Messaging Protocol over SSL)。RTMPS は、SSL 標準に基づいて、セキュアなネットワーク接続を可能にします。RTMPS では、セキュアなポート上の TCP ソケットを介して基本接続性を提供しています。暗号化されたデータが、セキュアな接続を介して渡されるため、第三者による不正データ傍受は防御されます。セキュアな接続はより高い処理能力を必要とし、サーバーのパフォーマンスを低下させる可能性があるため、高レベルのセキュリティを必要とするアプリケーションや重要または機密データを扱う場合に限って、RTMPS を使用するようになります。

RTMPS を使用するには、NetConnection.connect 呼び出しの `rtmp` を `rtmps` に置き換えます。ポート番号を指定しないと、Flash Player はデフォルトのポートであるポート 443 に接続します。セキュアポートは、たとえば、`<HostPort>:1935,80,-443</HostPort>` というように、マイナス符号で指定されます。この例は、Flash Media Player がポート 1935、80、443 (443 は、RTMPS 接続だけを受信するセキュアポートとして指定されます) のいずれかのインターフェイスに対してリッスンするように指定しています。1935 または 80 に対して RTMPS 接続しようとする、接続は失敗します。同様に、ポート 443 への RTMP 接続も失敗します。

ヒント	RTMP データを拒否するファイアウォールの内側に Flash Media Server が接続されていると、その Flash Media Server は別の Flash Media Server への接続に RTMPS を使用することができません。その回避策は、同じファイアウォールの内側にすべての Flash Media Server をクラスタ化することです。
-----	--

クライアント SWF ファイルの場所を確認する。Flash Media Server アプリケーションの配置にあたっては、サーバーサイドスクリプトを使用して、接続する SWF ファイルが正しい場所から入ってきていること (さらに、未知のコンピュータから入ってきていないこと) を検証します。このためには、サーバーが接続を許可する前に、クライアントオブジェクトの `client.referrer` プロパティを確認します。

サーバーサイドのスクリプトによる事前の警告を使用する。サーバーサイドのスクリプトにおいて、悪意のあるアプリケーションから呼び出せるプロシージャを使用しないでください。特にクライアントオブジェクトに付けられたプロシージャは、攻撃に対して脆弱です。プロシージャでは、書き込まれるデータの量をチェックせずにハードディスクに書き込む処理が含まれていないか、プロシージャが無限にループする可能性はないかなどという点について、注意が必要です。サードパーティーのコードを組み込む場合には、アプリケーションを悪意あるコードやバグから保護することも必要です。詳細については、[98 ページの「サードパーティーのコードからのスクリプトの保護」](#)を参照してください。

サードパーティーのコードからのスクリプトの保護

Flash Media Server の強力なスクリプトセキュリティモデルにより、サードパーティーのコードを FMS アプリケーションに安全に組み込むことができます。たとえば、このセキュリティモデルを使用することで、サードパーティーのコードからのファイルへのアクセスとアウトバウンドのネットワーク接続を防ぐことができます。さらに、サードパーティーのプラグインをユーザーにダウンロードさせて、それを FMS アプリケーションにロードするような広範囲な機能を持つアプリケーションでも、コードの保護が必要になるでしょう。悪意やバグが含まれているサードパーティーのコードからアクセスされることで損害を被る可能性があるため、サードパーティーのコードを安全に組み込むことで、アプリケーションコード内のすべてのオブジェクトを保護する必要があります。

注意

Flash Media Server のセキュリティモデルは、無限ループのようなサードパーティーコードのコーディングエラーを検出あるいは回避するには設計されていません。

このセクションでは、保護されているオブジェクト（システムオブジェクトとも呼びます）の作成に関する情報と、このオブジェクトに対する同期および非同期の呼び出し方法の例を紹介します。

システムオブジェクトの概要

システムオブジェクトとは、C レイヤーコードにラップされたビルトインまたはユーザー定義のオブジェクトで、アプリケーションコードからのオブジェクトの保護が可能です。システムオブジェクトに対するすべての呼び出しは C ラッパーを介して行われ、オブジェクトが直接アクセスあるいは検証されることはありません。

サーバーサイドのシステムオブジェクトは、`secure.asc`（または `secure.js` か `secure.ase`）ファイル内に作成し、このファイルをアプリケーションフォルダに置きます。アプリケーションのロード時に、Flash Media Server は、他のどのサーバーサイドのスクリプトファイルを処理するよりも前に、このファイルを探して自動的にロードします。

Flash Media Server 2 のスクリプト実行は、セキュアモードと通常モードの 2 つに分かれています。セキュアモードでは、`secure.asc` ファイル（存在する場合）のみがロードされ、評価されます。その他のアプリケーションスクリプトはロードされません。`secure.asc` ファイルがロードされると、サーバーは、通常のスクリプト実行モードに切り替えます。通常モードは、そのアプリケーションがアンロードされるまで継続されます。

スクリプトのセキュア実行モードでは、グローバル関数の `getGlobal()` と `protectObject()` を使用することができます。これらの関数によって、システムオブジェクトを作成することができます。

スクリプトの通常実行モードでは、サーバーサイドスクリプトの残りの部分と、サードパーティーのコードがあればそれもロードされ、評価されます。グローバル関数の `getGlobal()` と `protectObject()` は通常モードでは使用できないため、サードパーティーのコードからアクセスされることはありません。

`getGlobal()` と `protectObject()` については、『サーバーサイド ActionScript リファレンスガイド』を参照してください。

システムオブジェクトに定義されているすべてのオブジェクトは保護されるため、特権付きであると考えられますが、Flash Media Server が特権レベルを明示的に保存しておくわけではありません。保護されているオブジェクトを使用して、たとえば次のようなセキュリティを実装することができます。

- システム呼び出し
- 特権リング
- ACL (アクセスコントロールリスト)

メモ

システムオブジェクトの数に制限はありません。protectObject() 関数を呼び出すことで保護されるオブジェクトはどれもシステムオブジェクトです。

簡単なシステム呼び出しの例

システム呼び出しの次の例では、ID ジェネレータを実装しています。ID ジェネレータは、新しい ID に対して増分する数値を生成する必要があります。main.asc ファイルにこのジェネレータのコードを記述する場合には、スクリプトのいずれかの部分で単純にこの nextID() 関数を再定義するか、次の例にあるように、_nextID の値を直接変更します。

```
idGen = {};  
idGen._nextID = 0;  
idGen.nextID = function() {return this._nextID++;}
```

このジェネレータのコードを secure.asc ファイルに移すと、このファイルは main.asc ファイルより前にロードされるため、main.asc ファイルのコードからはアクセスされません。secure.asc ファイルの例を以下に示します。

```
// secure.asc の開始  
trace("loading secure.asc");  
  
var global = getGlobal(); // グローバルオブジェクトを取得  
var idgen = {};  
idgen._nextID = 0;  
idgen.nextID = function() {return this._nextID++;}
```

```
// idgen から保護されたオブジェクトを作成し、  
// idGen としてグローバルに使用可能にする
```

```
global.idGen = protectObject(idgen);
```

```
// idGen を非列挙型、読み取り専用、永続にする  
setAttributes(global, "idGen", false, true, true);
```

secure.asc ファイル内で保護した idGen 変数にアクセスしようとする main.asc ファイルの例を以下に示します。

```
// main.asc  
trace("Loading main.asc");  
trace("idGen = " + idGen);
```

```
idGen = 50;
trace("idGen = " + idGen);
```

両方のファイルからの出力を以下に示します。

```
Loading secure.asc
Loading main.asc
idGen = [object Redirector]
idGen = [object Redirector]
```

同期システム呼び出し

同期呼び出しでは、呼び出しから戻るまでに処理が完了しています。同期システム呼び出しとは、保護されていて、システムオブジェクトを介してルーティングされる同期呼び出しです。

次の `secure.asc` の例は、ビルトイングローバル関数の、この場合は `load()` をラップする方法を示しています。`secure.asc` が実行されると、`load()` への呼び出しは次のようにユーザー定義のシステム呼び出しを介して指示されます。

```
var sysobj = {};
sysobj._load = load; // load 関数を隠す
load = null; // 非特権付きコードから使用できないようにする

/* 以下の関数によって、アクセスされるファイルを検査し、必要であればそれを変更または評価できるようにする。fname を評価または変更するためのコードを記述する必要があるが、この関数には現在はそのためのコードは記述されていない。
*/
sysobj.load = function(fname){
    // fname を評価あるいは変更するためのユーザー定義のコード ...
    return this._load(fname);
}
// グローバルオブジェクトを取得
var global = getGlobal();

/* 次に、sysobj を保護し、システムオブジェクトとしてグローバルに利用できるようにする。さらに、このオブジェクトの属性をセットして、読み取り専用で削除できないようにする。
*/
global["system"] = protectObject(sysobj);

setAttributes( global, "system", false, true, true );

// 次に、互換性のためにグローバルの load() 関数を追加する。
// 読み取り専用で削除できないようにする。

global["load"] = function(path){
    return system.load( path );
}
setAttributes( global, "load", false, true, true );
```

非同期システム呼び出し

非同期呼び出しでは、直ちに呼び出しから戻りますが、動作がその段階で完了しているわけではありません。動作が完了すると、応答を受け取るオブジェクトのコールバックを通して、成功あるいは失敗のステータスが返されます。非同期システム呼び出しとは、保護されていて、システムオブジェクトを介してルーティングされる非同期呼び出しです。

ユーザーが定義した応答側のオブジェクトを使用する場合は、必要なときに使用できるようにその応答側を保存しておくことで、そのオブジェクトが公開されることはなくなります。次の例では、ユーザーが提供した受け取り側オブジェクトを保存し、必要なタイミングでそれをディスパッチすることで、非同期の通知を処理する方法を示しています。

Flash Media Server は、特権レベルを明示的には保存しません。たとえば、通常のアプリケーションコードでセットアップした非同期呼び出しは特権付きにはなりません。ただし、システムオブジェクト内での非同期呼び出しのセットアップでは、その呼び出しを特権付きと特権なしのどちらで完了させるかを選択することができます。次の `secure.asc` ファイルを参考にしてください。

```
// 以下のコードはシステム呼び出しの中に記述されている
var resultObj = {};
resultObj.sysobj = this;
/* 次の関数によって、保護されている sysobj を使用して何らかの特権付きの処理を実行することができる。sysobj を使用するコードを記述する必要があるが、この関数には現在はそのためのコードは記述されていない。
*/
resultObj.onResult = function(res) {
    // 保護されている sysobj を使用して
    // 特権付きの何らかの処理を実行する
}
this._nc.call("foo", resultObj, ...);
```

呼び出し側が特権付きではなく、非同期呼び出しのセットアップと完了をシステム呼び出しに頼らなければならない場合には、非同期システム呼び出しを使用することもできます。ただし、コールバックは引き続き特権なしのままで行わなければなりません。たとえば、次の例にあるようにシステムオブジェクトがネット接続をラップし、隠そうとする場合には、この方法が有効です。

```
// 次のコードは secure.asc ファイルに記述されている
sysobj.remoteCall = function(func, responder, arg1, arg2, ...) {
    // 引数を評価あるいは変更するコードを記述

    var sysResponder = {};
    sysResponder.sysobj = this;
    sysResponder.userResponder = responder;
    sysResponder.onResult = function(res) {
        // 応答を変更 / 評価するコードを記述
        // 他の何らかの特権付き関数を実行する場合は、そのコードを記述

        // システムオブジェクトに対するアクセスがあれば削除
        this.sysobj = null;
        delete this.sysobj;
    }
}
```

```
// ユーザーのコールバックに結果を渡す
    this.userResponder.onResult(res);
}
this._nc.call(func, sysResponder, arg1, arg2, ...);
}
system = protectObject(sysobj);
```

上記のコードは、次のようにして、通常のアプリケーションコードから呼び出されます。

```
system.remoteCall("foo", myOnResult, arg1, arg2);
```

コーディング規則

このドキュメントでは、ベストプラクティスのシステムの概要として、特に、ActionScript を使用したコーディングと、Macromedia Flash によるアプリケーションの構築を中心に紹介しています。ここで紹介するガイドラインを守ることで、アプリケーションがより効率的かつわかりやすいものになり、そこで使用されている ActionScript コードのデバッグと再利用が容易になります。

メモ

クライアントサイドのスクリプトの記述には、ActionScript 1 または 2 を使用することができます。サーバーサイドのスクリプトの記述には、ActionScript 1 または JavaScript 1.5 を使用することができます。

命名ガイドラインの順守

アプリケーションの命名規則は一貫性のあるものにし、わかりやすい名前を付けてください。つまり、理解しやすい語やフレーズを使って名前を付けます。そのエンティティの主な機能や目的をその名前から判断できるようにします。ActionScript 1 は、動的型付けの言語であるため、名前にはその型を示す接尾辞を付けます。一般的には、名詞と動詞や形容詞と名詞という組み合わせで名前を付けるのが、最も自然です。たとえば、次のようになります。

- アプリケーション名 (SWF ファイル) : my_movie.swf
- URL に付加されるエンティティ : course_list_output
- コンポーネントまたはオブジェクト : ProductInformation
- 変数またはプロパティ : userName

関数名と変数は小文字で始めます。オブジェクトとオブジェクトコンストラクタは大文字にします。変数の名前には、大文字と小文字の両方を使用します。ただし、そのアプリケーション内で統一がとれている限りは、他の形式でもかまいません。変数名には、英数字とドル記号 (\$) のみを使用できません。変数名の先頭に数字は指定できません。

次のような変数名は使用できません。

```
_count = 5; // アンダースコアで始まる
5count = 0; // 数字で始まる
```

```
foo/bar = true; // スラッシュが含まれている  
foo bar = false; // スペースが含まれている
```

上記に加えて、ActionScript が使用している語 (予約語) を名前として使用することはできません。さらに、Macromedia Flash Player が現在はそのようなコンストラクトをサポートしていない場合であっても、一般的なプログラミングコンストラクトを変数名として使用しないでください。この規則に従うことで、Flash Player の今後のバージョンとアプリケーションとの間でコンフリクトが発生するのを回避することができます。たとえば、MovieClip = "myMovieClip" や case = false というようなコマンドを使用しないでください。

ActionScript は ECMAScript がベースになっているため、現行および今後の ECMA 仕様を参照すれば、予約語のリストを調べることができます。Flash はコンスタント変数の使用を強制しませんが、アプリケーション作成者は、変数の用途を表す命名規則を使用してください。変数名には小文字が大文字と小文字を混在して使用し、さらに1文字目は小文字にします。定数名はすべて大文字を使用します。たとえば、次のようになります。

```
course_list_output = "foo"; // 変数、すべて小文字  
courseListOutput = "foo"; // 変数、大文字と小文字を使用  
BASEURL = http:// www.foo.com; // 定数、すべて大文字  
MAXCOUNTLIMIT = 10; // 定数、すべて大文字  
MyObject = function(){}; // コンストラクタ関数  
f = new MyObject(); // オブジェクト
```

最後に、SWF ファイルの名前には小文字だけを使用し、アンダースコアで区切ります (たとえば、lower_case.swf)。こうすることで、大文字と小文字を区別する、UNIX のようなオペレーティングシステムにアプリケーションを移植することができます。

これらの推奨事項はガイドラインであり、最も重要なのは、何らかの命名規則を規定した上で、その規則に常に従うことです。

コードヒントに対応するための変数命名法

Macromedia Flash ActionScript エディタには、コードヒントをサポートする機能が組み込まれています。この機能を活用するには、変数に対して、厳密な型指定 (ActionScript 2.0 の新機能) を使用するか、特定の形式を使用して変数に名前を付けます。デフォルトの形式では、変数の型を表す接尾辞を変数名に付加します。サポートしている接頭辞ストリングを次の表に示します。

オブジェクトの型	接尾辞ストリング	例
Camera	_cam	my_cam
Video	_video	small_video
Microphone	_mic	the_mic
NetConnection	_nc	my_nc

オブジェクトの型	接尾辞ストリング	例
NetStream	_ns	a_ns
SharedObject	_so	myRemote_so

接頭辞には、コードを読みやすくするというさらなる利点がありますが、接頭辞を使用する代わりに、あるいは、接頭辞の使用に加えて、変数に対する厳密な型指定を使用することができます。このマニュアルで使用している多くの例では、次のような厳密な型指定を使用しています。

```
var my_nc:NetConnection = new NetConnection();
var local_so:SharedObject = SharedObject.getLocal("mySO");
```

厳密な型指定とコードヒントに関する詳細は、Flash の『ActionScript 2.0 の学習』を参照するか、Flash ヘルプで「厳密な型指定」と「コードヒント」を検索してください。

コードのコメント記述

アプリケーションでは、常にコメントを使用してください。コメントを記述しておくことで、そのコードで何を行っているかを知らせることができます。アプリケーションの構築中のあらゆる決定事項をコメントとして記録しておいてください。また、アプリケーションのコーディング方法についての決定がなされるたびに、決定事項とその理由を示すコメントを記述します。

特定の問題の解決法を示すコードを記述する場合には、そのコードを後で見る開発者にとっても問題点が明らかになるようなコメントを追加してください。そうすることで、他の開発者が問題を特定するのに役立ちます。

次の例では、変数に対して簡単なコメントを記述しています。

```
var clicks:Number = 0; // ボタンのクリック回数を示す変数
```

次のようなブロックコメントは、大量のテキストをコメントとして記述する場合に役に立ちます。

```
/*
ボタンがクリックされた回数を記憶するための clicks 変数を
初期化する
*/
```

特定のトピックを示すための一般的な方法を以下に示します。

- // :TODO: トピック
さらに実行すべきことがあることを示します。
- // :BUG: [bugid] トピック
既知の問題をここに記述します。コメントには、問題点の説明と、可能であればバグ ID も記述します。
- // :KLUDGE:
これ以降のコードが必ずしも最適なものではないか、改良の余地があることを示します。このコメントでは、今後、どのようにコードを変更するかを他の作成者に対してアドバイスします。

■ // :TRICKY:

開発者に対して、これ以降のコードには他に影響する記述が多く含まれていることを知らせます。また、これ以降のコードを変更する場合にはあらかじめよく検討するようアドバイスします。

アプリケーションの初期化

アプリケーションの初期化は、開始時の状態に設定するために使用します。この処理は、アプリケーション内の最初の関数呼び出しとする必要があります。この関数は、プログラム内で行う初期化において唯一の呼び出しとする必要があります。他の呼び出しはすべてイベント駆動型です。

```
// フレーム 1
this.init();
function init()
{
    if (this.inited != undefined)
        return;
    this.inited = true;
    // 初期化コードをここに記述
}
```

ローカル変数としての var の使用

すべてのローカル変数には、キーワード `var` を使用します。そうすることで、変数がグローバルにアクセスされることや、さらには、変数が誤って上書きされることを防止できます。さらには、`var` を使用することで、オブジェクトに対する厳密な型指定を使用できるため、コンパイル時にオブジェクトの型がチェックされます。厳密な型指定に関する詳細は、Flash ヘルプで「厳密な型指定」を検索してください。

たとえば、次のコードでは、キーワード `var` を使用せずに変数を宣言しているため、誤って別の変数を上書きしてしまいます。

```
counter = 7;
function loopTest(){
    trace(counter);
    for(counter = 0; counter < 5; counter++){
        trace(counter);
    }
}
trace(counter);
loopTest();
trace(counter);
```

このコードの出力は次のようになります。

```
7
7
0
1
```

2
3
4
5

この場合には、メインタイムラインの `counter` 変数は、この関数内の `counter` 変数によって上書きされます。次のように訂正したコードでは、キーワード `counter` を使用して両方の変数を宣言しています。

```
var counter = 7;
function loopTest(){
  trace(counter);
  for(var counter = 0; counter < 5; counter++){
    trace(counter);
  }
}
trace(counter);
loopTest();
trace(counter);
```


索引

記号

#include コマンド 85

A

ACL (アクセスコントロールリスト)、実装 94

ActionScript

クライアントサイドとサーバーサイド 85

コードヒントのサポート 102

サーバーサイドの情報オブジェクト 77

サーバーに接続するためのコマンド 19

他のスクリプトでのインクルード 85

実行モード 97

[Application Instance] パネル 67

Application クラス

acceptConnection() メソッド 44

onConnect ハンドラ 43

onConnectAccept ハンドラ 46

onConnectReject ハンドラ 46

onDisconnect ハンドラ 45

関する 39、42

application.acceptConnection() メソッド 44、77

application.clearStreams() メソッド 63

application.disconnect() メソッド 77

application.onConnect ハンドラ 43

application.onConnectAccept ハンドラ 46

application.onConnectReject ハンドラ 46

application.onDisconnect ハンドラ 45

application.rejectConnection() メソッド 77

ASC ファイル

NetServices 18

UTF-8 エンコード 93

および Flash Media Server 21

説明 18

場所 16

C

Camera クラス

attachVideo() メソッド 46

bandwidth プロパティ 47

get() メソッド 38

関する 38、46

ヒント 46

Camera.attachVideo() メソッド 46

Camera.get() メソッド、および Video オブジェクト 38

Camera.setQuality() メソッド 92

Client クラス

call() メソッド 44

readAccess プロパティ 94

setBandwidthLimit() メソッド 92

virtualKey プロパティ 58

writeAccess プロパティ 94

およびクライアントサイドの NetConnection オブジェクト 84

関する 39

クライアントからのメッセージの受け取り 39

ヒント 47

client.call() メソッド

次の application.acceptConnection() メソッド 44

client.readAccess プロパティ 94

client.setBandwidthLimit() メソッド 92

client.virtualKey プロパティ 58

client.writeAccess プロパティ 94

[Clients] パネル 70

code プロパティ

情報オブジェクトを参照

components.asc ファイル 18

components.asc ファイルのロード 13

CPU 使用量 75

D

Dreamweaver MX 93

duplicateMovieClip() メソッド 56

F

FAR ファイル

scriptlib パス、格納 89

内容の確認 89

例の使用 88

FAR ユーティリティ

使用 86、89

スクリプトのバイトコードへのコンパイル 89

File クラス 39

Flash Media Server

アーキテクチャ 21

オンラインフォーラム 9

クラスクラスを参照

サードパーティーリソース 9

習得サポート 9

接続 19

通信モデル 23

ヒント 81

リリースノート 9

Flash Media Server オンラインフォーラム 9

Flash Media Server 習得サポート 9

Flash Media Server デベロッパーセンター 9

Flash Media Server に関するサードパーティーのリソース 9

Flash Media Server のドキュメント 8

Flash Media Server リリースノート 9

Flash Player 11

Flash Remoting 40

Flash オーサリングツール 11

FLV ファイル

Flash Media Server との併用 24

削除 55

説明 16、19

場所 16

FSO ファイル 19

G

getGlobal() メソッド 97

H

HTTPS 96

I

ID3 タグ

MP3 ファイルも参照

キャプチャ 64

再生 64

IDX ファイル

削除 55

説明 19

ID の表示 70

J

JavaScript エディタ 12

JavaScript エディタ、使用 27

JS ファイル 18

L

level プロパティ

情報オブジェクトを参照

Linux 12

[Live Log] パネル 69

LoadVars クラス 39

localhost、NetConnection.connect() メソッド 20

M

[Macromedia Flash Player 設定] パネル 91

main.asc ファイル

JavaScript エディタ 13

格納 16

代替名 16

Management Console (管理コンソール)

[Application Instance] パネル 67

[Clients] パネル 70

[Performance] パネル 74

[Shared Objects] パネル 71

[Streams] パネル 73

関する 65

更新間隔 68

サーバーからの切断 75

使用 65

接続 66

ログオン画面 66

ログメッセージ 69

Microphone クラス

setSilenceLevel() メソッド 48

useEchoSuppression() メソッド 48

オーディオフィードバック、回避 48

- 関する 38
- ヒント 48
- Microphone.setSilenceLevel() メソッド 48
- Microphone.useEchoSuppression() メソッド 48
- MP3 ファイル
 - ActionScript による制御 63
 - ID3 タグ 63
 - 説明 62
 - パブリッシュ 63

N

- NetConnection オブジェクト (クライアントサイド)
 - および Client オブジェクト 84
- NetConnection クラス (クライアントサイド)
 - call() メソッド 39
 - 関する 38
 - コンストラクタ 20
 - セキュリティに関する問題 49
- NetConnection クラス (サーバーサイド) 39
- NetConnection.call() メソッド 39
- NetConnection.connect() メソッド 39
 - URI の指定 20
 - およびインスタンス名 17
 - シンタックス 20
- NetConnection.Connect.Closed メッセージ
 - 関する 77
 - 情報オブジェクトも参照
- NetConnection.Connect.Failed メッセージ
 - 関する 78
 - 情報オブジェクトも参照
- NetConnection.Connect.Rejected メッセージ
 - 関する 77
 - 情報オブジェクトも参照
- NetConnection.Connect.Success メッセージ
 - 関する 77
 - 情報オブジェクトも参照
- NetConnection.onStatus ハンドラ 77
- NetServices 18
- NetStream クラス
 - attachVideo() メソッド 46
 - bufferLength プロパティ 50、51
 - bufferStream プロパティ 51
 - onPlayStatus ハンドラ 51
 - play() メソッド 38、40
 - publish() メソッド 38、40
 - send() メソッド 50
 - setBufferTime() メソッド 51
 - 関する 38
 - ストリーム長、取得 50

- ストリームのバッファリング 51
- ヒント 50
- NetStream.attachVideo() メソッド 46
- NetStream.bufferLength プロパティ 50、51
- NetStream.bufferStream プロパティ 51
- NetStream.bufferTime プロパティ 51
- NetStream.play() メソッド 38、40
 - および ID3 タグ 63
 - および Video オブジェクト 38
 - 説明 40
- NetStream.publish() メソッド 18、38、40、81
- NetStream.receiveVideo() メソッド 92
- NetStream.send() メソッド 50
- NetStream.setBufferTime() メソッド 51

O

- On2 V6 コーデック 58
- onResult ハンドラ 33
- onStatus ハンドラ
 - 使用 76
 - スクリプト内の記述場所 77
 - 無効化 77
- onStatus ハンドラの無効化 77

P

- [Performance] パネル 74
- protectObject() メソッド 97

R

- removeMovieClip() メソッド 56
- RTMP (Real-Time Messaging Protocol) 20、22
- RTMPS (Real-Time Messaging Protocol over SSL) 95

S

- scriptlib ディレクトリ 18
- secure.asc ファイル 97
- [Shared Objects] パネル 71
- SharedObject クラス
 - connect() メソッド 52
 - data プロパティ 52
 - flush() メソッド 53
 - getRemote() メソッド 52
 - lock() メソッド 54
 - onSync ハンドラ 52、55

- unlock() メソッド 54
- 共有オブジェクトも参照
- ヒント 52
- SharedObject クラス (クライアントサイド)
 - 関する 38
- SharedObject クラス (サーバーサイド) 40
- SharedObject.connect() メソッド 34、52
- SharedObject.data プロパティ 52
- SharedObject.flush() メソッド 53
- SharedObject.get() メソッド 83
- SharedObject.getLocal() メソッド 82
- SharedObject.getRemote() メソッド 18、34、52、83
- SharedObject.onSync ハンドラ 52、55
- SharedObject.resyncDepth() メソッド 72
- SharedObject.send() メソッド
 - SharedObject クラス
 - send() メソッド 55
- SOL ファイルと SOR ファイル 19
- Sorenson H263 コーデック 58
- SSL (Secure Sockets Layer) 95
- Stream クラス
 - get() メソッド 40
 - 説明 40
 - ヒント 55
- Stream.get() メソッド 40
- Stream.length() メソッド 63
- Stream.play() メソッド 63
- [Streams] パネル 73
- SWF ファイル
 - および Flash Media Server 21
 - 説明 18
 - 場所 16
- System クラス 55
 - onStatus ハンドラ 78
 - 説明 55
- System.onStatus ハンドラ 78
- System.showSettings() メソッド 91

U

- URI (Universal Resource Identifier)
 - localhost の使用 20
 - targetURI パラメータ 20
 - および Flash Player のセキュリティ 49
 - 指定のためのショートカット 83
 - 相対または絶対 83
- UTF-8 形式
 - Dreamweaver MX での有効化 93
 - およびダブルバイトのアプリケーション 93
 - 関する 55

V

- vhost.xml ファイル 58
- Video オブジェクト
 - ステージ上に配置 38
 - 動的な作成 56
- Video クラス
 - attachVideo() メソッド 38
 - および Camera.get() メソッド 38
 - および NetStream.play() メソッド 38
 - 関する 38、56
 - ヒント 56
- Video.attachVideo() メソッド 38

W

- warning プロパティ
 - 情報オブジェクトを参照
- Web サービス 26
- WebService クラス 40

X

- XML クラス 40
- XMLSocket クラス 40
- XMLStreams クラス 40

あ

- アクティブな接続、[Performance] パネル内 75
- アプリケーション
 - CPU とメモリの使用量 75
 - Management Console (コンソール)、表示 65
 - [Performance] パネルからの削除 75
 - アクティブな接続、表示 75
 - 移植性 83
 - イベントの処理 42
 - インスタンスのアンロード 94
 - インスタンスの再ロード 75、94
 - およびインスタンス名 15
 - 開発環境 13
 - 開発環境の作成 13
 - カメラ、使用 47
 - クライアントとサーバーの通信 84
 - クライアント、接続を表示 75
 - 継続時間、表示 75
 - コードのコメント記述 103
 - サーバーへの接続 43
 - 作成のためのチェックリスト 14
 - 実行中のものの一覧 75

- 初期化 104
- 設計 84
- 接続を拒否 46
- 接続を受諾 46
- 切断 45
- ダブルバイト 12、93
- ディレクトリ 15
- 登録アプリケーションディレクトリ 14
- トラブルシューティング 65
- 配置 15、27
- 秒あたりのバイト数 75
- ワークフロー 35
- アプリケーションインスタンス、使用 17
- アプリケーションのアンロードと再ロード 94
- アプリケーションの初期化 104
- アプリケーションの設計
 - サーバー間での移植性 83
 - 相互依存性 84
- アプリケーションの配置 15

い

- 移植性、アプリケーション設計における 83
- イベントハンドラ、onStatus 76
- インスタンス名 15

え

- 永続性と共有オブジェクト 82
- エッジサーバー 57

お

- オーディオ、フィードバック 48
- 大文字と小文字を区別する名前 83
- オブジェクト
 - クライアントとサーバーの通信 31、40、84
 - セキュアな 96
 - プロパティ 79
- オブジェクトプロパティ、値の取得 79
- オリジンサーバー 57

か

- 開発環境、セットアップ 11
- 外部データソース 26
- カスタムストリーム配信 59
- 仮想キー
 - 関する 58、60

- マッピング 60
- 仮想ディレクトリ
 - 関する 58、60
 - マッピング 60
- カメラ
 - Camera クラスも参照
 - オフにする 46
 - デフォルト、選択 12、91
 - 複数のアプリケーションでの使用 47

き

- 共有オブジェクト 19
 - Management Console (管理コンソール)、での表示 71
 - ShareObject クラスも参照
 - SOL ファイル形式と SOR ファイル形式 19
 - SOR ファイル形式 19
 - アンロック 54
 - 永続性、ローカルとリモート 82
 - 関する 25、41
 - サーバーサイド 40
 - サーバー上でのフラッシュ 53
 - 種類 82
 - 衝突、回避 17
 - スロット 52
 - 接続、表示 71
 - タイプ、表示 71
 - デバッグ 52
 - 同期化 52
 - 同期化の問題、回避 53
 - 名前、表示 71
 - ハイスコアのトラッキング 54
 - ヒント 52
 - ファイル形式 19
 - ファイルの場所 82
 - プロキシ 42
 - プロパティ、表示 71
 - 読み取り / 書き込みアクセス、コントロール 94
 - リモート 42
 - ローカル 38、41
 - ロック 54
 - 記録されたストリーム
 - 衝突の回避 17
 - ファイル 81
 - ファイルの削除 55
 - 記録されたストリームファイルの削除 55

く

- クライアント / サーバー
 - オブジェクト通信 31
 - 通信 84
 - ワークフロー 29
- クライアントサイド
 - ActionScript 13
 - サーバーメソッドの呼び出し 31
 - ファイル、格納 15
 - メソッド、サーバーからの呼び出し 41、84
- クライアント、接続を表示 75
- クラス
 - Application クラス 39、42
 - Camera クラス 38、46
 - Client クラス 39
 - File クラス 39
 - LoadVars クラス 39
 - Microphone クラス 38
 - NetConnection クラス (クライアントサイド) 38
 - NetConnection クラス (サーバーサイド) 39
 - NetStream クラス 38、50
 - SharedObject (クライアントサイド) 38
 - SharedObject クラス (サーバーサイド) 40
 - Stream クラス 40
 - System クラス 55
 - Video クラス 38、56
 - WebService クラス 40
 - XML クラス 40
 - XMLSocket クラス 40
 - XMLStreams クラス 40
 - クライアントサイド 37
 - クライアントとサーバーの通信 40
 - サーバーサイド 39

こ

- 更新間隔
 - Management Console (管理コンソール) を参照
- コーディング規則 101
- コーデック
 - ビデオコーデックを参照
- コードのコメント記述 103
- コードヒント 102
- コンポーネント 13

さ

- サーバー
 - アプリケーションの移動 83

- 接続 19
- 接続を開く 20
- サーバーサイド ActionScript
 - クライアントからのメソッドの呼び出し 41、84
 - クラスも参照
 - スクリプト 96
 - スクリプト、格納 15
 - セキュアなオブジェクトの作成 96
 - 説明 13
 - 特権レベル 97
- サーバーサイドスクリプト
 - セキュア実行モード 97
 - セキュリティモデル 97
 - 通常実行モード 97
 - バイトコードへのコンパイル 89
- セキュリティモデル、スクリプト 97
- サーバーサイドの共有オブジェクト
 - 共有オブジェクトを参照
- サーバーへの接続 19、31
- サーバーへの接続を開く 20

し

- システムオブジェクト 97
- システム呼び出し
 - 同期 99
 - 非同期 100
- 情報オブジェクト
 - code プロパティ 76
 - level プロパティ 76
 - warning プロパティ 76
 - 関する 76
 - サーバーサイド 77

す

- ストリーム
 - Management Console (管理コンソール)、表示 73
 - 値、表示 73
 - カスタム配信 58
 - 関する 23
 - 記録されたファイル、削除 55
 - 再生ステータス 51
 - 衝突の回避 17
 - タイプ、表示 73
 - 長さ、ActionScript による取得 50
 - 名前、表示 73
 - バッファリング再生 51
 - 複数のデータタイプの使用 50

プロパティ、表示 73
読み取り / 書き込みアクセス、コントロール 94
ライブ 57
ストリーム間の衝突 17
ストリームのタイプ、[Streams] パネル 73
スロット
共有オブジェクトを参照

せ

セキュリティ
HTTPS 96
サーバーサイドサイドスクリプト 96
接続の制限 96
セキュリティモデル、スクリプト 97
接続
拒否 46
受諾 43、46
切断 45
接続時間 70

た

帯域幅
Management Console (管理コンソール)、での表示
75
およびマイク 48
管理 92
速度設定 47
秒あたりのバイト数 75
ダイナミックアクセスコントロール 94
ダブルバイトのアプリケーション 12、93

て

ディレクトリ、命名 83
テキスト、エンコーディング 55
テクニカルサポート 9

と

登録アプリケーションディレクトリ 14
登録アプリケーション名 14
トラブルシューティング
NetConnection.Connect.Failed メッセージ 78
アプリケーションがサーバーに接続されない 78

は

バイトコード、スクリプトをコンパイル 89
バイト数、送受 70

ひ

ビデオ
On V6 コーデック 58
Sorenson H263 コーデック 58
Video クラスも参照 58
コーデック 58
ストリーミング 58
配信 58
ブロードキャスト 67
ライブブロードキャスト 57
ビデオコーデック
On2 V6 58
Sorenson H263 コーデック 58
関する 58

ふ

ファイル
クライアントサイド ActionScript で外部からインクルード 86
小文字の名前、使用 83
サーバーサイド ActionScript で外部からインクルード 86
複数、使用 85
ファイル、命名 83
ファイル形式、Flash Media Server 18
複数ファイル、使用 85
プライベート
合意を得る 14
フレームレート、FLV と SWF 56
プロパティ、オブジェクトの取得 79
プロパティ、[Shared Objects] パネルに表示 72

へ

変数、ローカル変数の使用 104

ま

マイク
Microphone クラスも参照
オン状態の維持 48
説明 12

デフォルトの選択 91

デフォルトのデバイス、指定 12

め

メッセージ

送受 70

欠落した 70

秒あたりの数、表示 75

メモリ使用量 75

よ

読み取り / 書き込みアクセス、共有オブジェクト 94

ら

ライブストリーム

ストリームを参照

り

リモート共有オブジェクト

共有オブジェクトを参照

リレーショナルデータベース 26

ろ

ローカルの共有オブジェクト

共有オブジェクトを参照

ローカル変数、変数の使用 104

ログメッセージ

Management Console (管理コンソール) を参照

わ

ワークフロー 13