

Developing Communication Applications

Macromedia Flash Communication Server MX 1.5

Trademarks

本ドキュメントは、ヒム・カンパニー 永井勝則が Macromedia 社とは何の関係もなく、勝手に翻訳したものです。

(2003/05/21)

CONTENTS

INTRODUCTION

About This Manual	7
Intended audience	7
About this manual.	8
About the Flash Communication Server documentation	8
Flash Communication Server support	9
Third-party resources	9
Flash Communication Server editions	9
Typographical conventions	10
Searching Flash Communication Server Help	10

CHAPTER 1

Getting Started	11
Installation and system requirements	11
Setting up your hardware and software	11
Creating your working environment	12
Preview of development tasks	13
Deploying applications and application instances	14
File types used by Flash Communication Server	16
Connecting to the server	17
Publishing and playing a stream	18

CHAPTER 2

Flash Communication Server architecture overview.	23
About streams and shared objects	25
Workflow for creating and deploying applications	29
Application flow	31

CHAPTER 3

Using Communication Objects	35
Flash Communication Server objects.	36
Application object	40
Camera object	42
Client object	43
Microphone object	43
NetConnection object (client-side)	44
NetStream object.	44

SharedObject object	46
Stream object	48
System object	49
Video object	49

CHAPTER 4

Debugging and Monitoring Applications	51
Using the Communication App inspector	51
Using the NetConnection Debugger	58
Using onStatus event handlers	61

CHAPTER 5

Application Development Tips and Tricks	65
Application design and development	65
Adding a privacy module	73
Coding conventions	75
File types and paths	79
Snapshots and thumbnails	82

CHAPTER 6

Using Communication Components	87
Using components in a simple application	87
Using the SimpleConnect component	90
Advanced development with components	93
Components List	94
SimpleConnect component	94
ConnectionLight component	95
FCConnectionLight object	97
VideoPlayback component	99
FCVideoPlayback object	99
VideoRecord component	100
FCVideoRecord object	103
Chat component	105
FCChat object	106
FCChat server-side object	108
PeopleList component	110
FCPeopleList object	111
UserColor component	113
FCUserColor object	114

Using your component with the UserColor component	115
Cursor component	115
FCCursor object	117
Whiteboard component	119
FCWhiteboard object	120
PresentationText component	121
FCPresentationText object	124
PresentationSWF component	127
FCPresentationSWF object	129
AVPresence component	133
AVPresence object	135
AudioConference component	136
FCAudioConference object	138
VideoConference component	140
FCVideoConference object	141
SetBandwidth component	143
FCSetBandwidth object	145
Using your component with the SetBandwidth component	148
RoomList component	149
FCRoomList object	154
CHAPTER 7	
Application Server Connectivity	159
Connecting through Flash Remoting	159
Sample 1: Simple Remoting	160
Sample 2: Sending Mail	164
Sample 3: Recordset	167
APPENDIX	
Working With MP3 Files	171
Playing MP3 files	171
About supported versions of ID3 tags	172
Using Server-Side Communication ActionScript	172
GLOSSARY	
Flash Communication Server Terms	175
INDEX	177

INTRODUCTION

このマニュアルについて

Macromedia Flash Communication Server MX 1.5 によろこ！

Flash Communication Server は、Macromedia リアル-タイム・メッセージング・プロトコル(RTMP)を使い、ストリーミング・オーディオやビデオ、データなどを Macromedia Flash MX アプリケーションに組み入れる技術です。Flash MX アプリケーションにコミュニケーションを組み入れることにより、チャットやウェブ放送、ホワイト・ボード、マルチユーザー・ゲームといった、広範囲に及ぶ多数のユーザーが参加する、共同アプリケーション開発のサポートが可能になります。Flash Communication Server はまた、サーバー間でアプリケーション処理を分配することによる拡張性や、アプリケーション・サーバーなどの外部ソースと通信することによる伸張性をもたらします。

以下のセクションは、Flash Communication Server を学ぶにあたっての初めに読むべき項目です。

- チャプター 1 の”Getting Started”(1 1 ページ)では接続に必要な基本的な情報を得ることができます。
- Macromedia Flash Communication Server のキー・コンセプトをまず読みたい場合は、チャプター 2 の”Flash Communication Server 構造概観”(2 3 ページ)へ進んでください。
- Flash Communication Server インターフェイスのウェルカム・ページには、チュートリアルやサンプル・アプリケーションがあります。
- インストールされた製品に付属するその他のドキュメントを読みたい場合は、About Flash Communication Server ドキュメントをご覧ください。

対象読者

このマニュアルでは、Flash 開発者を対象に、開発環境のセットアップを含むステップや、アプリケーションのデバッグ、テストも含む Flash Communication Server アプリケーションの作成を説明します。

Flash MX、ActionScript、Flash Player には習熟している必要があります。JavaScript やクライアント・サーバー・モデル、ネットワークのコンセプトなどの知識があると大変役に立ちます。特に JavaScript はサーバー・サイドの ActionScript の基本を形成しているので、非常に助けとなります。

Flash Communication Server のドキュメント・セットでは、Flash MX をすでにインストールし、使用方法は知っているものと仮定して書かれています。

このマニュアルについて

このマニュアルには、Flash Communication Server とドキュメント・セットを使って Flash Communication Server を始める際に助けとなる情報が書かれています。他のマニュアルやヘルプ・システムの場所や、ソフトウェアのインストールに必要なシステムや知識、Flash Communication Server の仕様紹介、ワークフローについての記述などが書かれています。またコミュニケーション・オブジェクトの詳細やコミュニケーション・アプリケーションの作成方法、デバッグや配置のしかたも書かれています。

Flash Communication Server ドキュメントについて

Flash Communication Server ドキュメントは、"Using Flash MX" という Flash MX ドキュメント、Flash MX オンライン ActionScript 辞書と合わせて使用するよう書かれています。

全ての Flash Communication Server ドキュメントは、Adobe Acrobat Reader による読めて印刷できる PDF ファイル形式や、HTML ヘルプで読むことができます。

Flash Communication Server Help を最もうまく活用するために、Macromedia は、Microsoft Internet Explorer6.0 かそれ以降の Java プレイヤーをサポートしたブラウザを使用することを強く推奨します。Flash Help はまた、Microsoft Windows と Macintosh 上での Netscape6.1 かそれ以降をサポートしています。Flash MX と Flash Communication Server Help を同時に作動させると、Macintosh 上では、ブラウザの必要とするメモリにより、32MB 以上必要になる場合があります。

Flash Communication Server には以下のドキュメントが含まれます。

- *Installing Flash Communication Server*、サーバーや Flash オーサリング・エクステンションのインストール方法を説明しています。Flash Communication Server CD にはこのマニュアルの PDF 版(FlashCom_Installing.pdf)があります。
- このマニュアル、*Developing Communication Applications*、開発環境のセットアップ方法やコミュニケーション・アプリケーションを作成する際の Flash MX オーサリング環境や、Flash Communication Server アプリケーションのプログラミング・インターフェイス(API)の使用法について説明しています。このマニュアルはまた、Flash communication components やその使用方法も書かれています。

このマニュアルの PDF 版(FlashCom_Developing.pdf)は、Flash Communication Server CD にあります。Flash MX 内で HTML ヘルプとしてこのマニュアルを見るには、Help>Welcome to FlashCom を選択し、Developer をクリック、Developing Communication Applications をクリックします。

- *Managing Flash Communication Server*、サーバーの構成や維持、Administration Console の使用の詳細が書かれています。
- *Client-Side Communication ActionScript Dictionary*、クライアント・サイドの機能を作成する際に使用できる ActionScript のドキュメントです。

このマニュアルの PDF 版(FlashCom_CS_ASD.pdf)は、Flash Communication Server CD にあります。Flash MX 内で HTML ヘルプとしてこのマニュアルを見るには、Help>Welcome to FlashCom を選択し、Developer をクリック、右矢印をクリックし、Client-Side Communication ActionScriptDictionary をクリックします。また、Flash MX Actions パネルの Reference ボタンをクリックしてもこの情報を読むことができます。

- *Server-Side Communication ActionScript Dictionary*、サーバー・サイドの機能を作成する際に使用できる ActionScript のドキュメントです。

このマニュアルの PDF 版(FlashCom_SS_ASD.pdf)は、Flash Communication Server CD にあります。Flash MX 内で HTML ヘルプとしてこのマニュアルを見るには、Help>Welcome to FlashCom を選択し、Developer をクリック、右矢印をクリックし、Server-Side Communication ActionScriptDictionary をクリックします。また、Flash MX Actions パネルの Reference ボタンをクリックしてもこの情報を読むことができます。

- *Server Management ActionScript Dictionary*,Administration Console を拡張するために使用したり、自前の管理ツール、監視ツールを作成するために使う上級の ActionScript メソッドのドキュメントです。

このマニュアルの PDF 版(FlashCom_Management_ASD.pdf)は、Flash Communication Server CD にあります。

Flash Communication Server サポート

Flash Communication Server のサポートは、他のリソースでも探すことができます。

- Flash Communication Server Support Center、www.macromedia.com/go/flashcom_support では、Flash Communication Server に関する技術資料や最新情報が載せられています。
- Flash Communication Server DevNet サイト、www.macromedia.com/go/flashcom_devnetでは、Flash Communication Server アプリケーション作成に関するチップスやサンプルが掲載されています。
- Flash Communication Server Online Forum、www.macromedia.com/go/flashcom_forumでは、他の Flash Communication Server ユーザーとチャットをする場が与えられています。
- Flash Communication Server リリース・ノート、最新の情報や懸案事項は、www.macromedia.com/go/flashcom_mx_releasenotesで読むことができます。

サードパーティ・リソース

Macromedia は、Flash Communication Server に関するサードパーティのリソースへのリンクをはり、以下の web サイトを推奨しています。

- Macromedia Flash コミュニティ・サイト
- Macromedia Flash 書籍
- オブジェクト指向プログラミングコンセプト

これらのサイトへは、www.macromedia.com/go/flashcom_resourcesからアクセスできます。

Flash Communication Server エディション

Flash Communication Server には多くのエディションがあります。それぞれのエディションに関する記述は、Flash Communication Server web サイト、www.macromedia.com/go/flashcom_mx で見ることができます。Flash Communication Server ドキュメント内の情報は、全ての Flash Communication Server エディションに適用できます。

表記

このマニュアルでは以下の表記が使用されています。

- Code font はサンプル内の ActionScript 文、HTML タグと属性名、リテラル・テキストを示します。
- イタリック体はコードやパスのプレースホルダ(内容を記述すべき場所)要素を示します。例えば、`attachAudio(source)`は、`source`部分に任意の値を与える必要があることを示します。`/settings/myPrinter`は、`myPrinter`部分に特定の場所を与える必要があることを意味します。
- ディレクトリ・パスはフォワード・スラッシュ(/)で書かれています。Windows で Flash Communication Server を動作させる場合は、フォワード・スラッシュを¥におきかえます。Windows ではパスは¥が使われます。

Flash Communication Server Help を探す

Flash Communication Server 1.5 Help には検索ツールが付属しています。Any、All、Exact で3種類の検索を行うことができます。

- Any は入力した言葉の少なくとも1つでの結果を、順不同で示します。例えば”mp3”か”flv”でヘルプ・トピックを求めると、以下を入力します。

mp3 flv

- All は入力した全ての言葉での結果を順不同で示します。例えば”clear”と”flv”でヘルプ・トピックを求めると、以下を入力します。

+clear flv

- Exact は入力した正確な言葉での結果を示します。例えば”clear mp3”でヘルプ・トピックを求めると、以下を入力します。

Clear+mp3

CHAPTER 1

始めよう

この章には、Macromedia Flash Communication Server MX 1.5 でアプリケーション開発を開始する前に読んでおく必要のある最初のステップが書かれています。開発環境のセットアップのほか、アプリケーションの配置方法、サーバーへ接続する簡単なアプリケーションの作成方法が書かれています。

システム条件

Flash Communication Server のインストール方法やシステム条件は *Installing Flash Communication Server* を参照してください。

ハードウェアとソフトウェアのセットアップ

Flash Communication Server アプリケーションを書くには、Macromedia Flash MX オーサリング・ソフトウェアと Flash Communication Server、最新の Flash Player をインストールしなければなりません。オーディオやビデオを記録するアプリケーションを書きたい場合は、マイクかカメラを接続する必要もあります。それに加え、アプリケーションが Flash Communication Server アプリケーションのサーバー・サイド・スクリプトを必要とするなら、Macromedia Dreamweaver MX のような UTF-8 JavaScript エディターが要ります。このセクションではこれら開発環境の各要素について説明されます。

Flash MX オーサリング・ソフトウェア まだ Flash MX をインストールしていない場合は、Flash MX ドキュメントを参照してください。

Flash Communication Server ソフトウェア まだサーバーをインストールしていない場合は、製品 CD の PDF ファイルやダウンロードもできる *Installing Flash Communication Server* を参照してください。

Flash Player Flash Player の最新版を使用していることを確認してください。最新版をダウンロードするには、Macromedia Flash Player ダウンロード・センター(www.macromedia.com/go/getflashplayer)へ行きます。

Note: UNIX で Flash Communication Server を作動させている場合は、コミュニケーション・アプリケーションを作成するための Flash MX がインストールされた Windows か Macintosh コンピュータが必要で、そのオーサリング・コンピュータにオーサリング・エクステンションをインストールする必要が生じます。管理ツールを使用するためには UNIX サーバー・コンピュータに Flash Player をインストールします。詳細は *Installing Flash Communication Server* を参照してください。

カメラとマイク カメラとマイクを接続するには、デバイスに付属する指示に従ってください。Flash Communication Server と互換性のあるカメラのリストは、Macromedia web サイトのカメラ互換性に関するドキュメント (www.macromedia.com/go/camera_compatibility)をご覧ください。リストに載っていないカメラも Flash Communication Server で使用できる可能性もあり、Macromedia でテストを行っています。

カメラの多くにはマイクも付属しています。マイクだけを接続することもできますが、最もよい結果が得られるのは、マイク/ヘッドセットの組み合わせです。

デバイスをインストールすると、そのまま Flash が使用するカメラやマイクを特定できます。Flash MX ムービーのプレビュー中に、右クリック(Windows)またはコントロール・キーを押してクリック(Macintosh)すると、コンテキスト・メニューから設定を選択し、マイクかカメラ・パネルをクリックすると、ポップアップ・メニューから好きなデバイスを選択できます。

JavaScript エディターを使用する .asc や .js の拡張子のついたファイルに書くことになる、Server-Side Communication ActionScript のコードは、どんなテキスト・エディタを使っても書くことができます。文法がハイライト表示されたりコード・ヒントが出たりする Macromedia Dreamweaver などの web ベースに特化されたソフトウェアもよいかも知れません。

アジアの言語で使用される 2 バイト文字のような非 ASCII テキストをサーバー・サイド・スクリプトに含みたい場合は、UTF-8 のエンコーディングが可能なエディターを使う必要があります。Flash Communication Server は、クライアントから別のクライアントに 2 バイト文字を渡すために、UTF-8 にエンコードされた ASC ファイルを要求します。2 バイト言語に関する Dreamweaver MX の設定の情報は、2 バイトアプリケーションを書く、を参照してください。

作業環境を作る

このセクションは、Flash Communication Server アプリケーションの作成前に知っておくべきことについて説明します。

サーバーが動作していることを確認する アプリケーションをパブリッシュしテストするには Flash Communication Server が動作していません。詳細はサービスの開始を参照してください。

サーバーの URI を特定する この章のコードは、Flash Communication Server が動作しているのと同じコンピュータ上の Flash MX オーサリング環境を使用していると仮定して書かれています。そうでない場合は、書かれている内容の全ての connect コマンドに、サーバー名を加えます。例えば、サーバーが *myServer.myDomain.com* で動作しているなら、以下のように変更します。

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

を

```
new_nc.connect("rtmp://myServer.myDomain.com/doc_record/room_02");
```

に変更します。

Note: 書き直されたコード内の、rtmp の後の 2 本のスラッシュ(/)を確認してください。1 本のスラッシュは、Flash Communication Server が動作している同じコンピュータ上に SWF アプリケーションがある場合のみサポートされます。

パブリッシュ形式を特定する Flash MX が SWF と HTML をパブリッシュするように設定します。パブリッシュ時に形式を特定するには、Flash MX の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・サイドの ActionScript コードを書く クライアント・サイドの ActionScript に限っていうなら、Fla ファイルの個別のオブジェクトでなく、最初のキーフレームのレイヤーにそのコードはアタッチされるべきです。

サーバー・サイドの ActionScript コードを書く サーバー・サイドの ActionScript コードを使用するアプリケーションには、サーバー・サイドのスクリプト・ファイル内にコードを書きます。ファイル名は main.asc で(もしくは、*registered_app_name.asc* も可能)、Macromedia Dreamweaver MX のような JavaScript で書くことができます。サーバー・サイドのコードは大文字と小文字を区別することを忘れないようにしてください。

components.asc をロードする アプリケーションでコミュニケーション・コンポーネントを使うには、scriptlib ディレクトリに置かれた components.asc ファイルをロードしなくてはなりません。このファイルをロードするには、まだ

作成していないなら、アプリケーション用のサーバー・サイド・スクリプトを作成し、main.asc といった適切な名前をつけ、1行めに次のコードを加えます。

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

Tip: コミュニケーション・コンポーネントを使用しているアプリケーション・ディレクトリにこのファイルをコピーすることもできます。しかし適切な名前をつけることを忘れないようにしてください。

クライアント・サイド・コードの初期化 各サンプルを作成するときは、その1行めに次のコードを加えます。

```
#include "Netdebug.as";
```

この行により NetConnection デバッガが使用できるようになり、アプリケーションが使用しているストリームや共有オブジェクトについて、詳細をトレースしたり究明できます。しかしアプリケーションを配置する前にこのコード部分を削除できます。デバッガの詳細は Using the NetConnection Debugger を参照してください。

ユーザーのプライバシー権について 誰かの画像や声を記録したり配信する前に、その人に作り手の意思を伝え同意、了解を得ることは重要です。オーディオやビデオの記録、配信方法を説明するサンプルでは、記録され配信されることをユーザーに伝え、アプリケーションから出る機会を与えるテキスト・ボックスを加えています。記録や配信についてさらに強くその許可を求める場合は、Adding a privacy module を参照してください。

作動するアプリケーションの監視 サーバー管理者の権利を持っているなら、生成されているログ・メッセージや共有オブジェクトの値など、テストしているアプリケーションの詳細を見ることができます。そうするには Flash MX でウィンドウ>Communication App Inspector を開き、Flash Communication Server に接続して、監視したいアプリケーション・インスタンスを選択し、View Detail を選びます。詳細は Using Communication App inspector を参照してください。

開発作業のプレビュー

次のチェックリストは、Flash Communication Server を用いて作業するアプリケーションに関する、完了させなければならない作業の概観です。この章ではさらに詳しく各作業について述べられており、そのためにはこの章全てを読む必要があります。

Flash Communication Server アプリケーションを作成、配置するには、次の作業を完了させます

1. my_app といった新しいアプリケーションの名前を選び、サーバーにアプリケーションを登録します。つまり、Flash Communication Server アプリケーション・ディレクトリに、アプリケーションの名前の新しいディレクトリを作成します。この名前は *registered application name* です。ディレクトリは *registered application directory* です。
2. Flash MX で、登録するアプリケーション名や URI 内のアプリケーション・インスタンスを含めた新しい NetConnection 文を書いた FLA ファイルを作成します。例えば以下ようになります。

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

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

これによりクライアントは *registered_app_name* アプリケーションに接続します。

3. 登録したアプリケーション名で FLA ファイルを保存します。FLA ファイルはどこでも保存できます。それは SWF を作成するために使用されるソース・ファイルであり、配置したアプリケーションの一部ではありません。
4. サーバー・サイドの ActionScript を含むスクリプト・ファイルがある場合は、Flash Communication Server アプリケーション・ディレクトリ内の登録したアプリケーション・ディレクトリに置くか、登録したアプリケーション

ン・ディレクトリ内に/scripts ディレクトリを作成しそこに置きます。サーバー・サイド・スクリプト・ファイルの名前は、main.asc か *registered_app_nameasc* の名前をつけることができます。

5. クライアントがアクセスでき、Flash Communication Server が使用していないディレクトリにアプリケーションの SWF をパブリッシュします。例えば、クライアントにアプリケーションを提供する web 公開ディレクトリに SWF を置きます。クライアントに SWF を e-mail で送って他のディレクトリに SWF を置いておくということもできます。

アプリケーションによっては、さらにステップが必要になる場合もあります。しかし、どんな機能をもったアプリケーションでも、これらのステップは必要です。

Tip: Macromedia は、Flash Communication Server やそのアプリケーションに使用するディレクトリ名やファイル名は、全て小文字にし、空白を使わないことを推奨します。こう習慣づけることで、開発中に別のプラットフォームのコンピュータにファイルを移したらアプリケーションが動作しなくなったということがなくなります。

この章の後半は、アプリケーションのセットアップに関連する事項や、Flash Communication Server アプリケーションで使用するオブジェクトやファイル紹介、Flash Communication Server アプリケーションへの接続や作成方法などについて説明します。

アプリケーションとアプリケーション・インスタンスの配置

このセクションは、アプリケーション・データの設置場所とアプリケーション・インスタンスを動作させる方法とその理由について説明します。

サーバー・サイトとクライアント・サイド・ファイルを置く

サーバー・サイド・アプリケーション・ファイルの既定の場所は Windows では C:\Program files\Macromedia\Flash Communication Server MX\applications で、UNIX では /opt/macromedia/flashcom/applications です。これはアプリケーション・ディレクトリとして参照されます。コミュニケーション・アプリケーションを作成したなら、アプリケーション・ディレクトリ内にアプリケーションのサブディレクトリである、アプリケーション・ディレクトリを作成しなければなりません。そしてそのサブディレクトリの中に、ASC ファイルや記録されたストリーム(FLV)、リモート共有オブジェクト・ファイルといったあらゆるアプリケーション・データを配置します。

SWF や HTML などのクライアント・サイド・ファイルはどこでも置くことが可能で、最も好ましいのが、web サーバー公開ディレクトリに配置することです。アプリケーションの FLA ファイルは配置するまで SWF や HTML ファイルと一緒に置いておくことができます。公開時にはそれを移動し、安全な場所に置きます。

開発作業中、組織的な目的のためには、FLA や SWF、HTML や ASC などの全てのクライアント、サーバー・アプリケーション・ファイルをひとつのサブディレクトリに置いておくのが便利です。アプリケーションを配置するときは SWF や HTML ファイルはどの場所にでも置けますが、登録されたアプリケーション・ディレクトリは、アプリケーションが使用する ASC や FSO、FLV ファイルとともにサーバーになくてはいけません。ASC や FLV、FSO ファイルなどのサーバー・サイド・ファイルと FLA ソース・ファイルは、サーバーやアプリケーションを配置するときに web ルート・ディレクトリにあってはいけません。SWF と HTML ファイルだけが web 公開ディレクトリにあるべきなのです。

どんな場合でも、NetConnection.connect コマンドを出して接続する、アプリケーションと同じ名前のアプリケーション・ディレクトリを作成する必要があります。

例えば、chat_App という名前のアプリケーションがあったとすると、

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

アプリケーション・ディレクトリ内に chat_App という名前のサブディレクトリを作成しなければなりません。アプリ

ケーションがサ 0 パー・サイド・スクリプトを使用するなら、chat_App.asc という名前のファイルを作成し、この同じディレクトリに chat_App.asc を置きます。

Note: たとえサーバー・サイド・スクリプト・ファイルを持たない場合でも、アプリケーション名と同じアプリケーション・ディレクトリを作成しなくてはなりません。これは、Flash Communication Server がアプリケーション・ディレクトリ内に、アプリケーションによって生成されたストリームや共有オブジェクトを保存するからです。またこのディレクトリの存在により、Flash Communication Server は、アプリケーションは正当なものであり、ユーザーがアプリケーションのインスタンスに接続できることを知ります。

アプリケーションのサーバー・サイド・スクリプト・ファイルを置く

アプリケーションのほとんどでサーバー・サイド・スクリプトを使用したいことでしょう。アプリケーションのサーバー・サイド・スクリプトは、JavaScript テキスト・エディターで作成します。このファイルに `main.asc` と名前をつけるか、`registered_app_name` がアプリケーション・ディレクトリの名前であり、アプリケーションの名前である `registered_app_name.asc` の名前をつけます。このファイルをアプリケーション・ディレクトリに保存します。例えば、チャット・アプリケーションの `applications/chat_app` ディレクトリを作成したなら、サーバー・サイド・スクリプト・ファイルの名前は `chat_app.asc` であり、`chat_app` ディレクトリにそれを置きます。

Note: サーバー・サイド・スクリプト・ファイルの拡張子に `.js` を使用することもできます。希望するなら、アプリケーション・ディレクトリ内の `scripts` と名づけたサブディレクトリ (`applications/chat_App/scripts`) にサーバー・サイド・スクリプトを置くことができます。サーバー・サイド・スクリプトの名前づけに関する詳細は、Flash Communication Server Support Center を参照してください。

アプリケーション・インスタンスを使用する

アプリケーションはアプリケーション・インスタンスを作成することにより動作します。クライアントがアプリケーションに接続するとき、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しています。例えば、クライアントが `chat_app` というアプリケーションに接続したとします。

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

この例では、クライアントは、インスタンスが特定されていないため、実際にはアプリケーションの既定のインスタンス、`_defInst_` に接続しています。

クライアントはまた `NetConnection.connect` コマンドの `instanceName` 値によって定義される特定のアプリケーション・インスタンスに接続できます。

```
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しています。

特定の目的のためにアプリケーション・インスタンスを使用したいかも知れません。例えば、異なる話題を扱う“ルーム”を使ったチャット・アプリケーションで、グループ同士にやりとりさせることなく、違ったグループを同じアプリケーションにアクセスさせたいという場合です。そのためには、下記のように、同時にチャット・アプリケーションの多重インスタンスを作ります。

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

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

それぞれのアプリケーション・インスタンスにはユニークな名前をつけます。親アプリケーションのように、インスタンスはサーバー上に定義された自分のディレクトリを必要としません。しかし、ストリームや共有オブジェクトなどのアプリケーション・リソースは、各インスタンスから独立しており、アプリケーションで構成されたストリームや

sharedObject のディレクトリ下にある自分自身のディレクトリに保持されます。

アプリケーション・インスタンスを使用するもうひとつの理由は、アプリケーションによって生成された、記録されたストリームや共有オブジェクトの衝突を避けるためです。上記の例では、例えば、room_01 で作られたストリームや共有オブジェクトは room_02 で作られたストリームや共有オブジェクトとは別のもので、逆に、たとえ両方のインスタンスが同じアプリケーション、chat_App を作動させていたとしても、それらは別個のもので、

例えば以下のコードで support アプリケーションが CustomerInfo オブジェクトという名の 2 つの共有オブジェクトを作成したとしても、support アプリケーションのそれぞれのインスタンスはそれ自身の CustomerInfo オブジェクトにだけアクセスします。また session1 が使用する CustomerInfo 内のデータは、session2 が使用する CustomerInfo 内のデータとは異なるものです。

```
//アプリケーション" support"のひとつのインスタンス
```

```
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);
```

```
//アプリケーション" support"の別ののインスタンス
```

```
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);
```

このマニュアルのサンプルの多くは、インスタンス名 room_01 を使用していますが、作成するアプリケーションで意味をなすインスタンス名のどんな文字でも使用することができます。インスタンス名を動的に作り出す例として、/tutorial_textchat ディレクトリにある tutorial_textchat fla サンプルファイルを見てください。

アプリケーション・インスタンスに関連するいくつかの設定を構成することができます。Application.xml ファイルでは、インスタンスがサーバーによってアンロードされる前の最大アイドル時間を構成できます。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ユーザー数や仮想ホストがロードできるインスタンス数を構成できます。

インスタンス名の使用に関する詳細は、*Client-Side Communication ActionScript Dictionary* の NetStream.publish 項を参照してください。多重アプリケーションが可能になるリモート共有オブジェクトの作成に関しては、*Client-Side Communication ActionScript Dictionary* の SharedObject.getRemote 項を参照してください。

Flash Communication Server が使用するファイル形式

Flash MX で作成し使用する FLA、SWF、SWD などのファイル形式に加え、Flash Communication Server では以下のファイル形式を作成し、使用します。

ASC と JS ファイル は記述したり、compenst.asc のように Flash Communication Server で提供されるサーバー・サイド・スクリプト・ファイルです。JavaScript エディターを使ってサーバー・サイド・スクリプトを作成し、アプリケーション・ディレクトリ内のアプリケーション用に作ったサブディレクトリ、例えば /applications/chat_app か、

/applications/chat_app/scripts のスクリプト・サブディレクトリにスクリプトを置きます。

Flash Communication Server は、/scriptlib ディレクトリに、コンポーネントや Flash Remoting サービスへのサーバー・サイド・スクリプトを含むスクリプト・ライブラリを提供します。Flash Communication Server でコンポーネントや Flash Remoting を使用すると、/scriptlib からアプリケーション・スクリプト・ファイルに適切なスクリプトがインクルード、ロードされます。/scriptlib ディレクトリの場所は Application.xml 構成ファイルの<ScriptLibPath>タグ内に特定されます。components.asc ファイルをロードしてコミュニケーション・コンポーネントを使用する際の情報は、作業環境を作るを参照してください。Flash Remoting サービスの使用に関する情報は、チャプター 7、159 ページの“Application Server Connectivity”を参照してください。

FLV と IDV ファイル は記録されたストリームとそれに関連したインデックス・ファイルです。サーバーがストリームを生成すると、サーバーはまた、アプリケーション・ディレクトリ内のストリーム・ディレクトリを生成し、アプリケーション・インスタンスに特定のサブディレクトリ(例えば、/applications/chat_app/streams/instance2)に FLV と IDX ファイルを保存します。

SOL、SOR、FSO ファイル はクライアントやサーバー、またはその両方にくつつく共有オブジェクトのファイルです。共有オブジェクトの場所は共有オブジェクトが何の種類かによって変わります。Flash Communication Server がストリームや共有オブジェクトのファイルをどこに保存するかについての情報は、ファイル形式とパスを参照してください。

サーバーに接続する

Flash Communication Server のインスタンスに接続するには、サービスを開始させ、クライアント・サイド・スクリプトで new NetConnection と NetConnections.connect コマンドを出します。このセクションではこれらの作業について説明します。

Tip: コンポーネントを使用してサーバーに接続できたとしても、このセクションに書かれているように、初めは ActionScript を使った接続方法を学ぶべきです。

サービスを開始させる

サーバーがまだ開始されていない場合は、手動で開始できます。Windows のスタート・メニューから、プログラム> Macromedia Flash Communication Server MX>Start Service を選択します。Windows でサービスが作動しているか確かめるには、タスクマネージャを開いてプロセス・タブに Flash.Com.exe と FlashComAdmin.exe の両方がリストされていることを確認します。

UNIX では、シェル・ウィンドウを開いて、Flash Communication Server をインストールしたディレクトリに移動し、ルート・ユーザーとして、fcsmgr server start と入力します。

サーバーへの接続を開く

Flash MX オーサリング環境で新規ファイルを開き、サーバーに接続するクライアント・サイド ActionScript を加えます。

Flash Communication Server に接続する

1. 新規 Flash ムービーで、以下のコマンドを出すことにより接続を開く準備を始めます。

```
my_nc = new NetConenction();
```

2. つづいて以下のコマンドを実行します。

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

NetConnection.connect(オプションのパラメータは削除)のこの基本的なシンタックスで、targetURI は、接続がなさ

れたとき作動する Flash Communication Server 上のアプリケーションの Uniform Resource Identifier(URI)です。
TargetURI を特定するには、次のフォーマットのひとつを使用します(ブラケットのアイテムはオプションです)。

`rtmp://localhost[:port]/appName[/instanceName]` (localhost の使用は、サーバーがローカル・コンピュータで作動していることを表します)。

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

targetURI のシンタックスのサンプルでは、接続のプロトコルとして rtmp(Real-Time Messaging Protocol)を明示しなくてはならないことに注目してください。これを抜かすと Flash Player はアプリケーション・サーバーに HTTP 接続がしたいのだと仮定して、接続は失敗します。

既定の仮想ホストでない仮想ホストに接続したい場合は、host に仮想ホスト名を明示する必要があります。サーバーがローカル・マシンで作動しているなら、URI のホスト名には"localhost"が使用できます。これはアプリケーションの開発時には便利です。

例として、以下のコードでは new NetConnection コンストラクタを使用して、新しい接続オブジェクトを作成し、my_nc.connect をコールすることでオブジェクトはサーバーに接続しています。

```
//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 で作動する Flash Communication Server 上にある  
  
//appName というアプリケーションのインスタンス appInstance に接続する  
  
my_nc.connect("rtmp://myServer.myDomain.com/appName/appInstance");
```

パブリッシュしストリームを再生する

コミュニケーション・アプリケーションの通例の機能は、公開(配信)しストリームを再生することです。Flash Communication Server のインストールには、/doc_connect ディレクトリに doc_connect fla というサンプル FLA ファイルが含まれており、これは 2 人のクライアント間でサーバーを通してビデオ・ストリームを送信するものです。サンプル・ファイルを見て、自分でこの簡単なアプリケーションを再作成することで、Flash Communication Server への接続方法を学び、どのようにストリーミングが働いているか見ることができます。このサンプルには、Flash Communication Server を使用したクライアント-サーバー・コミュニケーションを実行する ActionScript の使用方法の説明が含まれています。自分自身のプログラムを書き始める前に、チャプター 2、23 ページの"Flash Communication Server の構造概観"を参照してください。

サンプルの接続ファイルを試す

doc_connect ファイルを見れば、ムービーを初期化し、サーバーに接続、パブリッシュしストリームを再生する方法が分かります。またサンプル・ファイルの保存場所についても説明しています。

*Note:*このファイルでは、サーバーが作動していることを前提としています。サーバーの開始については、サービスを開始させるを参照してください。

ソース・ファイルを見るには

.../help_collateral/doc_connect ディレクトリにある doc_connect fla を開きます。

サンプルを動作させるには

1. Flash Communication Server のインストール・ディレクトリのアプリケーション・ディレクトリに doc_connect デ

ディレクトリを作成する。

2. .../help_collateral/doc_connect ディレクトリにある doc_connect.swf ファイルを開きます。

Note: アプリケーションが動作していないようなら、サーバーが作動していることを確認し、サービスを開始させるを参照してください。

オーディオ、ビデオ・デバイスの許可表示が出た後、カメラからのライブ・ストリームとサーバーから返されたパブリッシュされたストリームの2つの画像が現れます。



サンプルの再作成

このサンプルは、Flash Communication Server が作動しているコンピュータ上で Flash MX オーサリング環境を使用しているものと仮定しています。そうでない場合は、次のステップで“localhost”のサーバーの URL を置き換えます。このサンプルはまたサービスは作動していることを前提としています。

このサンプルのユーザー・インターフェイスを作成するには

1. Flash オーサリング環境で、ファイル>新規を選択し、新規ファイルを開きます。
2. ライブラリ・パネルが表示されていないなら、ウィンドウ>ライブラリを選択して表示させます。
3. ライブラリ・パネルの上方右のオプション・メニューをクリックして新規ビデオを選択し、ライブラリに埋め込みビデオ・オブジェクトを追加します。



4. ライブラリからステージ上へ埋め込みビデオ・オブジェクトをドラッグし、プロパティ・インスペクタで Live_video と Published_video の名前をつけます。



5. Published_video オブジェクトのサイズを 200 x 150 に修正します。



6. まだの場合は、アプリケーション・ディレクトリに doc_connect ディレクトリを作成します。このディレクトリに doc_connect.fla 名でファイルを保存します(既定では、アプリケーション・ディレクトリは Flash Communication Server インストール・ディレクトリにあります)。

このサンプルの ActionScript を書くには

1. タイムラインのキーフレーム(フレーム 1)を選択してアクションパネル(ウィンドウ>アクション)を開きます。
Note: サンプルを再作成している間、全ての ActionScript はこの最初のキーフレームにアタッチし、ステージに作ったオブジェクトにアタッチしないようにしてください。

2. 既定のカメラを取得し、埋め込みビデオオブジェクト Live_video にアタッチします。

```
client_cam = Camera.get();  
Live_video.attachVideo(client_cam);
```

3. Flash Communication Server に接続し、接続が成功したかどうかを示すトレース・メッセージを表示させ、room_01 という名前の、doc_connect アプリケーションのインスタンスを開く接続関数(ファンクション)を作成します。Real-Time Messaging Protocol、rtmp を明示することを忘れないようにしてください。

```
function doConnect(){  
    client_nc = new NetConnection();  
    client_nc.onStatus = function(info){  
        trace("Level: " + info.level + " Code: " + info.code);  
    }  
    client_nc.connect("rtmp://localhost/doc_connect/room_01");  
}
```

Note: SWF が Flash Communication Server の動作している同じコンピュータ上にある場合は、rtmp://localhost/doc_connect/test の短縮版として、rtmp:/doc_connect/test を使用できます。この相対パスによる使用方法により、コードを書きかえることなく別のサーバーに移すことができます。また、Flash Communication Server が作動するコンピュータ上で Flash MX オーサリング環境を使用していない場合は、"localhost"をサーバーの URL に置き換えることを忘れないようにしてください。

4. ネットワーク・ストリーム out_ns を作成し、そのストリームをカメラにアタッチした後、myTestStream 名でストリームをパブリッシュする関数を作成します。

```
function publishMe(){  
    out_ns = new NetStream(_root.client_nc);  
    out_ns.attachVideo(client_cam);  
    out_ns.publish("myTestStream");  
}
```

Note: `out_ns.publish("myTestStream")`文はオプションのパラメータ `howToPublish` を省略しています。このパラメータがないと、サーバーは自動的にライブでストリームをパブリッシュします(つまり、パブリッシュ中はストリームは記録されないということです)。

- 2 つめのネットワーク・ストリーム `in_ns` を作成し、`Published_video` ビデオ・オブジェクトにこのストリームのコンテンツをアタッチした後、サーバーによってパブリッシュされるストリーム `myTestStream` を再生することで、パブリッシュされたストリームを再生する関数を作成します。

```
function playMe(){
    in_ns = new NetStream(_root.client_nc);
    Published_video.attachVideo(in_ns);
    in_ns.play("myTestStream");
}
```

- 作成した関数を呼ぶコマンドを書きます。

```
//サーバーに接続する
doConnect();
//ライブ・ストリームをパブリッシュする
publishMe();
//サーバーからのストリームを再生する
playMe();
```

- ファイルを保存します。

サンプル・アプリケーションをテストするには

- ファイル>パブリッシュ設定>を選択し、Flash と HTML を選び、パブリッシュをクリックした後、OK をクリックします。
- 制御>ムービープレビューを選択します。
ステージ上に 2 つのビデオ・ウィンドウが現れ、どちらも同じ映像を表示します。



- ブラウザでムービーがどのように表示されるかを見るには、ファイル>パブリッシュプレビュー>デフォルトを選択するか、コントロールキー+F12(Windows)、コマンドキー+F12(Macintosh)を押します。

このサンプルは、少ないコードが Flash Communication Server を使用してどのようにクライアント-サーバーコミュニケーションを実行するかを示すものです。

開発環境をセットアップし、初めてのコミュニケーション・アプリケーションを作成しました。これからこれらのリソースを使って Flash Communication Server を学んでいきます。

- Flash Communication Server の Welcome ページは、簡単なアプリケーション作成のチュートリアルを含むほか、上級のサンプル・アプリケーションを見ることができます。
- チャプター 2、23 ページの”Flash Communication Server の構造概略”では、さらに詳細な概念が説明されています。
- チャプター 3、35 ページの”コミュニケーション・オブジェクトの使用”では、Flash Communication Server のクライアント・サイド、サーバー・サイド、共有オブジェクトが説明されています。
- チャプター 4、51 ページの”アプリケーションのデバッグと監視”では、Flash Communication Server に付属するデバッグ・ツールの使用方法が説明されています。
- チャプター 6、87 ページの”コミュニケーション・コンポーネントの使用”では、コミュニケーション・コンポーネントを使った完全なアプリケーションの作成方法がステップ・バイ・ステップで説明されています。

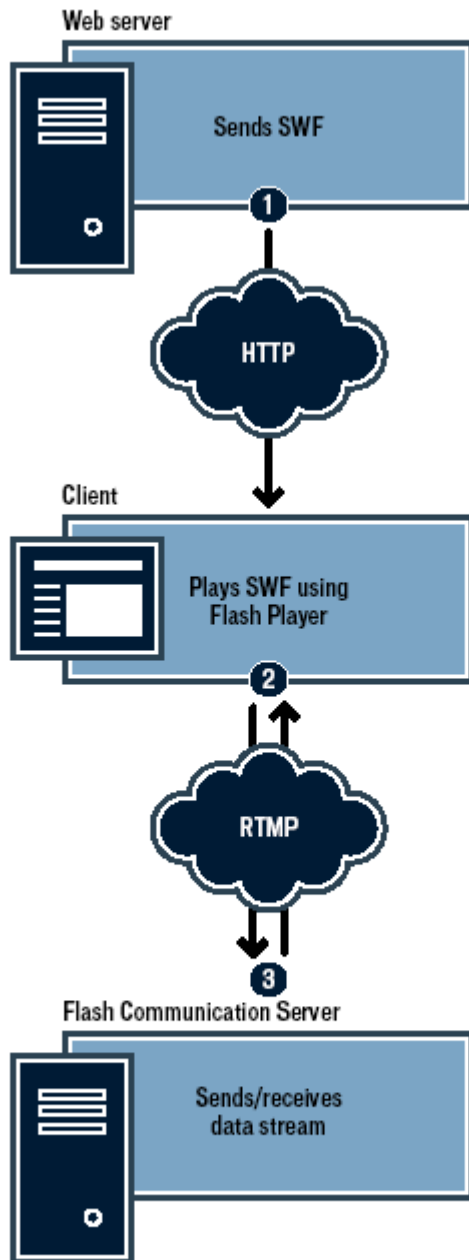
CHAPTER 2

Flash Communication Server の

構造概略

Macromedia Flash Communication Server MX 1.5 のプラットフォームは 2 つの部分から構成されます。それは、コミュニケーション手段を提供するサーバーと Macromedia Flash Player です。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 によって動作するクライアントの Macromedia Flash ムービー(SWF)と、そのクライアントと通信するサーバー・コンポーネントから成ります。サーバー・コンポーネントは主に、Flash Communication Server が動作するサーバー上に作成されたアプリケーション・フォルダから構成されます。このフォルダは、コミュニケーション・アプリケーションが使用する Server-Side Communication ActionScript(ASC)ファイルや他のリソース・ファイルを含むことができます。

サーバーと Flash クライアント・ムービーは Real-Time Message Protocol(RTMP)を使って繰り返し通信します。その典型として、Flash クライアントは web サーバーによって HTTP を通して Flash Player に配信されます。その後 Flash クライアントは、RTMP を使って Flash Communication Server に繰り返し接続を確立させ、データのストリームがクライアントとサーバー間を絶え間なく流れることを可能にします。



多数のユーザー(Flash クライアント)は、Flash Communication Server 上で動作している同じアプリケーションに接続でき、Flash Communication Server は接続したユーザーの間でコミュニケーション・チャネルのようにふるまいます。



Flash Communication Server はクライアントにコミュニケーション・チャネルを提供します。

ストリームと共有オブジェクトについて

伝統的なクライアント-サーバー・アプリケーションでは、サーバーはある種のトランザクションを実行するために使用されます。クライアントがリクエストを出し、サーバーがデータベースを調べたり、リソースに基づく計算をして、その結果をクライアントに返すのです。クライアントとサーバーの接続は、その処理を完了するまで長く維持されます。

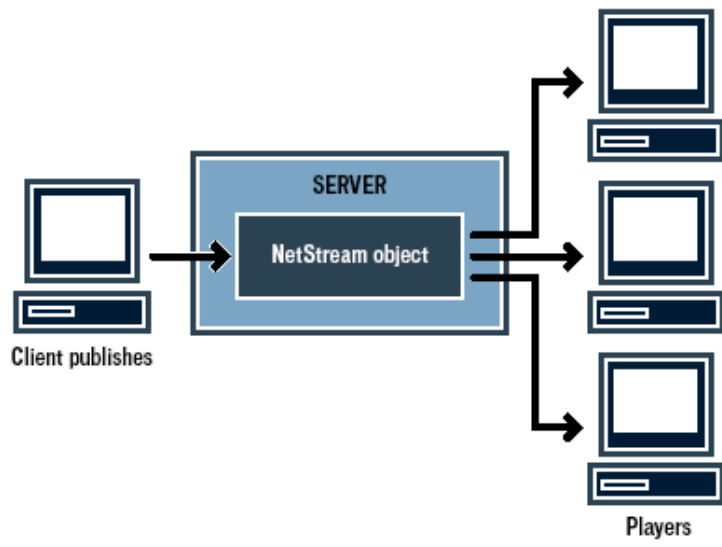
Flash Communication Server を使用してトランザクションを実行することができると、核の部分は、多数の接続したユーザー、クライアントの活動を調整したり、サーバー・サイド・データを送ったりする、インターアクションの管理に使用されます。Flash Communication Server は、ユーザーのインターアクション管理のプロセスをシンプルにする2つのモデル、ストリームと共有オブジェクトを提供します。

ストリームの概略

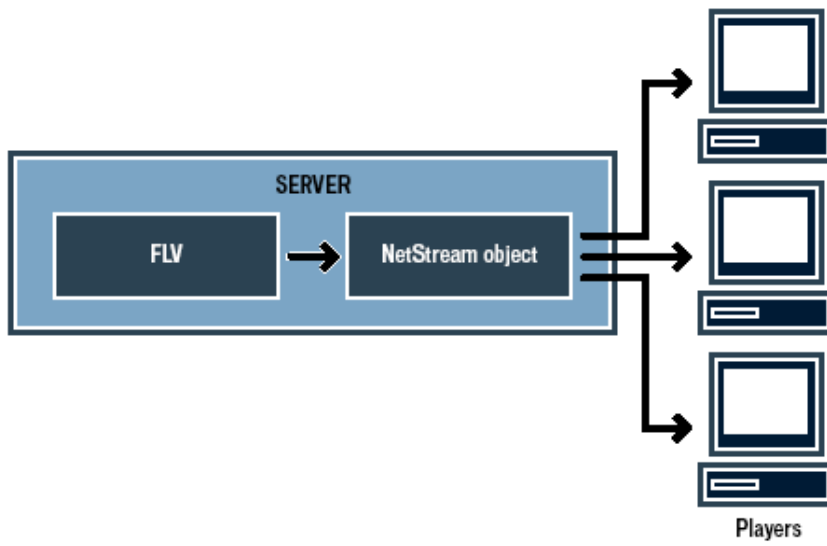
ストリームは、クライアントからサーバーへ、サーバーからクライアントへ流れる、同調したオーディオやビデオ、その両方のデータの、時間に基づく流れです。ストリームは、ストリームを使用するアプリケーション開発を簡単にするパブリッシュとサブスクライブ・モデルです。ビデオ・ストリームをパブリッシュし再生する例は、パブリッシュしストリームを再生することです。

パブリッシュされたストリームは、例えばビデオ・チャット・アプリケーションなどでリアル・タイムに再生することが可能で、記録することや後で再生することもできます。

記録されたストリームは、Flash Video(FLV)形式で保存されます(ビデオはもちろん、記録されたストリームもデータ・メッセージを含んでいます)。また Sorenson Squeeze などのサード・パーティ製ビデオ・エンコーディング・ユーティリティを使って今あるデジタル・ビデオやオーディオ・ファイルから FLV ファイルを作成したり、Flash MX から書き出すことができます。このように Flash Communication Server を使用して、記録される前の内容をストリームに流すこともできます。



Live stream



共有オブジェクトの概略

コミュニケーション・アプリケーションを作成するとき使用できる共有オブジェクトには2つの基本タイプ、ローカルとリモートがあります。ローカル共有オブジェクトは、“Flash cookies”ととらえることもできます。それによりオフライン時や保存したい場合にユーザーのコンピュータにデータを保存することができます。ローカル共有オブジェクトは、Flash Player の特徴であり、Flash Communication Server は必要としません。

リモート共有オブジェクトは Flash Communication Server により管理され、メッセージやデータ同調、データ保存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し読み込み、その共有オブジェクトに変化があったときはいつでも更新を受け取ります。またメッセージは、リモート共有オブジェクトに接続している全てのクライアントに送られます。リモート共有オブジェクトは、アプリケーションのセッションを通して持続します。



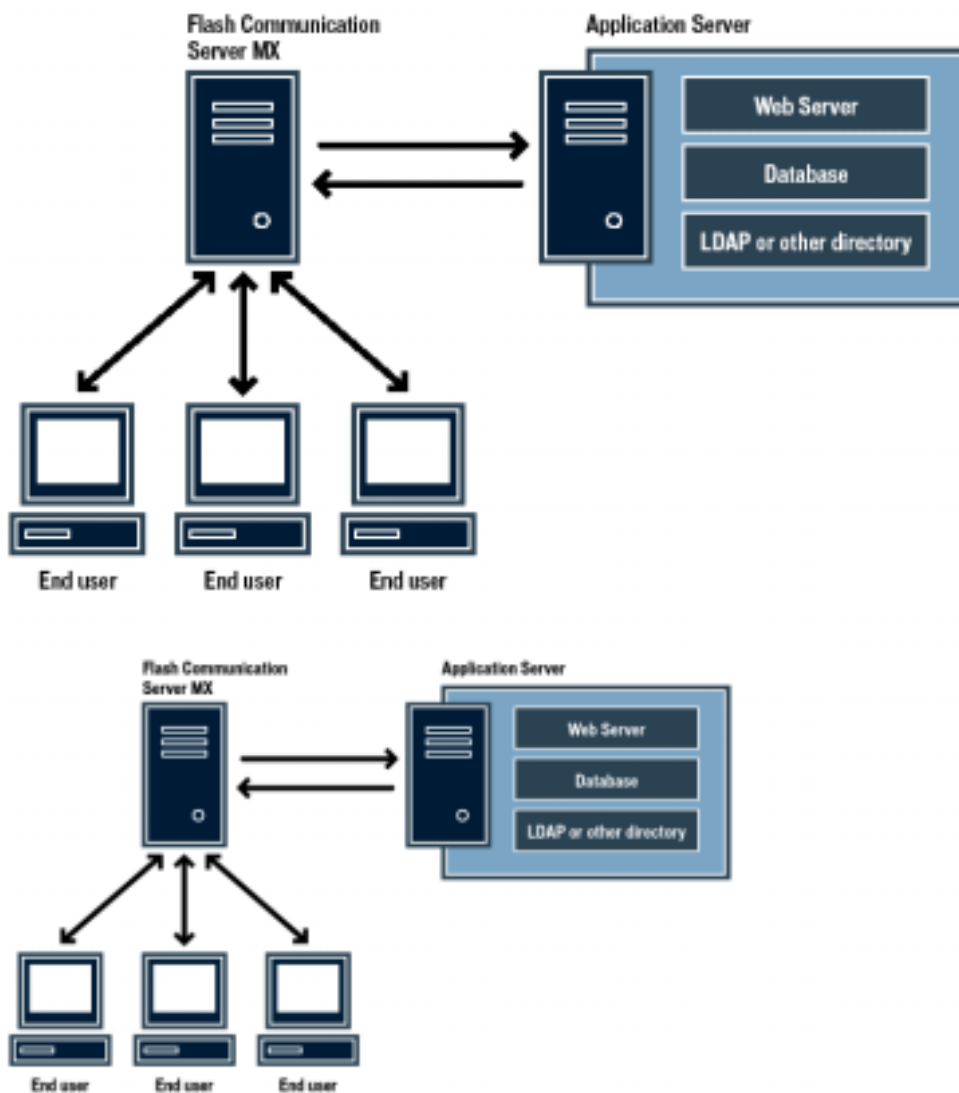
共有オブジェクトはクライアントにデータ保管と同調サービスを提供します。

共有オブジェクトの働きに関する情報は、共有オブジェクトの流れと SharedObject オブジェクトを参照してください。

共有オブジェクトの使用例は、Flash Communication Server の Welcome ページ、Shared Ball サンプルをご覧ください。

外部のデータ・ソースへ接続する

ストリームと共有オブジェクトによって提供されるコミュニケーション・モデルに加え、Flash Communication Server はまた、web サービスやリレーショナル・データベースといった、Flash Communication Server アプリケーション以外の外部のデータ・ソースと交信できます。例えば、サーバー・サイド ActionScript を書いて、web サービスや ColdFusion アプリケーションに接続し、名前や電話番号のリストを取得できます。問い合わせの結果は共有オブジェクトに置くことができます。



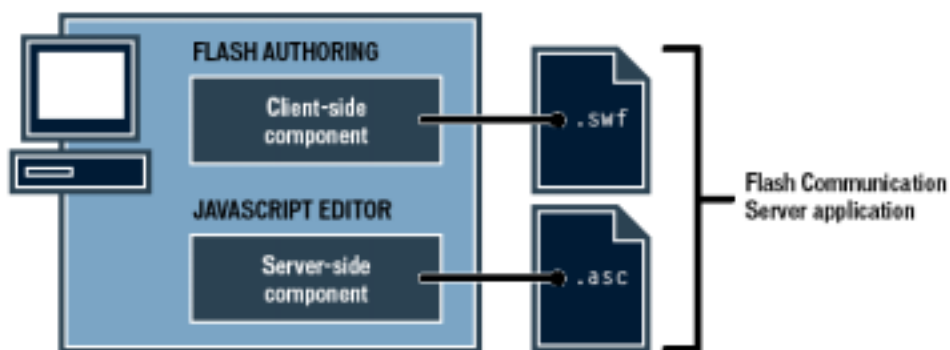
Flash Communication Server は外部のデータ・ソースと交信できます。

外部のデータ・ソースへの接続に関する情報は、[チャプター 7、159 ページの「アプリケーション・サーバーへの接続性」](#)を参照してください。

アプリケーションを作成し配置するワークフロー

Flash Player で作動するアプリケーションのクライアント・コンポーネント作成には、Flash MX オーサリング・ツールを使用します。Flash ムービー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し、Flash Communication Server に接続し、通信を管理する ActionScript を含みます。Client-Side Communication ActionScript の詳細は、*Client-Side Communication ActionScript Dictionary* を参照してください。

サーバー・コンポーネントは、最小で Flash Communication Server が作動するコンピュータ上に作成したアプリケーション・フォルダで構成されます。このフォルダは、共有される状態情報の管理を可能にする Server-Side Communication ActionScript API(ASC)ファイルを含むことができ、多数のユーザー間でリアル・タイムの通信を調整する論理を供給し、必要なら外部のリソースと通信します。ASC ファイルは JavaScript エディターなど、どんなテキスト・エディターでも書くことができます。サーバー・サイド ActionScript に関する情報は、*Server-Side Communication ActionScript Dictionary* を参照してください。

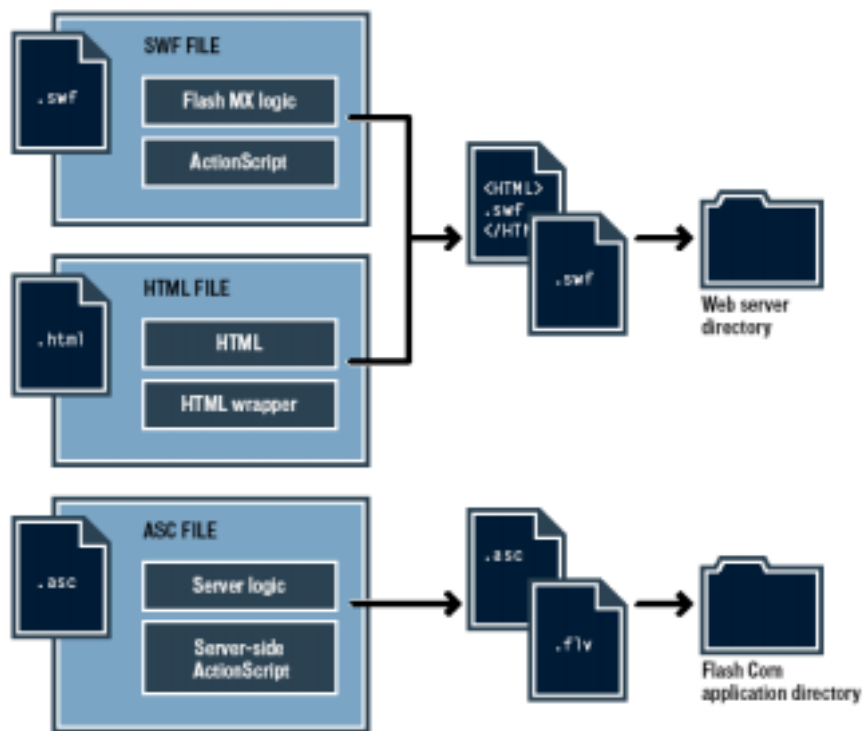


アプリケーションの配置と管理

アプリケーションを配置するには、クライアントとサーバー・ファイルは、適切な場所にパブリッシュされている必要があります。Web 上に配置されるアプリケーションが使う Flash ムービーや HTML コードなどのクライアント・ファイルは、web サーバーのディレクトリにパブリッシュされていなければなりません。

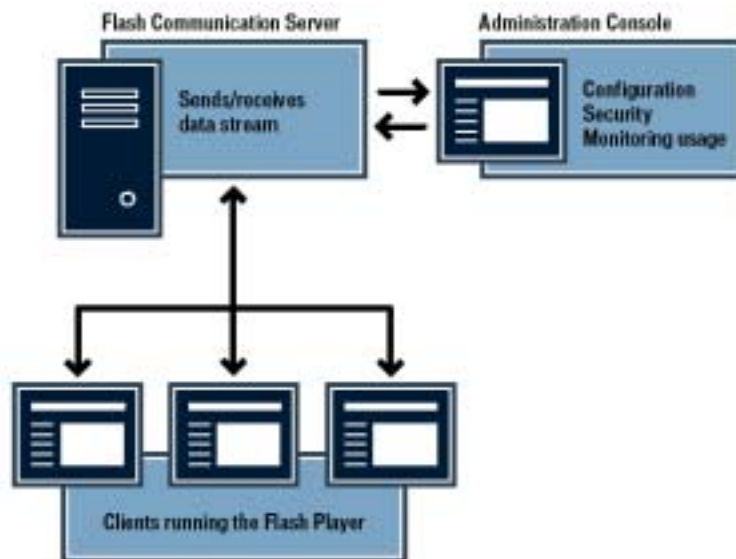
Note: Flash クライアント・ムービーは、email に添付するくらい簡単にユーザーに送ることができます。Flash Communication Server アプリケーションは、web サーバーを通らない RTMP で通信するので、web サーバーは必要ありません。

ASC ファイル、記録されたストリーム(FLV)ファイルやその他のサーバー・サイド・リソース・ファイルといったサーバー・ファイルは、サーバー上で定義されたアプリケーション・ディレクトリにパブリッシュされます。



アプリケーション・ディレクトリについての情報は、[Chapter 1, 11 ページの“始めよう”](#)を参照してください。

管理者は Administration Console を使用して Flash Communication Server を構成し、システム・セキュリティのセットアップ、管理、サーバーの開始と停止、ユーザーの追加などを行います。Flash Communication Server での管理者の作業に関する情報は、“*Managing Flash Communication Server*”を参照してください。



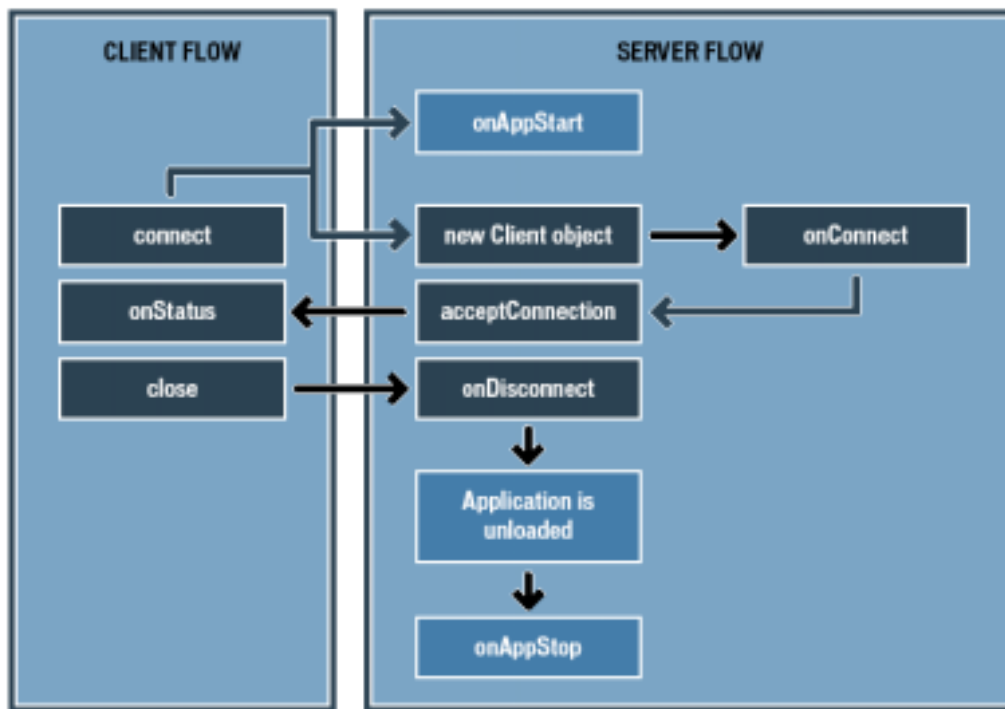
アプリケーションの流れ

このセクションでは、Flash クライアントはどのようにしてサーバーに接続しているか、共有オブジェクトにどのようにしてアクセスしているか、リモート・メソッドをどのようにして呼び出しているのかについてさらに詳しく説明します。

ユーザーが Flash SWF ファイルを作動させ SWF ファイルがサーバーに接続すると、まだ動作していなければサーバーはアプリケーションをロードし、アプリケーション・インスタンスを作成します。サーバーは接続を許可し、サーバー上でクライアント・アプリケーションの代わりをつとめる新しい Client オブジェクトを作成し、与えられたサーバー・サイド・スクリプトを実行します。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作業を行います。次のセクションでは、この後のイベントについてさらに詳しくみていきます。

接続の流れ

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・インスタンスがまだ作動していないならサーバーは onAppStart を呼びます。次に新しく作成された Client オブジェクトによってサーバー・サイドの onConnect メソッドが呼び出されます。このメソッド内の設定で、接続を許可するか拒否するか決定します。クライアント・サイドに戻って、onStatus メソッドが呼び出され、接続が許可されたか拒否されたかを通知します。クライアントが接続を閉じると、サーバー・サイドの onDisconnect メソッドが呼び出されます。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。



接続の流れ

Note: クライアント・アプリケーションだけがコミュニケーション・セッションを初期化できます。クライアントもサーバーも、ステータス・メッセージの送受信をし、ストリームの開閉、共有データの保存を行い、ネットワーク接続を終了させます。

リモート・メソッドの呼び出し

接続が成功すると、クライアントは、サーバー・コンポーネントが定義したリモート・メソッドを呼び出すことができます。サーバー・アプリケーションはまた、Flash クライアント・ムービーで定義されている遠隔の ActionScript メソッドを呼び出すこともできます。

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した各 Flash ムービーは、サーバー・サイドの Client オブジェクトのインスタンスがその代理となります。それぞれの Client オブジェクト・インスタンスは、その結果、遠隔から使用できるメソッドと開発者が定義したメソッドを連携させることができます(また、prototype プロパティを使って、全てのクライアントで使用可能なリモート・メソッドを作成することもできます。サンプルは、*Server-Side Communication ActionScript*

Dictionary の Client(object)項を参照してください。

Flash ムービーはその後、Netconnection オブジェクトの call メソッドを使って、遠隔で定義されたメソッドを呼び出すことができ、サーバーが返す結果を管理するコールバック・オブジェクトを指定することもできます。サーバー上では、クライアントの呼び出しに応えるメソッドが呼び出され、結果がクライアントに戻されます。

下の図は、クライアントからサーバーで定義されたメソッドを呼び出した例を説明したものです。図内では、nc という名前の NetConnection インスタンス上で遠隔のメソッド doThis を呼び出しています。doThis メソッドは、クライアントの代わりに果たす Client オブジェクトのインスタンス、この場合は ClientObj と連携しています。



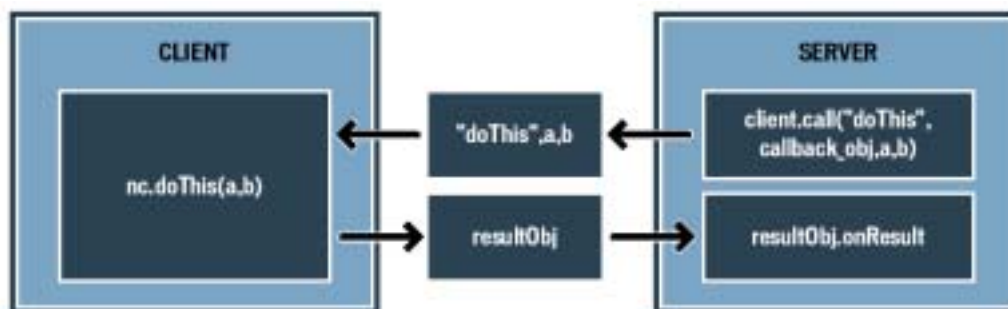
クライアントからサーバーへの、クライアントに結果を戻す、リモート・メソッド・コールの流れ

遠隔でのサーバー・メソッドの呼び出しに関する詳細は、*Server-Side Communication ActionScript Dictionary* の Client,"commandName"項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

サーバーからクライアントで定義されたメソッドを呼び出すプロセスは、その逆の場合によく似ています。クライアント上で、サーバーに接続するために使用された NetConnection オブジェクト・インスタンスに、ユーザー定義のメソッドをアタッチできます。その後そのメソッドは、Client オブジェクトの call メソッドを使ってサーバーから呼び出すことができます。

下の図は、サーバーからクライアント・メソッドを呼び出す過程を説明したものです。図内では、nc という名前の NetConnection オブジェクトのインスタンスと、クライアント・サイドのメソッド doThis は連携しています。サーバーは Client オブジェクトの 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、結果を返すハンドラ・オブジェクトを指定することもできます。再びクライアントは結果を戻し、サーバーの onResult ハンドラが、クライアントに送ったコールバック・オブジェクト上で呼ばれます。



サーバーからクライアントへの、サーバーに結果を戻すコールの流れ

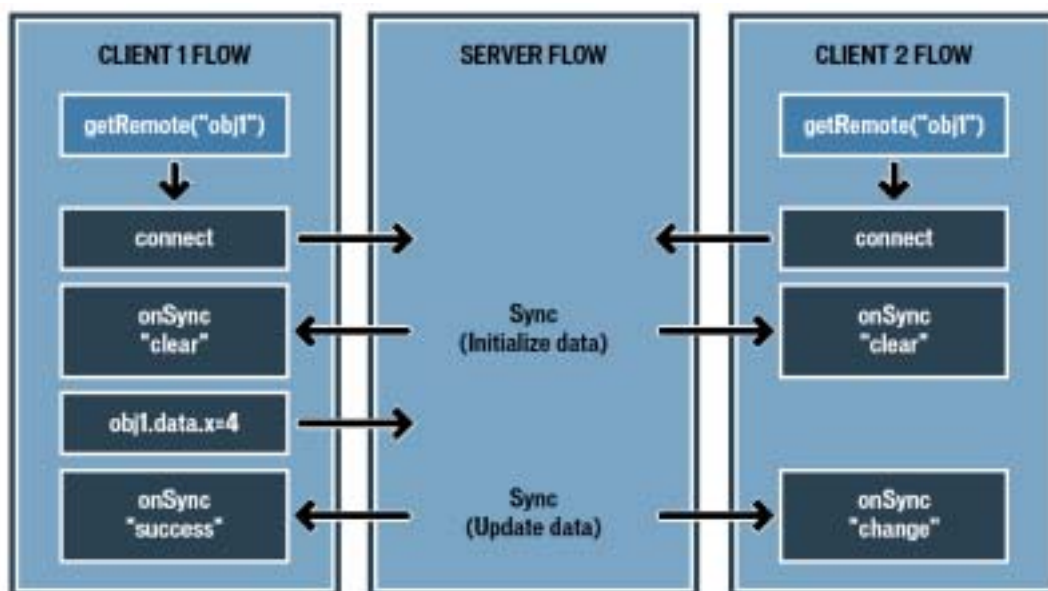
共有オブジェクトの流れ

共有オブジェクトは、多数のユーザー間でデータを共有する場合の開発者の作業を簡単なものにします。Flash クライアント・ムービーは、リモート共有オブジェクトへの参照を返す `SharedObject.getRemote` コマンドを出すことで離れた共有オブジェクトを読み込みます。その後クライアントは `SharedObject.connect` コマンドを出すことで、`Netconnection` オブジェクトにリモート共有オブジェクトを接続させます。

クライアントが一度共有オブジェクトに接続したら、サーバーは、クライアント上に定義された `SharedObject.onSync` メソッド・ハンドラによって管理される、同調メッセージを共有オブジェクトへ発信します。クライアントやサーバー、または他のムービー・インスタンスが共有オブジェクトに変化を与えると、サーバーは再び共有オブジェクトに向けて同調メッセージを発信します。

各同調メッセージには、共有オブジェクトに起こる変化の状態を明示するコードが含まれます。コードは環境により変化します。例えば、クライアントが初めて共有オブジェクトに接続すると、同調メッセージを伴うコードは値 `clear` を持ちます。値 `success` は、クライアントが共有オブジェクトの内容修正に成功したことを示します。コードやその記述に関する詳細は、"*Client-Side Communication ActionScript Dictionary*"の `SharedObject.onSync` 項を参照してください。

下の図は、Client1、Client2 の2人のクライアントが、同じリモート共有オブジェクト、obj1 に接続したところを表しています。各クライアントが初めて共有オブジェクトに接続すると、`clear` 同調メッセージがサーバーによって返されます。その後 Client1 は obj1 のデータ・プロパティ `x` を"4"に修正します。Client1 は、共有オブジェクトの変更が成功したことを示す同調メッセージ `success` を受け取ります。Client2 は共有オブジェクト obj1 が修正されたことを示す同調メッセージ `change` を受け取ります。



共有オブジェクトの流れ

CHAPTER 3

コミュニケーション・オブジェクトの使用

Macromedia Flash Communication Server MX 1.5 は、2つのアプリケーション・プログラム・インターフェイス(API)、クライアント・サイド API とサーバー・サイド API を提供します。このチャプターでは、クライアント・サイド・オブジェクト、サーバー・サイド・オブジェクトについて説明し、これらのオブジェクトが通信中にどのようにして1つに組み合わさるのかを説明します。また共有オブジェクトについても扱います。共有オブジェクトは、ユーザー間、アプリケーション・インスタンス間、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保持しています。

Flash Communication Server オブジェクトの核となる部分の概略を説明した後、いくつかのオブジェクトについて重要な詳細についても説明します。それは、Client オブジェクトを守り NetConnection オブジェクトの安全を確保するだけでなく、Application、Camera、Microphone、NetStream、Stream、System、Video オブジェクトを最適化する際に推奨されるものです。

Flash Communication Server オブジェクト

クライアント・サイド API は、Camera、Microphone、NetConnection、NetStream、SharedObject、Video のオブジェクトを提供します。これらオブジェクトの使用に関する詳細は、*Client-Side Communication ActionScript Dictionary* を参照してください。サーバー・サイド API は、Application、Client、NetConnection、SharedObject、Stream のオブジェクトを提供します。これらオブジェクトの使用に関する詳細は、*Server-Side Communication ActionScript Dictionary* を参照してください。

ここで分かるように、例えば NetConnection などのいくつかのオブジェクトは、両方の API で同じ名前を持っています。しかし、同じ名前のクライアント・サイド・オブジェクトとサーバー・サイド・オブジェクトは、同じ機能を提供する訳ではありません。加えてクライアント・サイド・オブジェクトにはサーバー・サイド・オブジェクトと似たものもありますが、そうでないものもあります。以下のセクションでは、クライアント・オブジェクトとサーバー・サイド・オブジェクトについて明確に述べ、それらがどのようにして協同してクライアント-サーバー・コミュニケーションを可能にしているかを説明します。

クライアント・サイド・オブジェクト

これらのオブジェクトはクライアント・サイド ActionScript でのみ使用されます。これらのオブジェクトに関する詳細は、*Client-Side Communication ActionScript Dictionary* を参照してください。

Camera オブジェクト クライアント・サイドの Camera オブジェクトにより、Macromedia Flash Player の作動するあらゆるコンピュータに接続されたビデオ・カメラからビデオをキャプチャーできます。Flash Communication Server と併用する場合、このオブジェクトはキャプチャーしたビデオを送信し、表示し、記録することができます。これらの機能を使って、ビデオ会議、インスタント・メッセージングといったコミュニケーション・アプリケーションを開発できます。

Microphone オブジェクト クライアント・サイドの Microphone オブジェクトにより、Flash Player が作動するあらゆるコンピュータに接続されたマイクからオーディオを取得できます。Flash Communication Server と併用する場合、このオブジェクトは取得したオーディオを送信し、再生し、記録することができます。これらの機能を使って、オーディオ版インスタント・メッセージング、他の人が後で再生できるレコーディング・プレゼンテーションといったコミュニケーション・アプリケーションを開発できます。

Microphone オブジェクトは、サーバーなしでも使用できます。例えば、ローカル・システムでスピーカーを通して、マイクからサウンドを出すことができます。

NetConnection オブジェクト クライアント・サイドの NetConnection オブジェクトによって、Flash クライアントは、Flash Communication Server 上で TCP ソケットを開き、Real-Time Messaging Protocol(RTMP)を使って絶え間ないデータのやりとりをすることができます。また NetConnection オブジェクトを使って、アプリケーション・サーバーに接続することもできます(チャプター 7、159 ページの”Flash Remoting を通じた接続”を参照してください)。しかしこのドキュメントでは、Flash Communication Server とやりとりする NetConnection オブジェクトの使用に焦点を当てています。

NetStream オブジェクト クライアント・サイドの NetStream オブジェクトは、クライアント・サイドの NetConnection オブジェクトによって可能になった接続を通して、Flash Player と Flash Communication Server 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 内のチャンネルのようなものです。このチャンネルは、NetStream.publish メソッドを使うとオーディオやビデオ、データのパブリッシュができます。また NetStream.play メソッドを使うとストリームや受け取ったデータを読み込むことができます。ライブ(リアルタイム)のデータをパブリッシュ

ュしたり再生したり、以前に記録したデータの再生も可能です。多くのクライアントは与えられたストリームを再生できますが、ストリームは一度にひとつのパブリッシャー(発行者)しかもてません。サーバー上に記録されたストリームの再生、保存についての詳細は、記録されたストリームを参照してください。

Local 共有オブジェクト クライアント・サイドのローカル共有オブジェクトによって、ゲームのハイスコアといった情報をユーザーのコンピュータに保存でき、その情報は、後から同じアプリケーションで利用できたり、そのコンピュータで作動している別のアプリケーションでも利用できます。ローカル共有オブジェクトの詳細な情報は、共有オブジェクトの理解を参照してください。

Video オブジェクト クライアント・サイドの Video オブジェクトによって、ステージ上にストリーミング・ビデオを表示できます。NetStream.play や Camera.get コマンドで再生、取得されたビデオを表示するには、ステージ上に Video オブジェクトを配置し、Video.attachVideo メソッドを使ってオブジェクトにビデオをアタッチします。

ステージに Video オブジェクトを配置するには

1. ライブラリ・パネルが見えない場合は、ウィンドウ>ライブラリを選択して表示させます。
2. ライブラリ・パネルの右上方のオプション・パネルをクリックして新規ビデオを選び、埋め込みビデオ・オブジェクトをライブラリに追加します。
3. ステージにビデオ・オブジェクトをドラッグし、プロパティ・インスペクタでユニークな名前をつけます。

サーバー・サイド・オブジェクト

これらのオブジェクトはサーバー・サイド ActionScript 内でのみ使用されます。これらのオブジェクトに関する詳細は、*Server-Side Communication ActionScript Dictionary* を参照してください。

Application オブジェクト サーバー・サイド Application オブジェクトは、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継続する Flash Communication Server アプリケーション・インスタンスについての情報を持っています。Application オブジェクトによって、接続を試みるユーザーを許可したり拒絶したり、クラスやプロキシを登録したり登録からはずしたり、アプリケーションが開始、または停止したときや、クライアントが接続、または解除したときに動作する関数を作成したりできます。

Client オブジェクト サーバー・サイド Client オブジェクトは、Flash Communication Server アプリケーション・インスタンスに、各ユーザーの接続を伝えます。Client オブジェクトはクライアント・サイドの NetConnection.call コマンドによって送られるメッセージを受け取ることができ、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は、各クライアントのバージョン、プラットフォーム、IP アドレスを明らかにするのに使用できます。Client オブジェクトを使用すると、Stream オブジェクトや SharedObject オブジェクトなど様々なアプリケーション・リソースのパーミッションを個別に取得、設定できます。詳細は動的アクセス管理の実行を参照してください。

NetConenction オブジェクト サーバー・サイド NetConnection オブジェクトにより、Flash Communication 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、他の Flash Communication Server、または同じサーバー上の別の Flash Communication Server アプリケーション・インスタンスの間での双方向の接続が作成できます。またサーバー・サイド NetConnection オブジェクトを使用して、さらにパワフルなアプリケーションを作成することもできます。例えば、アプリケーション・サーバーから天気の情報を得たり、他の Flash Communication Server やアプリケーション・インスタンスとアプリケーションのロードを共有できたりします。

このオブジェクトを使用すると、HTTP のような標準的のプロトコルを使用してサーバー間の通信をおこなうアプリケーション・サーバーに接続することができ、また Macromedia Real-Time Messaging Protocol(RTMP)を使用してオーディオやビデオ、データを共有する他の Flash Communication Server に接続することも可能です。

Flash Communication Server と Macromedia Flash Remoting を併用して、Macromedia ColdFusion MX、.NET、J2EE サーバーなどのアプリケーション・サーバーと通信することもできます。詳細は Flash Remoting サイト (www.macromedia.com/go/flashremoting)を参照してください。

リモート共有オブジェクト リモート共有オブジェクトは、クライアント上で作成されますが、サーバーにも使用できます。リモート共有オブジェクトにより、多くのクライアント間でリアルタイムにデータを共有でき、また同じアプリケーションでも違うアプリケーションでも後で実行できるデータを保持できます。クライアント・サイドリモート共有オブジェクトに関する詳細は、共有オブジェクトの理解を参照してください。

SharedObject オブジェクト サーバー・サイド共有オブジェクトにより、クライアント・サイドの共有オブジェクトや他の Flash Communication Server 上のオブジェクトと通信できます。サーバー・サイド共有オブジェクトに関する詳細は、共有オブジェクトの理解を参照してください。

Stream オブジェクト サーバー・サイド Stream オブジェクトにより、Flash Communication Server アプリケーションで各ストリームを管理できます。Flash Communication Server は、クライアント・サイド・スクリプトで NetStream.play や NetStream.publish メソッドが呼ばれると、自動的に Stream オブジェクトを生成します。また Stream.get メソッドを呼ぶことで、サーバー・サイド ActionScript にストリームを生成できます。ユーザーは同時に多くのストリームにアクセスでき、同時に多くの Stream オブジェクトをアクティブにすることもできます。

クライアント・サーバー・オブジェクトの通信

下記の”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間で確立できる潜在的な接続を表したものです。例えば、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 Client オブジェクトが生成されます。この Client オブジェクトはその後、その NetConnection オブジェクトのメソッドを呼びます。

下の表は、それぞれ関連し合うクライアント・サイドとサーバー・サイド・オブジェクトを示したものです。

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)

下の表では、左のクライアント・サイド・コールが、右のサーバー・サイド・コールを呼び出します。

クライアント・サイド・コール	サーバー・サイド・コール
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”)

下の表では、左のサーバー・サイド・コールが右のクライアント・サイド・コールを呼び出します。

サーバー・サイド・コール	クライアント・サイド・コール
my_client.call(“doThing”,myCallbackFcn,1,”foo”)	my_nc.doThing(1,”foo”)

<code>my_server_stream.send("doThing",1,"foo")</code>	<code>my_ns.doThing(1,"foo")</code>
<code>my_server_so.send("doThing",1,"foo")</code>	<code>my_so.doThing(1,"foo")</code>

共有オブジェクトの理解

共有オブジェクトは、異なるクライアント間や、Flash Communication Server 上で作動する異なるアプリケーション・インスタンス間で、また多くの Flash Communication Server 上で動作するアプリケーションを横断して、データを共有する手段です。Flash Communication Server は、ローカル、リモート、サーバー・サイドの3タイプの共有オブジェクトをサポートします。これらのオブジェクトについて下記に論じます。共有オブジェクトとの作業に関する情報は、チャプター5、80ページの”共有オブジェクト・ファイル”を参照してください。

ローカル共有オブジェクト

共有オブジェクトの1つの使用法は、ローカルにデータを保存しそれを使用することです。データはエンドユーザーのコンピュータに保存され、Flash アプリケーションは後から何度でもアクセスできます。ローカル共有オブジェクトはまた、アプリケーションが作動している間のみ使用できるもので、永続するものではありません。

ローカル共有オブジェクトは、Flash Communication Server への接続は必要ありません。詳細についてここでは言及しませんので、永続するローカル共有オブジェクトデータが保存される場所についての情報は、共有オブジェクトファイル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さらなる情報は、*Client-Side Communication ActionScript Dictionary*の `SharedObject.getLocal` 項を参照してください。

リモート共有オブジェクト クライアント・サイド ActionScript では、同じまたは異なるクライアント上で作動する他の Flash Communication Server アプリケーション・インスタンスに使用できる共有オブジェクトを作成、参照できます。ローカル共有オブジェクトのように、このオブジェクトはローカル・コンピュータで永続しますが、サーバー上でも永続し、それによって共有オブジェクトに接続したユーザーなら誰でも同じ情報にアクセスできます。

例えば、サーバー上で永続する電話リストのように、リモート共有オブジェクトを開くことができます。クライアントが共有オブジェクトを変化させるといつても、修正されたデータは、そのオブジェクトに今接続している全てのクライアントや後で接続するクライアントに使用可能になります。オブジェクトがローカルで永続し、クライアントが接続していないときにデータを変更させると、その変更は、クライアントがそのオブジェクトに接続する次のタイミングでリモート共有オブジェクトと同調します。

共有オブジェクトの全タイプの中でこのリモート共有オブジェクトが、Flash Communication Server アプリケーションにおいて一番使用頻度が高いと思われます。Flash Communication Server とともにインストールされるいくつかのチュートリアルでは、共有オブジェクトを使って、テキストを共有したり(`Shared Text FLA` ファイルを参照してください)、多くのクライアントにグラフィックなどの要素を同時にステージ上で操作させたり(`Shared Ball FLA` ファイルを参照してください)しています。永続するリモート共有オブジェクトのデータの保存場所に関する情報は、共有オブジェクト・ファイル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さらなる情報は、*Client-Side Communication ActionScript Dictionary*の `SharedObject.getRemote` 項を参照してください。

プロキシ共有オブジェクト

プロキシ共有オブジェクトは、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かわりに、2つの異なる Flash Communication Server アプリケーション間や同じアプリケーションのインスタンス間で共有される、リモート共有オブジェクトです。例えば、同じチャット・アプリケーションの2つのインスタンス、`/chat_01` と `/chat_01` を思い浮かべてください。サーバー上では、`chat_01` アプリケーション・インスタンスは `/chat_02` で定義された共有オブジェクトに接続でき、まるで `/chat_01` で定義されたかのようにその共有オブジェクト内の情報を使用できるのです。

プロキシ共有オブジェクトのさらなる情報は、*Server-Side Communication ActionScript Dictionary* の `SharedObject.get` 項を参照してください。

Application オブジェクト

Application オブジェクトによって、接続を試みるユーザーを許可したり拒絶したり、クラスやプロキシを登録したり登録からはずしたり、アプリケーションが開始、または停止したときや、クライアントが接続、または解除したときに動作する関数を作成したりできます。このセクションでは、コミュニケーション・コンポーネントを使用するアプリケーションでの、`application.OnConnect` や `application.OnDisconnect` 関数のセットアップ方法やセットアップする理由、`application.onConnectAccept` や `application.onConnectReject` 関数の使用のタイミングや使用方法を説明します。

Application オブジェクトに関するさらなる情報は、*Server-Side Communication ActionScript Dictionary* の `Application(object)` 項を参照してください。

Application.onConnect

`application.onConnect` イベント・ハンドラは、クライアントが `NetConnection.connect` メソッドを呼んだとき、サーバー上で呼び出されます。ハンドラは、`NetConnection.onConnect` メソッドに渡す開発者が定義したパラメータはもちろん、接続しようとしているクライアントの代わりとなる、サーバー・サイドの `Client` オブジェクトへの参照を自動的に渡されます。

`application.onConnect` でメソッドをセットアップすることは、ユーザーのログイン情報に基づいた異なるメソッドをセットアップしたいときには、よいアプローチになります。しかし、全クライアントに使用可能なメソッドをセットアップしたいなら、`Client` オブジェクトの `prototype` プロパティを使用すべきです。

//ユーザーの接続

```
application.OnConnect = function(newClient, userName){
    if(userName == "admin"){
        newClient.adminFunc = function(param){
            //管理者にのみ有用なコード
            newClient.myAdminProperty = param;
        }
    }else{
        //ほとんどの場合のコード
    }
    //ログオン許可
    application.acceptConnection(newClient);
}
//この部分は別ファイルも可能
//管理者を含むどのクライアントもこの関数を持つ
//"prototype"の機能による
Client.prototype.commonFunction = function(myColor){
    //何かのコード
    //newClient の代わるクライアントへの参照にこれを使用する
```

```
this.color = myColor;
```

```
}
```

また、接続中のクライアント・オブジェクト上で call メソッドを使用したい場合は、application.acceptConnection を呼び、追加のコマンドを出す前にクライアントが接続していることを確認します。

```
Application.onConnect = function(clientObj.name, passwd){
    //最初の接続許可
    application.acceptConnection(clientObj);
    //クライアントをアプリケーション・インスタンスに登録した後、
    //”call”コマンドを使用できます。
    ClientObj.call(“onWelcome, “You are now connected!!!”);
    Return;
    //onConnect 内で、一度 acceptConnection か rejectConnection を呼ぶと、
    //return の値は無視されます。
}
```

Tip: コンポーネントを使用している場合は、Application.onConnectAccept と application.onConnectReject を参照してください。

Application.onDisconnect

アプリケーションからクライアントが接続を解除すると、サーバーは application.onDisconnect メソッドを呼びます。このハンドラにコードを加えて、このイベントを他の全ユーザーに知らせることもできます。

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

Tip: コンポーネントを使用している場合は、Application.onConnectAccept と application.onConnectReject を参照してください。

Application.onConnectAccept と application.onConnectReject

アプリケーションにコンポーネントを使用するとき、コンポーネントが onConnectAccept と onConnectReject イベントを持ち込むことを知っていることは良いことです。これらのイベントを管理するにはコードが必要になります。コンポーネントを使用するときにはいつでも、サーバー・サイド・コードで、コンポーネントにユニークな、application.onConnectAccept と application.OnConnectReject を含む application.onConnect 文を修正する必要があります。実行順で onConnect メソッドの最後の行は、application.acceptConnection か application.rejectConnection であるべきです。もし、アプリケーションが、ユーザーのアプリケーションへの許可を認めた、または拒否したということを示すメッセージといったような、acceptConnection や rejectConnection の次に追加のコードが必要ならば、application.onConnectAccept や application.onConnectReject 内でコードを置き換えます。

Tip: コミュニケーション・コンポーネントを使用していない場合は、application.onConnectAccept や application.onConnectReject は使用できません。

それぞれのコンポーネントは、それ自身の application.onConnect メソッドを持つことができます。OnConnectAccept、onConnectReject を使用することにより、コンポーネントをカスタマイズしたり、オーセンティケーションを実行する新しいコンポーネントを作成することができます。例えば、コンポーネントのコードがデータベースに問い合わせ、データベースに蓄積されているユーザー名とパスワードに基づいて、ユーザーの接続を許可するか拒否するかを決めることができます。

application.onConnectAccept と application.onConnectReject に関する詳細は、*Server-Side Communication ActionScript Dictionary* を参照してください。

Camera オブジェクト

このセクションでは、Camera オブジェクトの使用を最適化する際に役立つ推奨事項を説明します。カメラ設定を使用可能な帯域幅に合わせたり、マルチプル・アプリケーションでのカメラ使用に関するチップスなどを含みます。

カメラを切る

アプリケーションが、データを記録する NetStream オブジェクトのアタッチされている Camera オブジェクトを使用するならば、カメラは記録を終了するまでそのままです。カメラを切るには、記録が終わった時点で NetStream.attachVideo(false)を使用します。

異なる帯域幅スピードへの設定

既定のカメラ設定は、全ての帯域幅設定で良く見えるように設定されています。しかし、異なる帯域幅で設定して試してみることもできます。

以下はカメラ設定のコードです。

帯域幅	効果	コード
Modem	イメージ品質を低く、モーション品質を高く イメージ品質を高く、モーション品質を低く	my_cam.setQuality(4000,0) my_cam.setQuality(0,65)
DSL	イメージ品質を低く、モーション品質を高く イメージ品質を高く、モーション品質を低く	my_cam.setQuality(12000,0) my_cam.setQuality(0,90)
LAN	イメージ品質を低く、モーション品質を高く イメージ品質を高く、モーション品質を低く	my_cam.setQuality(400000,0) my_cam.setQuality(0,100)

マルチプル・アプリケーションで1つのカメラを使う

マルチプル・アプリケーション(SWF)では、同じプロセスで動作している場合なら、同時に同じカメラを使用できます。一般的に、多数のブラウザ・ウィンドウは全て同じプロセスで動作していますので、ブラウザ環境ではうまく動作します。

しかし2つの異なるプロセス、例えば1つはブラウザでもう1つはスタンド・アロン・プレイヤーで動作するアプリケーションの間でカメラを共有することはできません。

Client オブジェクト

サーバー・サイド・スクリプトで Client オブジェクトにメソッドをアタッチする際、そのクライアント・オブジェクト上の全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にコール可能だということを忘れないください。リモート・コンピュータで呼び出したいクライアント上のメソッドは含むべきではありません。例えば、クライアントがアプリケーションを切断できるメソッドを呼ぶことは望まれません。

Microphone オブジェクト

このセクションでは、Microphone オブジェクトの使用を最適化する際に役立つ推奨事項を説明します。ハウリングを避けるチップスも紹介します。

ハウリングを避ける

マイクを、無理のない高音量設定の内蔵スピーカーと併用したとき、ハウリングの問題が起きたりします。スピーカーからのハウリングを減らすには、Flash Communication Server はエコー抑制を実行します。エコー抑制を使用するには、以下のコマンドを使います。

```
MyMicrophone.useEchoSuppression(true);
```

これでスピーカーから多くのエコーを出すことなく入力レベルを最適に保てます。

エコー抑制はユーザーの指定した出力信号の一部のみ削除するので、マイクがまだスピーカーに近すぎる場合は、ハウリングが残るかも知れません。ハウリングを避けるには、以下のガイドラインを試してください。

- スピーカー音量を下げる
- マイクをスピーカーから遠ざける
- ハードウェアを適切にインストールし設定しなおす。
- ヘッドセットを使う

マイクを開けたままにする

帯域幅を保つため、Flash Communication Server は、使用されていないとき、規定でマイクを切ります。しかし、例えばマイクが稼働しているときは絶対に遅らせたくないなど、アプリケーション上でマイクを開けたままにしたい場合があります。そのときは、`my_mic.setSilenceLevel(0)`コマンドを使用します。

NetConnection オブジェクト(クライアント・サイド)

ムービーを含む HTML ページが、ムービー自体が Flash Communication Server にアクセスする経路(ドメイン名に関して)とは違う経路から、アクセスされると、接続は成功しません。これは Flash Player のセキュリティの特徴です。しかしこれが不便になる場合もあります。

例えば、ムービーを含む web ページが、<http://deptserver.mycorp.com> というイントラネット上にあると、<http://deptserver> で簡単にアクセスできます。もしページが<http://deptserver/tcpage.htm> 経由でアクセスされると、ムービーは *targetURI* として `deptServer.mycorp.com` を指定するので、ムービーはサーバーへの接続を成功させることができません。同様に、web ページとムービーが<http://deptserver.mycorp.com/tcpage.htm> としてアクセスされると、ムービーは *targetURI* として `rtmp://deptserver` を指定し、接続はできません。

この少々不便なセキュリティ・ポリシーを避けるためにできることは、ひとつめは、最も簡単なことで、*targetURI* にサーバー名を含まないことです(これは Flash Communication Server と web サーバーが同一のマシンで動作している場合のみ適用できます)。そのためには、*targetURI* のふたつめのスラッシュを取ります。例えば以下のコマンドを使用すれば、Flash Player は SWF ファイルのある web サーバーと同じホストとドメインに接続しようとしています。

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

ふたつめは、セキュリティ問題を避けるために HTML ページ内に JavaScript を加えます。ムービーは Flash Communication Server にアクセスするために、フル・ドメイン名 URL を使用すると仮定して、この JavaScript は、フル URL に web ページをリダイレクトします。

```
<SCRIPT language = "javascript">
if(document.URL.indexOf("mycorp.com") == 01){
    document.URL = http://deptServer.mycorp.com/tcpage.htm;
}
</script>
```

最後は、ドメイン名を所有しているなら、ドメイン名の DNS レコードにアクセスして、Flash Communication Server 用の静的 IP アドレスを取得し、アドレス("A")レコードを作製してその IP アドレスにホスト名を当てます。例えば、`flashcom.mycorp.com` は Flash Communication Server が稼動し、IT 部門か ISP の与えた IP アドレスを持っているマシンをマップできます。Web ページは、今使用しているどんな方法によってでもホストされつづけます(ホスト名にアクセスできる DNS を持つサーバーにトラフィックを送る必要があるが、静的 IP アドレスを持つか持たない場合は、"CNAME" レコードが推奨されます)。

NetStream オブジェクト

このセクションでは、NetStream オブジェクトの使用を最適化する際に役立つ推奨事項を説明します。ストリームにデータを組み入れたり、ストリームのバッファを管理するチップスも紹介します。

ストリーム内のマルチプル・データ・タイプ

ストリーミングのオーディオやビデオに加えて、ストリームの中にテキスト・メッセージといったデータも含ませることができます。そうするには、`NetStream.send` コマンドを使用します。

ActionScript 内でストリーム時間の長さを取得する

ストリームをバッファリングしている場合、NetStream.bufferLength プロパティを使って、バッファ内の今の秒数を取得できます。しかし時には、ストリームの合計した長さを取得したいかも知れません。その場合には、Flash Player はストリームの長さを知ることはありませんが、サーバーは知っています。サーバーは、サーバーへのメッセージを通してクライアントが要求できる Stream.length プロパティを持っています。

以下のコードでクライアント・サイドの ActionScript を書いて、ストリームの長さを要求できます。

```
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);
    };
}
```

その後、対応するサーバー・サイド ActionScript で、main.asc ファイルに以下を書きます。

```
application.onAppStart = function(){
    Trace("::: Application has started :::");
}
application.onConnect = function(client){
    application.acceptConnection(client);
    Client.prototype.sendInfo = function(name){
        Var slen = Stream.length(name);
        Trace("slen " + slen);
        Return slen;
    };
}
```

NetStream.bufferTime

ストリームの再生には、Flash Player が再生を開始する前に、Flash Communication Server がどれだけのストリームをバッファするか、NetStream.bufferTime が返す値で指定します。例えば、記録したストリームは 50 秒で、再生スピードはダウンロード・スピードの 2 倍であるなら、この値を 25 秒に設定するよう考えます。こうすると失速することなく、全ストリームを再生することができます。

ストリームをパブリッシュするには、Flash Player がメッセージを落とし始める前に、外部への待ち分をどれだけ生じさせるかこの値で指定します。どれだけのデータが今待ち分にあるか明らかにするには、NetStream.bufferLength を使い

ます。速い接続では、`NetStream.bufferLength` が返す値が、`NetStream.bufferTime` の値に近くなることはありませんが、遅い接続ではあり得ます。

そこで、アプリケーションが遅い接続上で動作することが分かっている、メッセージの欠落を最低限におさえ、再生の失速をおさえたいなら、`NetStream.setBufferTime` を使用して、`NetStream.bufferTime` の値を大きくします。

SharedObject オブジェクト

共有オブジェクトは広い用途で、多くの方法に合わせて使用できます。このセクションでは、アプリケーションで共有オブジェクトを使い始めるとき考慮すべきいくつかの事項を説明します。

SharedObject.onSync

リモート共有オブジェクトで作業を始める前には、最初に、接続に成功したことを示す、`SharedObject.connect` が返す `true` をチェックします。それから `SharedObject.onSync` に割り当てた関数から結果を受けとるまで待ちます。そうしないと、`SharedObject.onSync` が呼び出される前、ローカルでなされた変化は全て失われます。

`SharedObject.onSync` ハンドラが呼び出されるのは、リモート共有オブジェクトの `SharedObject.data` プロパティが変化したり、クライアントが初めて `SharedObject.getRemote` コマンドを使ってローカル、またはサーバーに存続するリモート共有オブジェクトに接続したときです。後者の場合には、オブジェクトの全てのプロパティは、空のストリングに設定され、`SharedObject.onSync` の `code` の値は“clear”になります。その後 `SharedObject.onSync` は再び呼び出され、このとき `code` の値は“change”に設定され、クライアントの持つ、リモート共有オブジェクトのインスタンスのプロパティはサーバーのそれらに合うよう設定されます。

共有オブジェクトのふるまいに関して理解が難しい場合は、`SharedObject.onSync` ハンドラでデバッグ・コードを置いてみると理解の手助けになります。

```
so.onSync = function(list){
    for(var k in list){
        trace("name = " + list[k].name + ", event = " + list[k].code);
    }
}
```

共有オブジェクトのスロットを効果的に使用する

1つのスロット(`SharedObject.data` プロパティ)に共有オブジェクトのデータをまとめるか、多数のスロットに広げたりするかは、アプリケーションがオブジェクトにどのように変化を与えるかによる部分もあります。例えば、アプリケーションが、ある特定の区間で接続している全ユーザーにストリングの配列を送る必要がある場合は、ひとつのスロットに全ての配列をおさめることができます。

他方、アプリケーションが、変更した情報だけを送る必要がある場合は、多数のスロット間でデータを分けるべきです。そうすることで、ネットワーク・トラフィックを減らすことができ、アプリケーションのパフォーマンスを上げることが可能です。また、多数のスロットはデータの衝突なしに同時に更新されるので、衝突を解決するコードも最低限で済ませられます。

リモート共有オブジェクトの書き込み

Flash Communication Server は、全ユーザーが共有オブジェクトから接続を解除したりサーバーがシャットダウンすると、自動的に、クライアントとサーバー両方のハードディスクに、リモート共有オブジェクトを書き込みます。それ以外の時

で、ハードディスク上の共有オブジェクトを更新したい時は、クライアント・サイドの `SharedObject.flush` メソッドを呼びます。

しかし、クライアント・サイドの `ActionScript` で `SharedObject.flush` を呼ぶと、共有オブジェクトのローカル側のみに書き込みます。共有オブジェクトのサーバー側に手動で書き込むには、サーバー・サイド・スクリプトで `SharedObject.flush` を呼ぶ必要があります。

```
application.onAppStart = function(){
    application.mySO = SharedObject.get("SharedObjName", true);
}
application.onDisconnect = function(client){
    application.mySO.flush();
}
```

共有オブジェクトの同調問題の回避

1人以上のクライアント（またはサーバー・アプリケーション）が同時に共有オブジェクトの1つのスロットでデータを変更すると、衝突解決の策をうつ必要があります。以下は例です。

異なるスロットを使う 最も単純な打開策は、オブジェクトのデータを変更するかも知れない各ユーザーに、異なるスロットを使用することです。例えば、チャット・ルームでユーザーの追跡をつづけている共有オブジェクトでは、1人の1つのスロットを与え、ユーザーには自分のスロットのみ変更させます。

所有者をあてがう もっと複雑な打開策は、ある制限時間の中において、共有オブジェクトで1人のクライアントを所有者プロパティを定義することです。クライアントがスロットの所有権を求める、“lock”オブジェクトを生成するサーバー・コードを書く必要があります。サーバーが要求の成功メッセージを出したら、クライアントは、共有オブジェクトの中でデータを変更したのは1人だけだったと知ります。

以下は共有オブジェクトをロックし、アンロックするサーバー・サイドの ActionScript で、ゲームにハイスコアを戻す true を確認します。現在のハイスコアが 95 のとき、プレイヤー 1 が 105 を出し、プレイヤー 2 が 110 を出したとします。もしロックされないと、両方のプレイヤーのスコアが現在のハイスコア 95 と比較され、両方のクライアントが同時に updateHighScore をコールすると衝突が生じます。ハイスコアが最新のハイスコアなのか、それともたった今他のユーザーが出したハイスコアなのかに関わらず、各ユーザーのスコアをハイスコアと比較して確認したいでしょう。共有オブジェクトをロック、アンロックしハイスコアの保持に使用すれば、連続して出るスコアそれぞれを比較し、比較されるものが失われないようにできます。

```
application.onAppStart = function(){
    application.scoreSO = SharedObject.get("high_score_so", true);
    application.scoreSO.onSync = function(listVal){
        trace(" got an onSync scoreSO");
    }
}
application.onConnect = function(newClient, name,paswd){
    newClient.updateHighScore = function(final_score){
        application.scoreSO.lock();
        if(application.scoreSO.getProperty("high_score_so") < final_score){
            application.scoreSO.setProperty("high_score_so",final_score);
        }
        application.scoreSO.unlock();
    }
}
```

クライアントに知らせる クライアントが要求した共有オブジェクトの変更をサーバーが拒否すると、SharedObject.onSync イベント・ハンドラが、変更は拒否されたことをクライアントに知らせます。このように、アプリケーションのユーザー・インターフェイスを使って、ユーザーに衝突を解決させることができます。このテクニックは、共有アドレス・ブックのような、データが減多に変更されない場合に最も有効に働きます。同調による衝突が生じると、ユーザーがその変更を認めるか拒否するかを決めます。

いくつかの変更は認め他は拒否する アプリケーションの中には、“早く来た、早く出された”方を優先する方針に基づいて変更を認めるものもあります。これは、他の人の変化が先に起こった場合、変更を再適用することで、ユーザーが衝突を解決するとき有効に機能します。

send メソッドを使ってオブジェクト変更に関する管理レベルを上げる SharedObject.onSync ばかりで同調の動きを管理するのではなく、SharedObject.send コマンドを使用します。SharedObject.send コマンドは、メッセージを送ったクライアントを含む、リモート共有オブジェクトに接続する全クライアントにメッセージを発信します。

Stream オブジェクト

記録されたストリームに関連する FLV や IDX ファイルを削除するには、サーバー・サイド・コードを使用しなければなりません。

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

System オブジェクト

Macromedia Flash Communication Server MX は UTF-8 フォーマットでテキストを送ります。ActionScript コードで `System.useCodepage = true` を設定している場合、コミュニケーション・アプリケーションは正しく動かないかも知れません。true 値は、Flash Player がエンド・ユーザーのコードページに基づく全てのテキストを Flash Player の前のバージョンのものと見なす原因となります。それでも Flash Player は Flash Communication Server に UTF-8 でエンコードされたテキストを送り、サーバーから受ける全てのテキストは UTF-8 にエンコードされます。Flash Communication Server で `System.useCodepage = true` を設定すると、予期しない振る舞いを起こす結果となり、妨害となります。例えば、テキストはサーバーから適切に送られたり受け取られなかったりかも知れません。System.useCodepage に関するさらなる情報は、Macromedia Flash Support Center(www.macromedia.com/support/flash/languages/unicode_in_fmxml)の Macromedia MX の記事 Unicode を参照してください。

Video オブジェクト

このセクションでは、Video オブジェクトの使用を最適化する際に役立つ推奨事項を説明します。ビデオオブジェクトを動的に作成するチップスも含まれます。

Video オブジェクトを動的に作成する

Flash オーサリング環境でのみ、ステージにライブラリから埋め込みビデオオブジェクトをドラッグすることによって、アプリケーションにビデオオブジェクトを追加できます。しかし、ActionScript コード内からビデオオブジェクトを実行させたい場合は、ムービークリップ内に埋め込むことで実現できます。このテクニックにより、`duplicateMovieClip()` と `removeMovieClip()` を使用してビデオオブジェクトを動的に作成し削除できます。

フレーム・レートを理解する

静的 SWF ファイル内のムービーに FLV ファイルを埋め込む場合、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。FLV にデータをストリームさせる場合は、Flash Communication Server を通して再生されるビデオは、それを含む SWF とはことなるフレームレートを持つことができます。

例えば、アプリケーションを使って 15 fps でブロードバンド FLV ファイルを出力すると仮定します。この FLV ファイルを 6fps の Flash MX ムービーに読み込むと、ビデオは、そのフレームレートの違いにより同調しません。しかし Flash Communication Server を使って FLV ファイルを同じ 6fps ムービーの中へストリームさせると、15fps に同調します。ストリーミング・ファイルはフレームに基づく再生をしないからです。

CHAPTER 4

アプリケーションのデバッグとモニタリング

アプリケーション開発者として最も重要な作業の1つは、使用されている間アプリケーションを監視しデバッグすることです。このチャプターでは、アプリケーションの監視とデバッグの助けとなるいくつかのツールを説明します。

- **Communication App インスペクタ**は、ログ・メッセージ、ストリーム・データ、アプリケーションの統計など、今動いているアプリケーション・インスタンスについての重要な情報を表示します。また個々のアプリケーション・インスタンスをアンロードしたり、リロードする際にも使います。詳細は **Communication App インスペクタ** を使うを参照してください。
- **NetConnection Debugger** は、NetConnection オブジェクトに関連するデバッグ中のイベントを監視します。詳細は **NetConnection Debugger** を使用するを参照してください。
- **onStatus イベント・ハンドラ**は、クライアント・サイドとサーバー・サイドで呼ぶことができ、アプリケーションの動作に関する重要なデータを提供します。詳細は **onStatus イベント・ハンドラ** を使用するを参照してください。

これらのツールに加え、Administration Console では、サーバーの反応に関する様々な情報を見ることができます。またサーバーや仮想ホスト、アプリケーションを開始したり停止したりできます。詳細は *Managing Flash Communication Server* を参照してください。

Communication App インスペクタを使う

Communication App インスペクタは、サーバーで今動いているアプリケーション・インスタンスのリストを表示します。選択したどのインスタンスでも、以下のような状態に関する詳細を見ることができます。

- サーバー上のアプリケーション・インスタンスによって生成された、動作しているログ・メッセージのリスト。
- インスタンスに関連するストリームや共有オブジェクトの情報
- 合計の稼働時間、ユーザー数など、選択したアプリケーション・インスタンスの全体的な情報。

また Communication App インスペクタを使って、アプリケーション・インスタンスのロードとアンロードが行えます。

Communication App インスペクタを開くには、Macromedia Flash MX から、ウィンドウ> Communication App Inspector を選択します。Flash MX がインストールされていないサーバー・マシン上では、Flash Communication Server がインストールされているディレクトリの、/flashcom_help/html/admin サブディレクトリにある app_inspector.html を開きます。

Communication App インスペクタを使用するには、完全な管理者権限か vhost 管理者権限が必要です。

サーバーに Communication App インспекタを接続する

Communication App インспекタを開き、ログオン画面を使って管理者として接続し、管理したいアプリケーションが動作するサーバーに接続します。



右上のライト・アイコンは、App インспекタがサーバーに接続しているかどうかを示します。App インспекタを初めて起動すると、ライトは、接続していないことを示す黄色になります。

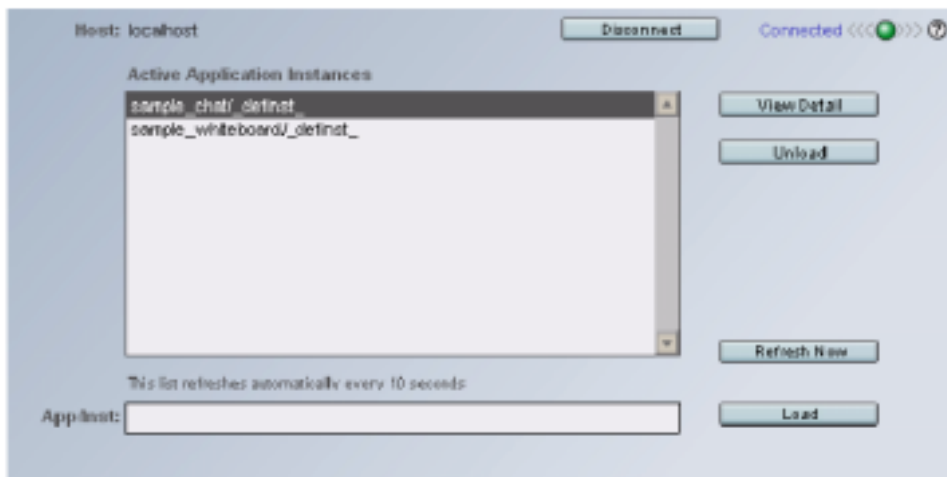
サーバーに接続するには：

1. Host 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 App インспекタが同じコンピュータで作動している場合は `localhost` と入力します。サーバーが既定の 1111 以外のポートでインストールされている場合は、例えば、`localhost:1234` のように、ポート・ナンバーを入力します。
2. 管理者名とパスワードを入力します。
3. ユーザー名とパスワードをセッション中保存しておきたい場合は、Remember Connection Data を選択します。また App インспекタを開くと自動的に接続させたい場合は、Automatically Connect を選択します。
4. Connect をクリックします。

接続が成功すると、ライト・アイコンが緑に変わり、“Not Connected”が“Connected”に変化します。

Application Instance パネル

サーバーに接続すると、App インスペクタには今動作しているアプリケーション・インスタンスのリストのパネルが現れます。このリストは 10 秒ごとに自動的に更新されます。すぐ更新させたいときは、Refresh Now をクリックします。



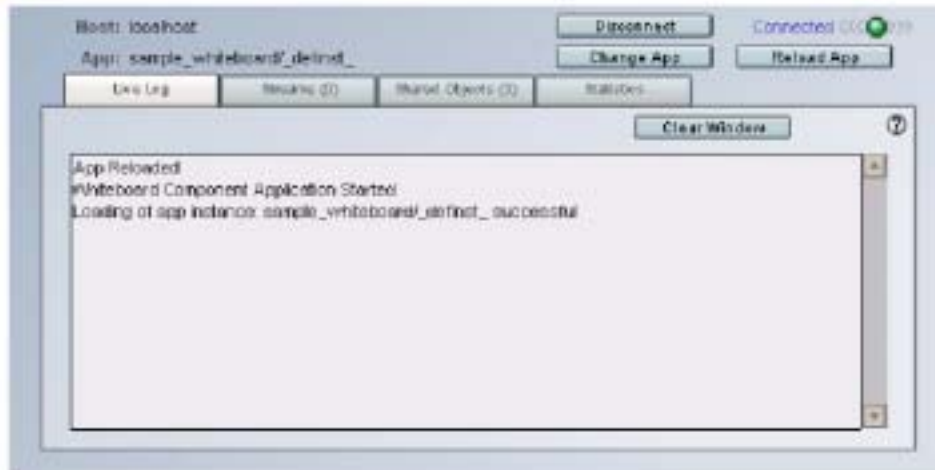
App インスペクタにアプリケーション・インスタンスのリストが表示されない場合は、サーバー・サイド ActionScript のエラーで、アプリケーションがロードされていないことがその原因に考えられます。アプリケーションがロードされていないと、パネルの下部右にアラート・アイコンが現れます。サーバー・サイド・ログ・メッセージを含むパネルを表示するには、そのアイコンをクリックします。このメッセージはアプリケーションがロードされない原因を探る助けとなります。パネルを閉じると、Application Instance パネルに戻ります。

この Application Instance パネルでは、以下の作業ができます。

- 動作しているアプリケーション・インスタンスに関する情報を見るには、そのインスタンスを選択し View Detail をクリックします。Live Log パネルが表示されます(Live Log パネル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し、ユーザーの接続を解除し、アプリケーション・インスタンスが使用しているリソースを解放するには、そのアプリケーション・インスタンスを選択し Unload をクリックします。
- 手動でアプリケーション・インスタンスをロードするには、パネル下部のテキスト・ボックスにその名前を入力し、Load をクリックします。アプリケーションはそれまでにサーバー上で構成されていなければなりません。
- サーバーから App インスペクタを切断するには、Disconnect をクリックします。ログオン画面に戻ります(サーバーに Communication App インスペクタを接続するを参照してください)。

Live Log パネル

Live Log パネルは、サーバー上で選択されたアプリケーション・インスタンスにより生成されたログ・メッセージや App インспекタにサーバーから送られたログ・メッセージを表示します。パネル内の情報は、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と必ず更新されます。選択して他のアプリケーションへコピーできます。

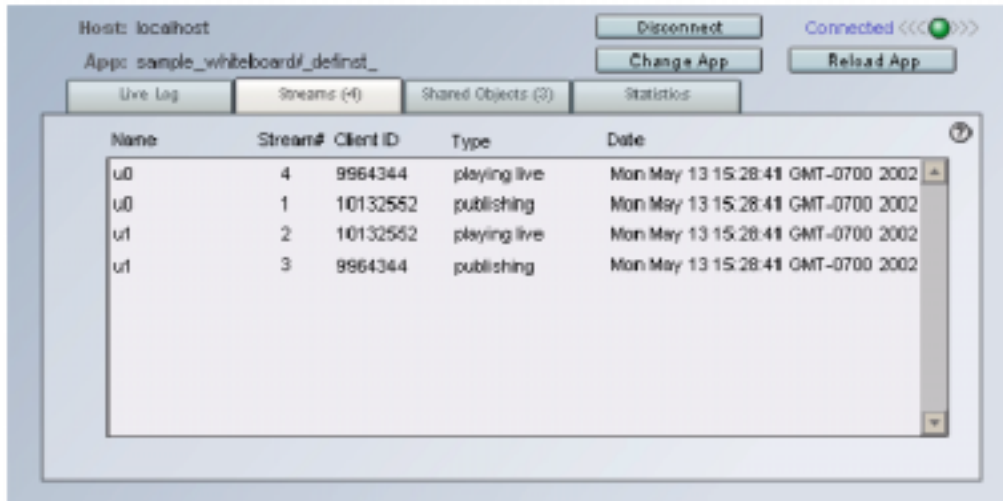


このパネルでは以下の作業を行います。

- ログ内容をクリアするには、Clear Window をクリックします。
- 今動作しているアプリケーション・インスタンスのリストに戻るには(そして他のアプリケーション・インスタンスの情報を見るには)、Change App をクリックします。そうするとアプリケーション・インスタンス・パネルに戻ります(Application Instance パネルを参照してください)。
- アプリケーション・インスタンスをリロードするには、--例えば、アプリケーションが使うサーバー・サイド・スクリプトのひとつを変更したとか、全ユーザーの接続を解除したい場合など--Reload App をクリックします。
- サーバーから App インспекタの接続を解除するには、Disconnect をクリックします。ログオン画面に戻ります(サーバーに Communication App インспекタを接続させるを参照してください)。
- 別のパネルを見るには、対応するパネル名(Stream、Shared Objects、Statics)をクリックします。

Stream パネル

Stream パネルは、選択したアプリケーション・インスタンスが関連するストリームについての情報を表示します。情報は 15 秒ごとに自動的に更新されます。すぐに更新するには Stream パネルの名前をクリックします。



このパネルは以下の情報を表示します。

Name は、NetStream.publish か NetStream.play コマンドで指定されたストリーム名を示します。

Stream#は、ストリーム ID 番号を示します。これはストリームがロードされた順番を表す、ただの数字です。

Client ID は、クライアントの内蔵 ID を示します。これは Flash Communication Server が各クライアントの確認に使用する内蔵番号を表します。

Type は、このストリームに生じていることを記すサーバーが提供するストリングを示します。

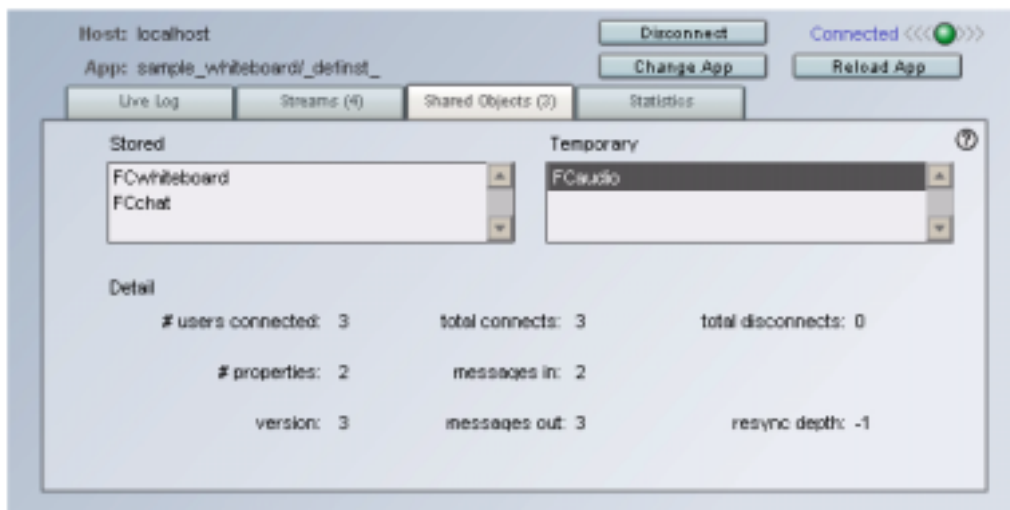
Date は、そのストリームがパブリッシュや再生を開始した日時を表します。

Stream パネルでは以下の作業を行います。

- 今動作しているアプリケーション・インスタンスのリストに戻るには(そして他のアプリケーション・インスタンスの情報を見るには)、Change App をクリックします。そうするとアプリケーション・インスタンス・パネルに戻ります(Application Instance パネルを参照してください)。
- アプリケーション・インスタンスをリロードするには、Reload App をクリックします。
- サーバーから App インспекタの接続を解除するには、Disconnect をクリックします。ログオン画面に戻ります(サーバーに Communication App インспекタを接続させるを参照してください)。
- 別のパネルを見るには、対応するパネル名(Live Log、Shared Objects、Statics)をクリックします。

Shared Object パネル

このパネルは、アプリケーション・インスタンスが使用する共有オブジェクトの情報を示します。情報は 15 秒ごとに自動的に更新されます。すぐに更新するには Shared Object パネルの名前をクリックします。



共有オブジェクトはリモートで永続したり(サーバー上に保持されたり、アプリケーション・インスタンスやサーバーが停止したりリスタートした後でも使用できます)、インスタンスの存在中にのみ使用できたり(一時的なもので、インスタンスが作動をやめると消去されます)します。このパネルは、アプリケーション・インスタンスが使用する両方のタイプの共有オブジェクトの情報を表示します。

特定の共有オブジェクトの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。その後パネルは以下の情報を表示します。

#Users Connected は、今接続し、この共有オブジェクトを使用しているユーザー数を示します。

#Properties は、共有オブジェクトにあてがわれた data プロパティの数を示します。

Version は、共有オブジェクトのバージョン数を示します。共有オブジェクトのプロパティが変化すると、バージョンは 1 増えます。

Total Connects と **Total Disconnects** は、共有オブジェクトが生成されてから、オブジェクトに接続、または接続を解除した合計数を示します。

Messages In と **Messages Out** は、共有オブジェクトが送ったり受けたりしたメッセージ数を示します。“messages in”は、クライアントからサーバーに送られた更新要求を表します。“messages out”は、サーバーから共有オブジェクトに接続しているクライアントに送られた、成功した更新の通知を表します。

Resync Depth は、Flash Communication Server が使用し、共有オブジェクトのスロットを永久に消去するか、共有オブジェクトのクライアント・バージョンをクリアし、また場所を作るかを決定します。詳細は、*Server-Side Communication ActionScript Dictionary* の SharedObject.resyncDepth 項を参照してください。

このパネルでは以下の作業を行います。

- 今動作しているアプリケーション・インスタンスのリストに戻るには(そして他のアプリケーション・インスタンスの情報を見るには)、Change App をクリックします。そうするとアプリケーション・インスタンス・パネルに戻ります(Application Instance パネルを参照してください)。
- アプリケーション・インスタンスをリロードするには、Reload App をクリックします。
- サーバーから App インспекタの接続を解除するには、Disconnect をクリックします。ログオン画面に戻ります(サーバーに Communication App インспекタを接続させるを参照してください)。
- 別のパネルを見るには、対応するパネル名(Live Log、Streams、Statics)をクリックします。

Statistics パネル

このパネルは、アプリケーション・インスタンスの全般的な状況に関する情報を示します。情報は 15 秒ごとに自動的に更新されます。すぐに更新するには Statistics パネルの名前をクリックします。



このパネルは以下の情報を表示します。

#Active は、今アプリケーション・インスタンスに接続しているユーザー数を示します。

Accepted(Total)は、アプリケーション・インスタンスが開始してからアプリケーション・インスタンスに接続したユーザー数を示します。

Rejected(Total)は、アプリケーションが開始してからアプリケーション・インスタンスへの接続を拒否されたユーザー数を示します。接続が失敗した理由を探るには、App インспекタの Live Log パネルや(Live Log パネルを参照してください)、Administration Console(*Managing Flash Communication Server*)のアクセス・ログを調べます。

Uptime は、アプリケーション・インスタンスが動作している時間の長さを示します。

Launch Time は、アプリケーション・インスタンスが動作を開始した日時を示します。

Bytes per Sec(In/Out)は、サーバーに送ったりサーバーから受けたりした、毎秒の平均バイト数を示します。App インспекタはこの 15 秒間に受信したバイト数の合計を 15 で割ってこの比率を計算します。パネルを初めて表示したときは、これらの数字は"pending"になっています。これは開始時の数字ひとつしかないからです。パネルを開けて 15 秒たつと数字が表示されます。

Messages per Sec(In/Out)は、サーバーに送ったりサーバーから受けたりした、毎秒の平均メッセージ(カメラやオーディオ・パケットからのビデオ・フレーム、コマンド・メッセージなど)数を示します。

Messages Dropped は、アプリケーション・インスタンスが開始してから落ちたメッセージ数を示します。サーバーが送信しているデータを受け取るとき、クライアントがずっと遅れてしまっていることによります。ライブ・ストリームやオーディオ、ビデオ・メッセージは落ちるかも知れませんが、記録されたストリームでは、ビデオ・メッセージのみ落ちます。コマンド・メッセージが落ちることはありません。

NetConnection Debugger を使う

NetConnection Debugger は、Flash MX 開発者に、Flash クライアントや Flash Communication Server、Flash Remoting と、サポートされるアプリケーション・サーバー間で、過程やコミュニケーションの問題を報告し診断するツールを提供します。

NetConnection Debugger を使用して Flash Communication Server のイベントをトレースするには：

1. ムービーの最初のキーフレームのレイヤーに、`#include "NetDebug.as"`ステートメントを追加する。

Note: ムービーで不必要なコードを避け、リモート・デバッグできるようにするには、デバッグが終わりアプリケーションを配置するときにこの行を削除します。

2. Flash MX から、ウィンドウ>NetConnection Debugger を選択して、デバッガを開きます。
3. フィルタをクリックして、フィルタ UI(このドキュメントの後半で詳細を説明します)を開きます。
4. アプリケーションが接続しているサーバーの、管理者のユーザー名とパスワードを入力します。また、サーバーが既定の 1111 以外のポートでインストールされている場合には、ポート番号も入力します。

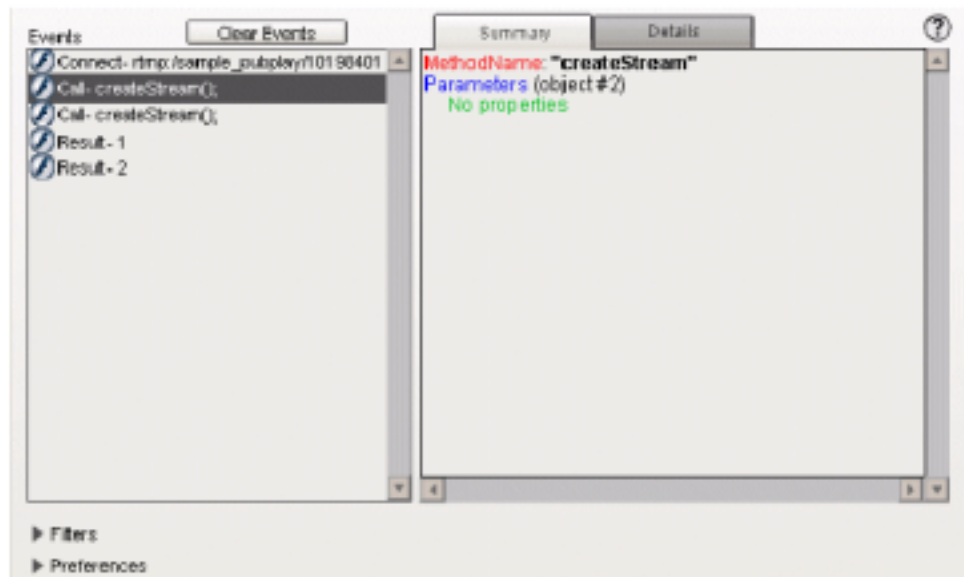
Note: NetConnection Debugger を、Flash remoting などの Flash Communication Server でないアプリケーションをトラックする場合、このステップは不要です。

5. (オプション)フィルタ UI を閉じて、他のイベントを表示させる場所を空けます。
6. Flash オブジェクト環境で制御>ムービープレビューを選択するか、ブラウザで SWF ファイルを開いて、ムービーをテストします。

NetConnection Debugger を閉じてまた開く場合は、ステップ 3 とステップ 4 を繰り返します。

NetConnection Debugger ユーザー・インターフェイス

NetConnection Debugger ユーザー・インターフェイスは、イベント表示を含みます。デバッガをカスタマイズするには、フィルタ・メニューと環境設定メニューを使用します。以下は、デバッガ・ユーザー・インターフェイスです。



このウィンドウは以下の情報を表示します。

イベントは、個別のデバッグ・イベントのリストを表示します。各デバッグ・イベントは、デバッグ・イベント・ソースや、デバッグ・イベントのタイプ、イベントの一覧説明を表すアイコンを含みます。

一覧パネルは、選択したデバッグ・イベントのデバッグ中のイベントについての簡単な説明を表示します。

詳細パネルは、選択したデバッグ・イベントの詳しいデバッグ中の情報を表示します。色と句読点は表示されるデバッグ中の情報のタイプを区別するために使用されています。

以下の色と情報がサポートされます。

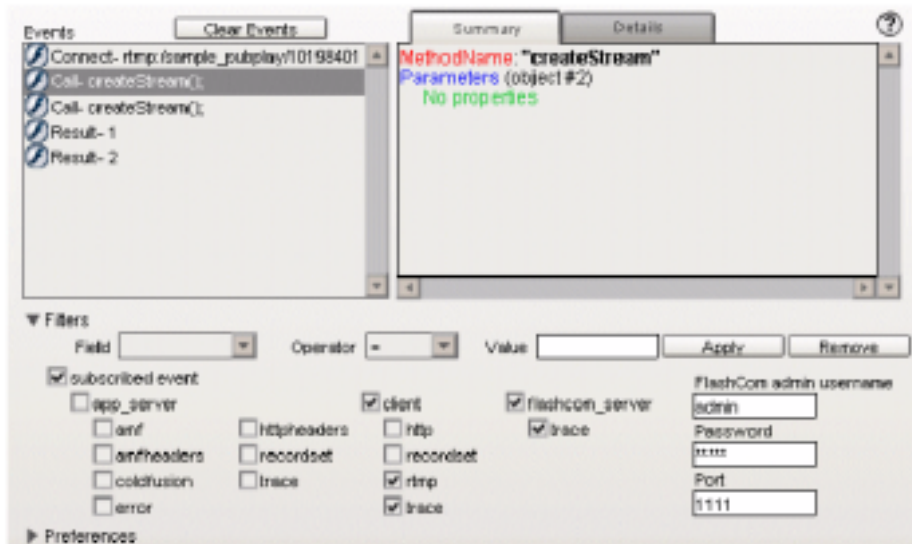
- 赤：プロパティ名
- 青：プロパティ・オブジェクト
- 黒：プロパティ値
- 緑：デバッガ・エラー・メッセージ

プロパティ値は、句読点を使ってデータ・タイプを伝えます。以下のデータ・タイプがサポートされます。

- スtring：ダブル・クォーテーション・マーク("string")
- 数値：特に形式はなし
- 配列番号：配列インデックスのブラケット(array[1])
- 他のタイプ：true(boolean)のように、カッコ内に入っているタイプ名で表されるString。

フィルタ・メニュー

フィルタ・メニューにより、デバッグ・イベント・フィールドでデバッグ中のイベントを知り、監視するイベント・タイプを決めることができます。以下はフィルタ・メニューです。



フィールド、演算子、値は、連携して動作し、イベント・リストに含まれる基準を指定します(実行するには選択して、適用をクリックする必要があります)。

フィールドでは、フィルタをかけるフィールド名を選択するか入力します。

演算子では、イベント・リストに表示されるイベントを満たす比較基準を選択できます。contains 演算子はストリングでのみ機能し、イベント・フィールドでは、サブストリングとして値を比較するよう指定します。

値では、選択した演算子を使って、比較する値を入力します。フィールド、演算子、値ボックスで指定した基準を満たすイベントのみがイベント・リストに表示されます。

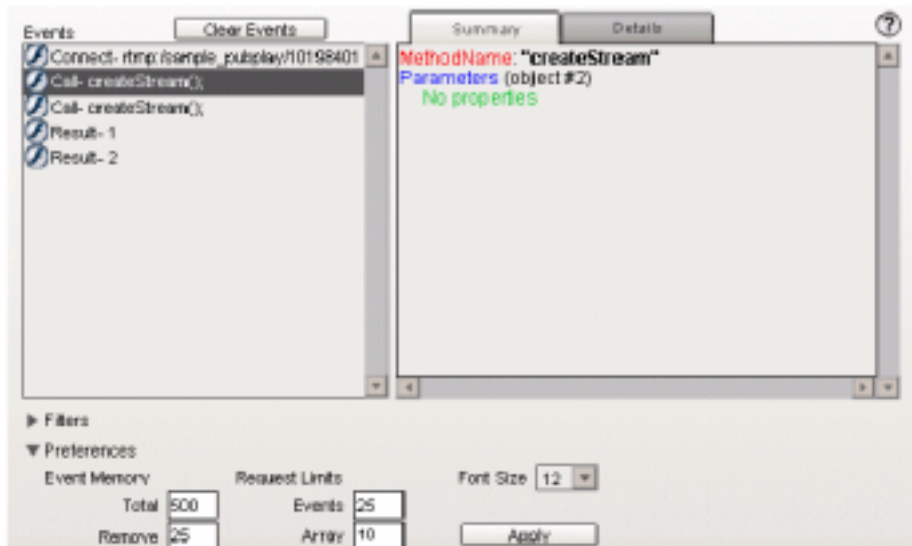
引用するイベントでは、イベント・タイプの隣にあるチェックボックスを選択、クリアすることで、デバッガにどのデバッグ・イベントが報告されるかを指定します。適用をクリックすると、新しいデバッグ・イベントとしてその変更が効果を表します。以下のチェックボックスは、Flash Communication Server に特に関係するものです。

- 引用するイベント(これが選択されていないと、デバッグ・イベントがデバッガに記録されません)
- client(これが選択されていないと、クライアント・イベントがデバッガに記録されません)
- rtmp(サーバーへの接続に使用されるプロトコルです)
- flashcom_server(これが選択されていないと、サーバー・イベントがデバッガに記録されません)
- trace(これが選択されていないと、トレースがデバッガに記録されません)

ユーザー名、パスワード、ポートについては、NetConnection Debugger を使用するを参照してください。

環境設定メニュー

環境設定メニューでは、表示オプションや NetDebug のパフォーマンス制限を設定します。以下は環境設定メニューです。



環境設定メニューの主な要素は、イベントメモリ、リクエストの制限、フォントサイズです(選択を実行するには適用をクリックする必要があります)。

イベントメモリは、イベント・リストでデバッガが保持するイベント数の合計を管理します。新しくイベントの発生したとき蓄えたイベントを廃棄するには、デバッガの増加率を調整します。合計では、メモリに保持するイベントの最大数を指定します。削除では、メモリ内のイベントの合計数が超過したとき、削除するイベント数を指定します。デバッガのパフォーマンス問題に当たったら、この設定を調節します。

リクエストの制限は、トランザクション当たりのアプリケーション・サーバーから表示されるイベントの最大数と、サーバーからのネストされた配列当たりの配列要素の最大数を管理します。さらにデバッグ・イベントを見たり、デバッガのパフォーマンス問題に当たったら、この設定を調節します。

フォントサイズは、一覧と詳細パネルに表示されるテキスト・サイズを管理します。

onStatus イベントハンドラを使う

クライアント・サイドの Camera、Microphone、NetConnection、NetStream、SharedObject オブジェクトは、サーバー・サイドの Application、NetConnection、Stream、SharedObject オブジェクトと同様に、情報オブジェクトを使用して、情報や状況、エラー・メッセージを供給する onStatus イベントハンドラを提供します。このイベントハンドラに反応するには、関数を作って情報オブジェクトを処理する必要があり、戻って来る情報オブジェクトの型や内容を知らなければなりません。

既定では、どの情報オブジェクトも onStatus メソッドの結果を記したストリングを含む code プロパティと、"status" が "warning" が "error" のストリングを含む level プロパティを持ちます。情報オブジェクトには、追加の既定プロパティを持つものもあり、onStatus が呼び出された理由についてのさらなる情報を供給します。

様々な `onStatus` ハンドラが戻す値についての多くの情報は、*ClientSide Communication ActionScript Dictionary* を参照してください。以下の表は、サーバー・サイド・コールによって戻される情報オブジェクトのサマリーです。左のサーバー・サイド・コールが、右の `NetConnection.onStatus` ハンドラを呼び出します。(この情報は *ClientSide Communication ActionScript Dictionary* には含まれていません)。

サーバー・サイド・コール	クライアント・サイドの <code>NetConnection.onStatus</code> ハンドラのコード値
<code>application.acceptConnection()</code>	<code>NetConnection.Connect.Success</code>
<code>application.rejectConnection()</code>	<code>NetConnection.Connect.Rejected</code>
<code>application.disconnect()</code>	<code>NetConnection.Connect.Closed</code>

次のセクションでは、最も効果的なアプリケーションの書きかたについての補足的推奨事項を説明します。

スクリプトの正しい場所に `onStatus` ハンドラを置く

ネットワークやスレッドのタイミングの関係から、`onStatus` ハンドラを置くのは、スクリプト内の `connect` メソッドの前が最適です。そうでなければ、スクリプトが `onStatus` ハンドラの初期化を実行する前に接続が終了してしまいます。また、全てのセキュリティ・チェックが、`connect` メソッド内でなされ、`onStatus` ハンドラがまだセットアップされていない場合、通知は失われてしまいます。

`onStatus` のオーバーライド

`onStatus` ハンドラで何もする必要がなくても、アプリケーションがなすべき最初のことの1つは、全 Flash Communication Server オブジェクトの `onStatus` ハンドラをオーバーライドすることです。アプリケーションを書き始め、`onStatus` ハンドラをオーバーライドするには、以下の例のようにします。

```
function traceStatus(info){
    trace("Level: " + info.level + "    Code: " + info.code);
}
```

```
NetConnection.prototype.onStatus = traceStatus;
```

```
NetStream.prototype.onStatus = traceStatus;
```

```
SharedObject.prototype.onStatus = traceStatus;
```

アプリケーションを開発し、ある特定の目的のためにハンドラを実際にオーバーライドする必要があると決めると、上記のコードは消去できますが、それまではアプリケーションに關係する全メッセージを目にします。

クライアントとサーバー両方で状況メッセージを常にチェックします。

`System.onStatus` を使う

上記にリストしたオブジェクトが供給する特定の `onStatus` メソッドに加えて、Flash MX はまた、`System.onStatus` という“スーパー関数”を提供します。`onStatus` が“Error”のレベル・プロパティであるオブジェクトから呼ばれ、それに應えるよう割り当てられた関数が存在しない場合、Flash は `System.onStatus` に割り当てられた関数があればそれを呼び出します。

以下のサンプル・コードは、`onStatus` ハンドラが割り当てられていないオブジェクトにエラー・メッセージが発生すると、出力ウィンドウにメッセージを送ります。

```
System.onStatus = function(genericError){
    trace("An Error has occurred. Please try again.");
};
```

NetConnection.Connect.Failed のデバッグ

NetConnection.Connect コマンドが NetConnection.Connect.Failed の code 値で情報オブジェクトを戻してくると、サーバーへの接続は確立できません。このエラーを受け取ったら常に、標準的なトラブルシューティング問題を自分に問い掛けます。

- 適切なアプリケーションに接続していますか？ 言い換えれば、Flash Communication Server のアプリケーション・ディレクトリに、アプリケーションと同じ名前のサブディレクトリがありますか？(これは接続失敗のよくある理由です)。
- 正しいサーバーに接続していますか？
- サーバーは作動していますか？
- サーバーへの接続にプロトコル(rtmp:)を指定していますか？
- ファイアウォール内から接続している場合、正しいプロトコル(rtmp:)を指定していますか？

また、クライアントかサーバーのマシン上のソケット接続の許可された数が制限されているため接続できない場合もあります。この制限はサーバーの物理的メモリ数や接続の頻度に依存します。Windows システムでは、ソケット接続のメモリは、非ページ・メモリから割り当てられるため、ページ・ファイルにスワップできません。

オブジェクトのプロパティを取得する

特定のオブジェクトで問題が生じる原因を知りたい場合は、以下のように反復しプロパティを取得します。

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


CHAPTER 4

アプリケーション開発のチップスとコツ

このチャプターでは、*Using Flash MX* に含まれる推奨されるアプリケーション開発や最適な学習についての補足事項を説明します。Macromedia Falsh MX のデータのストリーム方法や Flash ムービーのパフォーマンスのテストや改善の仕方に関する一般的な情報は、Macromedia Flash Support Site(www.macromedia.com/go/flash_stream_optimize)にある“Streaming and file optimization techniques for Flash Player”を参照してください。

このチャプターでは、Macromedia Flash Communication Server MX 1.5 に特化した有益な情報を指摘します。アプリケーション・デザインや慣例的なコーディング、デバッグのテクニックなど一般的なセクションを含みます。チャプター 1 と 2 に記述されたコンセプトを理解し、Flash Communication Server Welcome ページにあるような簡単なサンプルアプリケーションを見てからこのチャプターを読んでください。

このチャプターの最も良い使用法は、Flash Communication Server プログラムを書き始める前に、一度ざっと全部を読み、その内容に親しんでおくことです。そうすればコードを書き始めたときに、必要に応じて参照できます。

アプリケーションのデザインと開発

このセクションは、作成する全ての Flash Communication Server アプリケーションに応用できる推奨事項を含みます。

どのサーバーでも移動可能なデザイン

アプリケーションを開発するとき、開発構成や製品構成を考えることは重要です。

アプリケーション要素には全て小文字のテキストを使用して名前をつける Macromedia は Flash Communication Server とそのアプリケーションで使用するディレクトリやファイルの名前をつける際、全て小文字を使用しスペースは使用しないことを推奨しています。この習慣をつけることによって、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ファイルを移動した場合でもアプリケーションが作動することを確実にします。

サーバーに接続するには相対パスか絶対パスを使用する 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を使って、Flash Communication Server ディレクトリにあるアプリケーション・ディレクトリに接続することができます。アプリケーション名につづけてシングル・スラッシュを使用すると、それは相対パスを意味し、コード変更せずに異なるサーバーにファイルを移動できます。ダブル・スラッシュは絶対パスを意味します。

SWF ファイルが、Flash Communication Server が作動する同じコンピュータ上にある場合は、`rtmp://server.domain.com/appName/instanceName` の短縮版の `rtmp://appName/instanceName` を使用することができます。開発用サーバーでアプリケーションをオーサリングしていて、SWF ファイルに相対パスを使用していない場合は、製品サーバーに SWF ファイル全てを移動すると、余分の作業が生じます。SWF ファイルに記述された `NetConnection.connect` ステートメントの Uniform Resource Identifier (URI) を、開発サーバーでなく製品サーバーに接続するよう編集しなければならないからです。

相対パスを使用する場合、Macromedia Flash Player は、SWF をホスティングするマシンは Macromedia Flash Communication Server であり、SWF を含むディレクトリは正しいアプリケーション・ディレクトリにあると仮定します。アプリケーションが開発サーバーと製品サーバー両方で同じ名前である限り、URI を編集する必要はありません。

SWF が Macromedia Flash Communication Server と同じマシンに存在しないなら、絶対パスが必要となり、製品時にファイルが移されたとき編集する必要があります。

相互依存のデザイン

クライアント・サイドとサーバー・サイドの ActionScript のコードは、同じアプリケーションの一部なので、相互に依存して動作しなければなりません。クライアントとサーバー・コード間の相互依存の例は、サーバー・サイド ActionScript の `call` メソッドです。これは関連するオブジェクト、クライアント・サイドの `NetConnection` オブジェクトかサーバー・サイドの `Client` オブジェクト、によって異なったふるまいをします。

例えば、以下のコードはクライアント・サイドでのネットワーク接続の作成方法と、サーバー・サイドからのコールのしかたを示します。

```
//FLA 内のクライアント・サイド ActionScript
```

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

```
my_nc.someClientMethod = function(){
```

```
    // code here
```

```
}
```

```
my_nc.connect("rtmp://hostname:port/appname);
```

```
//main.asc ファイル内のサーバー・サイド ActionScript
```

```
clientObj.call("someClientMethod");
```

`clientObj.call` に渡される引数は、クライアント・サイドの `NetConnection` オブジェクトに明示的に定義されたメソッドでなくてはなりません。これは、サーバーによってコールされる可能性のあるクライアント・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あらねばならないからです。

それと対照的に、クライアントから、サーバー・サイドのメソッドを呼び出す `call` メソッドを使用したと仮定します。

```
//FLA 内のクライアント・サイド ActionScript
```

```
NetConnection.call("someServerMethod);
```

```
//main.asc ファイル内のサーバー・サイド ActionScript
```

```
client.prototype.someServerMethod = function(){
```

```
    //code here
```

```
}
```

```
//以下も動作する
onConnect(newClient){
    newClient.someServerMethod = function(){
        //code here
    }
}
}
```

この場合は、NetConnection.call に渡される引数は、main.asc ファイルで明示的に定義されたサーバー・サイド Client オブジェクトのメソッドでなくてはなりません。これは、クライアントに呼ばれる可能性のあるサーバー・コードのどのメソッドもサーバー・サイド Client オブジェクトのプロパティであらねばならないからです。

多数のファイルを使用する

アプリケーションが複雑さを増すと、1つのスクリプトで何でもしようとするのはよくありません。機能を多くの、それぞれが特定の機能を供給するスクリプトに分けます。1つ以上のファイルを使用する場合は、1度以上使用されることをさける、“once-only inclusion”定義を使用することで、パフォーマンスを最適化することができます。

クライアント・サイドで

例えば、クライアント・サイドで、アプリケーション内の my_file.as という名前のファイルを他のファイルでも使えるようにしたい場合は、my_file.as に以下のコードを加えます。

```
if(!_root._my_file_as == null){
    _root._my_file_as = true;
    //myFile.as の全コードはここから
}
}
```

それから、他のファイルに以下のコマンドを与えます。

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

Note: #include はランタイム中でなくコードがコンパイルされたとき生じます。

サーバー・サイドで

サーバー・サイドでは、少々異なるコードを実行する必要があります。サーバー・サイド ActionScript にはグローバル・オブジェクトがありません。以下は、my_file.asc というファイルをインクルードしたい場合です。

```
if(!_my_file_asc == null){
    _my_file_asc = true;
    //myFile.as の全コードはここから
}
}
```

また、#include コマンドを使う代わりに、load コマンドを使用できます。

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

ユニークなインスタンス名を使う

典型的な配置環境は、同時に同じサーバーに多くのユーザーが接続することが想定されます。ストリーミングが交差したり共有オブジェクトが上書きされたりといった衝突を避けるために、NetConenction.connect コマンドではユニークなインスタンス名を使用します。このテクニックは、このドキュメントの他のところでも詳しく説明されています(アプ

リケーションとアプリケーション・インスタンスの配置を参照してください。ここでは Flash Communication Server アプリケーションの配置にとってこのことがどれだけ重要か、思い起こしの意味もこめて説明します。

NetConnection デバッガ・ファイルを追加する

NetConnection デバッガを使用したいなら、クライアント・サイドのソース・コードの一行めで、NetDebug.as ファイルを明快地インクルードしなくてはなりません。FLA ファイルに以下の ActionScript を追加することで、サーバーへのネットワーク接続に関する詳細を見ることができるようになります。

```
#include "NetDebug.as";
```

NetConnection デバッガの更なる情報は、NetConnection デバッガを使用するを参照してください。

強制的に Player 設定パネルを表示させる

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したり、クライアント・マシンにデータを保存しようとする時、Flash Player は許可を得るための Player 設定パネル・ダイアログ・ボックスを表示します。ユーザーはまた、player のステージで右クリック(Windows)するか、コントロール・クリック(Macintosh)することで、Player 設定パネルを開くことができます。強制的に Player 設定パネルを表示させたい場合は、以下のコードを使用します。

```
//ユーザーが最後に見たパネルを開きます
```

```
System.ShowSettings();
```

```
//プライバシー・パネルを開きます
```

```
System.ShowSettings(0);
```

```
//ローカル記憶領域・パネルを開きます
```

```
System.ShowSettings(1);
```

```
//マイク・パネルを開きます
```

```
System.ShowSettings(2);
```

```
//カメラ・パネルを開きます。
```

```
System.ShowSettings(3);
```

例えば、アプリケーションがカメラの使用を要求する場合、プライバシー設定パネルで、ユーザーが“許可”を選択できるよう通知でき、その後 System.showSettings(0)を実行します(ステージ・サイズは少なくとも 215 x 138 ピクセル必要です。これはパネルを表示するのに Flash が必要とする最小のサイズです)。さらに多くの情報は、*Client-Side Communication ActionScript* の System.showSettings 項を参照してください。

帯域幅の管理

サーバーが各クライアントに送るデータ量は、おおよその帯域幅のキャパシティを決めることで管理できます。そうするにはいくつかの方法があります。1 つは、構成ファイル(Config.cml)内の Flash Communication Server のキャパシティを構成することです。このテクニックに関する更なる情報は、*Managing Flash Communication Server* を参照してください。2 つめの方法は、NetStream.receiveVideo を使用して、入ってくるビデオの毎秒のフレーム・レートを指定することです(規定値はムービーのフレーム・レートです)。データの流れとキャパシティを合わせる 3 つめの方法は、クライアント・サイドの Camera.setQuality とサーバー・サイドの Client.setBandwidthLimit を使用して、サーバーにクライアントのキャパシティを知らせることです。このテクニックは以下のサンプルで説明します。

doc_bandwidth サンプルでは、ユーザーは異なる帯域幅を選択できます。クライアント・サイドでは、マイクとカメラ

の設定がユーザの設定に従って更新されます。サーバー・サイドでは、帯域幅の制限が更新されます。doc_bandwidth サンプルの FLA と SWF ファイルは、Macromedia Flash MX をインストールしたオーサリング・ディレクトリ下か、サーバーをインストールしたディレクトリの、/flashcom_help/help_collateral/doc_bandwidth ディレクトリにあります。

このアプリケーションを動作させて見るには、Flash Communication Server アプリケーション・ディレクトリに doc_bandwidth という名前のディレクトリを作成し、doc_bandwidth を開きます。

クライアント・サイドの ActionScript は、ユーザーに、Modem、DSL、LAN の3つの設定を選択させるインターフェイスを提供します。ステージには3つのボタンがあり、1を選択するとモデム設定、2がDSL設定、3がLAN設定となります。ユーザーが帯域幅制限の選択を変更すると、カメラとマイクの設定が更新されます。この変更はほとんど分からないので、updateBandwidth 関数のコードで、選択された設定に基づくスクリーン・サイズに変更しています。

```
_root.Modem_btn.onPress = function(){
    //カメラとマイク設定を変更し、サーバーに知らせる updateBandwidth を呼ぶ
    _root.updateBandwidth(1);
}
_root.DSL_btn.onPress = function(){
    _root.updateBandwidth(2);
}
_root.LAN_btn.onPress = function(){
    _root.updateBandwidth(3);
}
```

FLA ファイルの最初のフレームには、カメラとマイクを初期化し、サーバーからのストリームを再生し、ユーザーの選択に基づいてカメラとマイク設定を更新する、以下のコードが書かれています。

Note: ヘッドセットでなくスピーカーを使用している場合は、ハウリングを避けるために Microphone.get()のコールはコメントアウトした方がよいかも知れません。詳細はハウリングを避けるを参照してください。

```
#include "NetDebug.as"

stop();

//カメラとマイクを取得してムービーを初期化、
// カメラとマイクの設定を構成します

client_cam = Camera.get();
client_cam.setMode(150,120,5);
client_cam.setQuality(0, 90);
client_mic = Microphone.get();
client_mic.setRate(22);

// パブリッシュし再生するストリームを取得します

function getPlayStream() {
    // 新しいネットコネクションを取得します
    client_nc = new NetConnection();
```

```

// ステータス・メッセージを処理します
client_nc.onStatus = function(info) {
trace("Level: " + info.level + newline + "Code: " + info.code);
}

client_nc.connect("rtmp://doc_bandwidth/room_01");
// パブリッシュするストリームを作成します
out_ns = new NetStream(client_nc);
out_ns.attachVideo(client_cam);
out_ns.attachAudio(client_mic);
out_ns.publish("myAV");
// パブリッシュされたデータを受け取るストリームを作成します
in_ns = new NetStream(client_nc);
Output_mc.fromSrvr.attachVideo(in_ns);
Output_mc.fromSrvr.attachAudio(in_ns);
in_ns.play("myAV");
}

// 帯域幅ボタンからのコール
function updateBandwidth(b) {
// 帯域幅の変化に応じる
// "Modem" が選択されたら
if ( b == 1 ) {
client_cam.setMode(160,120,2);
client_cam.setQuality(0, 75);
client_cam.setKeyFrameInterval(3);
client_mic.setRate(5);
// デモ目的のために、画面サイズを変更します
Output_mc._height = 100;
Output_mc._width = 150;
// "DSL"が選択されたら
} else if ( b == 2 ) {
client_cam.setMode(160,120,5);
client_cam.setQuality(0, 85);
cam.setKeyFrameInterval(5);
client_mic.setRate(11);
// デモ目的のために、画面サイズを変更します
Output_mc._height = 130;

```

```

Output_mc._width = 175;
// "LAN" が選択されたら
} else /*if ( b == 3 )*/ {
client_cam = Camera.get();
client_cam.setMode(160,120,15);
client_cam.setQuality(0, 90);
client_cam.setKeyFrameInterval(10);
client_mic.setRate(22);
// デモ目的のために、画面サイズを変更します
Output_mc._height = 150;
Output_mc._width = 200;
}
// サーバーの関数setBandwidthを呼び出し、ユーザーの選択bを渡す
client_nc.call("setBandwidth", 0, b);
}
// ストリームの再生
getPlayStream();
一方、main.asc ファイルには、クライアント・コードから呼ばれ、適切に設定した帯域幅制限を更新する setBandwidth
関数が含まれます。
//サーバー・サイド・コードがアプリケーションの一部の場合、クライアントの接続を許可する onConnect 関数を定義
する必要が
//あります。
application.onConnect = function(client) {
// 接続の確立
application.acceptConnection(client);
}
// ユーザーが帯域幅の選択をするとコールされます (Modem=1, DSL=2, LAN=3)
Client.prototype.setBandwidth = function(bw) {
// クライアントの帯域幅の設定
if ( bw == 1 ) {
// Modem設定
this.setBandwidthLimit( 35000/8, 22000/8 );
} else if ( bw == 2 ) {
// DSL設定
this.setBandwidthLimit( 800000/8, 100000/8 );
} else /*if ( bw == 3 )*/ {

```

```
// LAN設定
this.setBandwidthLimit( 400000, 400000 );
}
}
```

2 バイト・アプリケーションを書く

開発環境が、アジアの言語文字セットのような 2 バイト文字の扱いを容易にするランゲージ・キットでサーバー・サイドの ActionScript を使用しているなら、サーバー・サイド ActionScript は、UTF-8 でエンコードされた ASC ファイルでなければなりません。このためには Macromedia Dreamweaver MX のような、UTF-8 スタンダードにエンコードできる JavaScript エディターが必要です。その後、JavaScript 組み込みのメソッド、例えば、ストリングをそのシステムのロケールにコンバートする `Date.toLocaleString` を使用できます。

Tip: シンプルなテキスト・エディターの中には、UTF-8 スタンダードにエンコードできないものもあります。Windows XP や Windows 2000 の Microsoft Windows Notepad は、名前を付けて保存オプションで、UTF-8 スタンダードへのエンコード保存ができます。

Dreamweaver MX で UTF-8 エンコーディングするには、エンコーディング設定とインライン入力設定を確認する必要があります。

- 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**を選択し、**エンコーディング**を選択します。その他を選択し、**オペレーティング**。システムで使用しているエンコード・ファイルとは別のファイルを作成します。
- インライン入力設定を変更するには、**編集>環境設定**か、**Dreamweaver>環境設定(Mac OS X)**を選択し、**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、**2 バイト文字の入力**を可能にします。

メソッド名に 2 バイト文字を使用するには、メソッド名はドット演算子ではなく、配列オブジェクト演算子を使用して割り当てます。

// 2 バイト文字を使ったメソッド名作成の正しい方法

```
obj["Any_hi_byte_name"] = function(){}
```

// 2 バイト文字を使ったメソッド名作成の正しくない方法

```
obj.Any_hi_byte_name = function() {}
```

アプリケーションのアンロードとリロード

アプリケーション・インスタンスは通常ガベージ・コレクションによりアンロードされます。全クライアントがアプリケーションから接続を切断してガベージ・コレクターが初めて動作すると、アプリケーションはアンロードされます。言いかえると、アプリケーションは、全クライアントが接続を切断してすぐにはアンロードされません。アプリケーションは通常スタートアップに一定の時間を使うので、しばらくの間は開いたままの方が好ましいのです。これにより接続する次のクライアントがスタートアップにかかる時間を被ることがなくなります。

既定では、ガベージ・コレクションは 20 分に 1 回生じます(使用されない I/O スレッドでは 5 分ごと)。この分数は 0 以上なら何分でも値を入れて構成することができます。詳細は *Managing Flash Communication Server* を参照してください。

また、`main.asc` ファイルを変更したときはいつでも、変更を適用するためにアプリケーションをリロードする必要があります。これにより現在アプリケーションに接続しているユーザーは全て切断されます。この操作は、Administration

Console(*Managing Flash Communication Server* を参照してください)か、Communication App インспекタ(Communication App インспекタを使用するを参照してください)を通して行うことができます。

ダイナミック・アクセス・コントロールを実行する

サーバー・サイド ActionScript は、共有オブジェクトやストリームへのダイナミック・アクセス・コントロール・リスト(ACL)機能を実行するメカニズムを提供します。既定では、全ての接続は全てのストリーム、共有オブジェクトへフル・アクセスできます。誰が共有オブジェクトやストリームを作成し、取得し、更新するか、アクセスを管理することができるのです。サーバー・サイド・アプリケーション・インスタンスへの接続はどれもサーバー・サイド上の Client オブジェクトが代理となり、どの Client オブジェクトも `readAccess`、`writeAccess` の2つのプロパティを持ちます。このプロパティを使って、毎接続ベースでアクセスを管理できます。

共有オブジェクト名とストリーム名はストリングであり、URI エンコード・データの同じルールに従うので、それらに基づきアクセスを定義できます。`client.readAccess` と `client.writeAccess` コマンドはストリング値を取ります。これらの値は複数のストリング・トークンを含むか、管理したいオブジェクト名をつけたユニークな ID を持つことができ、セミコロン(;)で分けます。以下はその例です。

```
client.readAccess = "appStream:/appSO/"
```

```
client.writeAccess = "appStreams/public;/appSO/public/"
```

これらのコールとストリング・トークンを使って、うまく定義されたパターンに従った共有オブジェクトやストリームを作成することができます。例えば、アプリケーションによって作成される全共有オブジェクトには接頭辞 `appSO` を、全ユーザーにアクセス可能な共有オブジェクトには接頭辞 `appSO/public` を、プロテクトをかけたい共有オブジェクトには接頭辞 `appSO/private` をつけると仮定します。

取得を設定するには以下のようにします。

```
client.readAccess = "appSO/"
```

サーバーによって、`appSO` で始まる共有オブジェクト接続している全ユーザーに読むことができるようになります。

同様に、コールできます。

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

クライアントは、`appSO/public/foo` のような、`appSO/public` で始まる共有オブジェクトは作成できますが、`appSO/private` へのアクセスは拒否されます。

上記の方法や、ストリームや共有オブジェクトの命名方法の工夫により、ACL を実行できます。詳細は、*Server-Side Communication ActionScript Dictionary* の `Client.writeAccess` 項を参照してください。

プライバシー・モジュールを追加する

クライアントのオーディオやビデオ・ストリームをキャプチャーするアプリケーションを作成するときは、ユーザーに行おうとしていることについての明快な注意を与え、その許可を求めるべきです。ユーザーに自分自身の顔を見せることは、“今映っている”ことを知らせる良い方法ですし、送信ライトやマイク・アクティブ・インジケータはその人にマイクがオンであることを知らせる良い方法です。ライブ・コミュニケーション・アプリケーションでは、ユーザーが、カメラやマイクを簡単にオフにしたり、受信しているオーディオを簡単に消音できるようにすることは良い考えです。

Flash Communication Server アプリケーション・ディレクトリにあるサンプル・アプリケーションの多くで、こういった

テクニックの実行方法を学ぶことができます。以下のサンプルは、記録する前にユーザーの許可を求める1つの方法を示しています。このコードのサンプルは、インストールした Macromedia Flash MX オーサリング・ディレクトリ下か、サーバーのインストール・ディレクトリの/flashcom_help/help_collateral_doc_approval ディレクトリにあります。

プライバシー・モジュールを作成するには

1. Flash MX オーサリング環境で、ファイル>新規を選択し、新しいファイルを開きます。
2. Components パネル(ウィンドウ>Components)から、Flash UI Components 下の Push Button シンボルをステージ下部にドラッグします。プロパティ・インスペクタ(ウィンドウ>プロパティ)で、インスタンス名に Record_btn、Label 名に Record、Click Handler 名に doRecord をつけます。
3. ライブラリ・パネル(ウィンドウ>ライブラリ)のオプション・メニューから、新規シンボルを選択し、warnRec_mc の名前をつけます。表示されていない場合は、高度な設定をクリックして、リンケージ下の ActionScript に書き出しを選択して、OK をクリックします。
4. warnRec_mc ステージ上で、テキスト・ツールを選択しテキスト・ボックスをドラッグします。プロパティ・インスペクタで、テキスト・ボックスのタイプを静止テキストにします。以下のテキストをテキスト・ボックスに入力(コピー・アンド・ペースト)します。**あなたの画像は記録されるか配信され、後日パブリッシュされます。**
Note; このテキストはあくまでも例です。使用する正確な言い回しは、お使いのアプリケーションやサービス、それら起因するプライバシーや法律、その他の問題の性質によって変更してください。
5. Push Button シンボルの2つのインスタンスを warnRec_mc ステージ上にドラッグします。プロパティ・パネルで、1つのボタンには、インスタンス名に Yes_btn、Label 名に Yes、Click Handler 名に onYes をつけます。もう1つのボタンには、インスタンス名に No_btn、Label 名に No、Click Handler 名に onNo をつけます。
6. ムービーの最初のフレームのアクション・パネルに以下のコードをコピー・アンド・ペーストします。

```
stop();  
var userAnswer = false;  
function doRecord() {  
    // If user selects button to record...  
    if (Record_btn.getLabel() == "Record") {  
        // Call function to get user 's approval to record.  
        getApproval();  
        // When user has provided an answer, if the answer 's  
        // yes, begin to record, otherwise, send a status  
        // message  
        WarnNow_mc.onUnload = function () {  
            // If user approved, Record.  
            if (userAnswer == true) {  
                //  
                // Record  
                // .
```

```

// .
// .
//
trace("Recording...");
// Change the button label
Record_btn.setLabel("Stop");
// Else if user refused, give status.
} else {
trace("User did not approve streaming.");
}
}
// Else if user selects button to stop recording...
} else if (Record_btn.getLabel() == "Stop") {
trace("Stopped Recording.");
// Change the button label
Record_btn.setLabel("Record");
}
}
function getApproval(){
// Attach the movie clip to prompt user input,
// and align it on the stage.
_root.attachMovie("warnRec_mc", "WarnNow_mc", 1);
var x = 275;
var y = 160;
setProperty("WarnNow_mc", _x, x);
setProperty("WarnNow_mc", _y, y);
// If the user selects the Yes button, they
// approved the recording. So, set userAnswer
// to true, unload the movie clip and return
// the updated userAnswer value.
WarnNow_mc.Yes_btn.onRelease = function () {
userAnswer = true;
trace("userAnswer: " + userAnswer);
WarnNow_mc.unloadMovie();
trace("Returning: " + userAnswer);
return userAnswer;
}
}

```

```

}

// If the user selects the No button, they
// do not want to record. So, set userAnswer
// to false, unload the movie clip and return
// the updated userAnswer value.
WarnNow_mc.No_btn.onRelease = function () {
userAnswer = false;
trace("userAnswer: " + userAnswer);
WarnNow_mc.unloadMovie();
trace("Returning: " + userAnswer);
return userAnswer;
}
}

```

- Flash Communication Server アプリケーション・ディレクトリに doc_approval 名のディレクトリを作成し、そこに doc_approval.fla の名前でファイルを保存します。

コーディングの慣例

このドキュメントでは、ActionScript でコーディングし、Macromedia Flash MX でアプリケーションを構築するために特化して作成された最良の慣例システムのあらましを説明します。このガイドラインを使用するアプリケーションは、能率が高く、理解しやすいものになります。基本にのっとった ActionScript コードは、デバッグも再利用も容易です

命名のガイドラインに従う

アプリケーションの命名スキームは首尾一貫したものでなくてはならず、読みやすさが大切です。これは、名前が分かりやすい言葉や語句であるべきだということを意味します。一番大切な機能や目的はその名前を見ればすぐに分かるものであるべきです。ActionScript はダイナミックな言語ですので、名前はまた名前によって参照されるオブジェクト型を定義する接尾辞を含むべきです。一般的には、名詞-動詞、形容詞-名詞の語句が名前の選択には最も自然です。例えば、

- ムービー名 : my_movie.swf
- URI に加えるもの : course_list_output
- コンポーネントかオブジェクト : ProductInformation
- 変数かプロパティ : userName

関数名と変数は小文字で始めます。オブジェクトやオブジェクト・コンストラクタは大文字にします。大文字と小文字の混在もまた、変数の名前をつけるときは、アプリケーション内で首尾一貫して使用されているなら、たとえ他の形式が可能であっても、推奨されます。変数名は文字、数字、アンダースコアのみ含むことができます。しかし、数字、アンダースコアで変数名を始めてはいけません。

以下は認められない変数名のサンプルです。

```
_count = 5; // アンダースコアで始める
```

```
5count = 0; // 数字で始める
```

```
foo/bar = true; // スラッシュを含む
```

```
foo bar = false; // スペースを含む
```

加えて、ActionScript で使用される言葉(予約語)は名前として使用してはいけません。また、たとえ今 Macromedia Flash Player が一般的なプログラミング・コンストラクタ名をサポートしていなくても、そうしたコンストラクタ名の使用は避けます。こうしたことが、Pkayer が将来バージョンアップしてもアプリケーションとのコンフリクトを避けることになるのです。例えば、MovieClip = "myMovieClip"や、case = false といったコマンドは使用しません。

ActionScript は、ECMAScript に準じているので、アプリケーション作成者は、現在や将来の ECMA 仕様書を参照して、予約語のリストを調べることができます。Flash MX は定数値の使用を強制しないので、作成者は変数の意図を示す命名法を使用できます。変数名は小文字か、小文字で始める大文字小文字混在であるべきで、定数名は全て大文字であるべきです。例です。

```
course_list_output = "foo"; // 変数、全て小文字
```

```
courseListOutput = "foo"; // 変数、混在
```

```
BASEURL = http://www.foo.com; // 定数、全て大文字
```

```
MAXCOUNTLIMIT = 10; // 定数、全て大文字
```

```
MyObject = function(){}; // コンストラクタ関数
```

```
f = new MyObject(); // オブジェクト
```

最後に、SWF ファイルは全て、アンダースコアで分けられた小文字の名前(例えば、lower_case.swf)にすべきです。このことで、UNIX のような大文字小文字を区別するオペレーティング・システムにファイルを移すことが容易になります。

こうしたシンタックスの推奨事項は簡単なガイダンスであることを覚えておいてください。最も重要なことは、命名法を選択してそれを一貫して使用することです。

コードのコメント

アプリケーション内では常にコードにコメントをつけます。コメントは、コードが何をするために書かれたかについてストーリーを語る作成者の機会です。コメントは、アプリケーション作成中になされた決定の証拠となります。アプリケーションにどんなコードを書くか、選択されたポイントに全て、その選択を記述し、その理由もコメントに書き込みます。

特定の問題に取り掛かっているコードを書くときは、そのコードを見るかも知れない後の開発者にその問題を明確にさせるコメントを加えます。このことで、開発者はその問題に取り組みやすくなります。

以下は変数の簡単なコメントの例です。

```
var clicks = 0; // ボタンをクリックする回数の変数
```

ブロック・コメントは、多くの行にわたりコメントするとき便利です。

```
/*
```

```
ボタンが押された回数を追う、クリック変数を
```

```
初期化する
```

```
*/
```

特定のトピックスを示す一般的なメソッドは、

- `//:TODO:` トピック
ここですることがあることを示します。
- `//:BUG:` [bugid] トピック
ここで分かっている問題を示します。コメントはまた問題を説明し、適切な場合はバグ ID を加えます。
- `//:KLUDGE:` ちょっと問題あり
以下のコードはすっきりしたコードでないが、最善の慣例になっていないことを示します。このコメントは、他の人に注意を喚起し、次の異なるコーディングについての助言となります。
- `//:TRICKY:`
つづくコードが相互作用していることを開発者に知らせます。またそれを修正する前に、よく考えるよう開発者に助言します。

アクションを一緒にする

可能ならいつでも全てのコードはひとつの場所に置かれるべきです。このことで、コードを見つけやすく、デバッグしやすくなります。Macromedia Flash MX ムービーで難しいことのひとつは、コードを見つけることです。ほとんどのコードをひとつのフレームに書けば、この問題は解決します。めったにないことですが、コードを置く最も良い場所はフレーム 1 です。

多量のコードが最初のフレームにあるときは、以下のように、コメントでセクションを分けて、読みやすくします。

```
/** ボタン関数セクション */
/** 定数値 */
```

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、開始状態をセットするために使われます。それはアプリケーション内で最初の関数コールであるべきです。この関数は、プログラムにおいて初期化の作用をする唯一のコールであるべきです。他のコールは全てイベント・ドリブンであるべきです。

```
//フレーム 1
this.init();

function init(){
    if(this.inited != undefined)
        return;
    this.inited = true;
    //ここに初期化コード
}
```

ローカル変数には var を使う

ローカル変数全てにはキーワード `var` を使用します。これによりグローバルにアクセスすることが避けられ、さらに重要なことは、不注意に上書きされることも避けられます。例えば、以下のコードは変数を宣言するキーワード `var` を使用せず、他の変数を上書きしてしまうものです。

```
counter = 7;

function loopTest(){
```

```

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

```

このコードは以下を出力します。

```
7,7,0,1,2,3,4,5
```

このコード内で、メインのタイムライン上の counter 変数は関数内の counter 変数によって上書きされます。いかが正しいコードで、両方の変数宣言にキーワード var を使用しています。関数内で var 宣言をすることで、上記コードのバグを修正しています。

```

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

```

オブジェクト作成に prototypes を使用する

オブジェクトを作成する際には、オブジェクトのインスタンス全てで共有されるオブジェクト関数とプロパティをオブジェクトのプロトタイプにアタッチします。このことで、各関数の唯一のコピーがメモリ内に存在することになります。一般的な決まりでは、コンストラクタ内では関数を定義しません。これは、各オブジェクト・インスタンスの同じ関数を持つ別のコピーを作成し、メモリを不必要に浪費します。以下のサンプルは、オブジェクト作成の最善の慣例です。

//オブジェクト作成の最善の慣例

```

MyObject = function()
{
}
MyObject.prototype.name = "";
MyObject.prototype.setName = function(name)

```

```

{
    this.name = name;
}
MyObject.prototype.getName = function()
{
    return this.name;
}

```

以下の例は、オブジェクト作成の正確な技術を示しますが、オブジェクトのプロパティがプロトタイプ・ベースでなくインスタンス・ベースであってほしいときのみ使用します。

```

MyObject = function()
{
    this.name = "";
    this.setName = function(name)
    {
        this.name = name;
    }
    this.getName = function()
    {
        return this.name;
    }
}

```

最初の例では、MyObject の各インスタンスはオブジェクトのプロトタイプで定義された同じ関数とプロパティを示します。つまり、MyObject オブジェクトがいくつ生成しようとも、メモリ内にはただ 1 つの getName のコピーしか存在しないのです。

ふたつめの例では、MyObject が生成した各インスタンスは、各プロパティと関数のコピーを作ります。これらの余分のプロパティや関数は、よけいなメモリを使用し、多くの場合、有利には働きません。

コード・ヒントを表示する変数名をつける

Macromedia Flash MX ActionScript エディターは、コード・ヒントをサポートしています。これを利用するには、変数にある形式をつけて名前をつけます。既定の形式は、変数名に変数型を表す接尾辞を加えます。以下はサポートされる接尾辞ストリングの表です。

オブジェクト型	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
NetStream	_ns	a_ns
SharedObject	_so	myRemote_so

ファイル型とパス

Flash Communication Server は、あるフィーチャーが使用されるとファイルを生成します。このセクションでは、その生

成したファイルを Flash Communication Server が保存する場所について説明します。

Note: このセクションでは、アプリケーション名と一致するインスタンス名を通してしていると仮定しています(アプリケーションとアプリケーション・インスタンスを配置するを参照してください)。インスタンス名を通していない場合には、Flash Communication Server は、登録したアプリケーション・ディレクトリ内の`¥_definst_`(既定のインスタンスの意味)という名前のサブディレクトリにファイルを保存します。

記録されたストリーム・ファイル

オーディオやビデオ、データ・ストリームを記録するメソッド(例えば `NetStream.publish`)を使用すると、Flash Communication Server は、`ファイル名.flv` と `ファイル名.idx` という 2 つのファイルを生成します。ここでのファイル名は、ストリームを記録したメソッドに渡される文字列です。これらのファイルは、記録されたストリームとそれに関連するインデックス・ファイルです。例えば、`NetStream.publish("me", "record")` というコマンドを出すと、`me.flv` と `me.idx` という名前のファイルが生成されます。

Flash Communication Server は、アプリケーションのストリーム・ファイルを生成すると、アプリケーション・インスタンス・サブディレクトリとともにストリーム・ディレクトリを作成します。例えば、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と、`chat.flv` と `chat.idx` は、`/applications/ChatApp/streams/MondayChat` に保持されます。`TuesdayChat` という `ChatApp` のインスタンスを動作させると、そのファイルは `applications/ChatApp/streams/TuesdayChat` に保持されます。

`Sorenson Squeeze` などの特殊なビデオ・アプリケーションで作成された FLV ファイルを再生したいなら、Flash Communication Server が見つけやすい場所、つまり前のパラグラフで述べた通り、`/streams` ディレクトリのサブディレクトリに、ファイルを置きます。アプリケーションを動作させると、Flash Communication Server は、IDX ファイルを生成し、同じサブディレクトリに保持します。サブディレクトリが正しいアプリケーション・インスタンスのサブディレクトリにある限り、サブディレクトリを作成して、さらに FLV ファイルを構成できます。例えば、`/streams/TuesdayChat` 内なら、`/streams/TuesdayChat/morning` や `streams/TuesdayChat/afternoon` などのサブディレクトリを追加作成し、その FLV ファイルを保持できます。

MP3 ファイルを再生したい場合は、付録の 171 ページ、「MP3 ファイルの再生」を参照してください。

ストリームの上書きを避けるには、ユーザーやストリームなどの名前にユニークなものを使用するように気をつけます。例えば、新しいストリームを記録するなら、インクリメンタルな値を与えます。

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

記録されたストリーム・ファイルを削除する情報については、ストリーム・オブジェクトを参照してください。インスタンス名やファイル保持についての更なる情報は、*Client-Side Communication ActionScript Dictionary* の `NetConnections.onConnect` と `NetStream.publish` を参照してください。

共有オブジェクト・ファイル

ローカルでありリモートであれ、共有オブジェクトはどれも、ダイナミックに存在でき(アプリケーション・インスタンスの存在中)、使用できる永続するデータとして保存できます。このセクションでは、3 タイプの作成できる永続的共有オブジェクトを説明します。永続するローカル共有オブジェクト、サーバー上でのみ永続するリモート共有オブジェクト、クライアントとサーバー上で永続するリモート共有オブジェクトです(一般的な共有オブジェクトについては、共有オブジェクトの理解を参照してください)。

ローカル共有オブジェクトは、メモリとハードディスクの空き容量が許す限り、クライアント上に永続します。しかし、既定では Flash は 100K までしかローカルに永続する共有オブジェクトを保存できません。さらに大きなオブジェクトを保存しようとするれば、Flash Player がローカル保存ダイアログ・ボックスを表示し、ローカルへのアクセス要求を許可するか拒否するかユーザーに示します。ユーザーはまた、このダイアログ・ボックスを使用して、ローカルで永続する全てのリモート共有オブジェクトを消去することもできます。詳細は、*Client-Side Communication ActionScript Dictionary* の SharedObject 項にある“Local disk space considerations”を参照してください。

永続するローカル共有オブジェクト

永続する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できません。永続するローカル共有オブジェクトは、拡張子.sol を持ち、クライアント・マシン上の共有オブジェクトを作成したユーザーに関連するディレクトリに保持されます。Windows での既定の場所は、C:\Documents and Settings\¥userName¥Application Data¥Macromedia¥FlashPlayer¥serverSubdomain¥pathToMovie¥MovieName.swf になります(既定は、SharedObject.getLocal コマンドの localPath パラメータに値を渡すことでオーバーライドできます)。

例えば、ログオン ID が jsmith で、ローカル・マシン(localhost)でサーバーを動作させており、ムービー名は myMovie.swf で、C:\test ディレクトリにある場合は、下記のコードで生成されるローカル共有オブジェクトは、以下に保持されます。

```
C:\Documents and Settings\¥jsmith¥Application Data¥Macromedia¥FlashPlayer¥localhost¥myMovie.swf
¥myObject.sol。
```

```
my_so = SharedObject.getLocal("myObject");
```

リモートで永続する共有オブジェクト

サーバー上でのみ永続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 コマンドかサーバー・サイドの SharedObject.get コマンドで、persistence パラメータに true の値を渡すことで作成できます。これらの共有オブジェクトは拡張子.fso がつき、サーバー上の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持されます。Flash Communication Server はこれらのディレクトリを自動的に作成するので、各インスタンス名ごとにディレクトリを作成する必要はありません。Windows での既定場所は、C:\Program Files¥Macromedia¥Flash Communication Server MX¥application¥appName¥sharedobjects¥instanceName です。

例えば、下記コードにより作成されたリモート共有オブジェクトは、..\Applications¥myWhiteboard¥sharedobjects¥Monday¥myObject.fso に保持されます。

```
my_nc = new NetConnection();
my_nc.connect("rtmp://myFlashServer.myDomain.com/myWhiteboard/monday");
// 3つめのパラメータ"true"は、オブジェクトがサーバー上で永続することを指定しています。
my_so.getRemote("myObject", my_nc.uri, true);
my_so.connect(my_nc);
```

リモートとローカルで永続する共有オブジェクト

クライアントとサーバー上で永続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 コマンドで persistence パラメータにローカル・パスを渡すことで作成できます。ローカルで永続する共有オブジェクトは拡張子.sor がつき、クライアント上の指定されたパスに保持されます。リモートで永続する.fso ファイルは、サーバー上の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持されます(上記"リモートで永続する

共有オブジェクト”を参照してください)。

ローカルで永続するリモート共有オブジェクトの場所を示す部分的パスを指定することで、同じドメインの複数のムービーが同じ共有オブジェクトにアクセスできるようになります。詳細は、*Client-Side Communication ActionScript Dictionary* の `SharedObject.getRemote` を参照してください。

スナップショットとサムネイル

このセクションは、アプリケーション内で使用できる画像として、ビデオを1フレームを取得するテクニックを比較します。コードのサンプル・ファイルは、Macromedia Flash MX オーサリング・ディレクトリ下か、サーバーのインストール・ディレクトリの、`/flashcom_help/help_collateral/doc_snapshot` と `/doc_thumbnails` にあります。アプリケーションを動作させて見るには、Flash Communication Server アプリケーション・ディレクトリに `doc_snapshot` と `doc_thumbnails` という名前のディレクトリを作成し、当該 SWF ファイルを開きます。

これらのサンプルは見た目は同じように動くように見えますが、実際は大変異なる動作をしています。`doc_snapshots` サンプルは、1つのフレームだけを記録し、それを表示します。`doc_thumbnails` サンプルは、多数のフレーム(ストリーム全部)を記録しますが、最初のフレームしか表示しません。スナップショット・アプリケーションは、ストリームを記録するより、“写真を撮り”たいとき、使用するのが良いでしょう。一方、サムネイル・アプリケーションは、多数のストリームの中からユーザーに選択させたいとき使用するのが良いでしょう。各ストリーム上に何が記録されているか、ユーザーに考えさせるようなストリームを表示させることもできます。

スナップショット

Client-Side Communication ActionScript Dictionary の `NetStream.attachVideo` 項を見ると、以下の使用方法が書かれています。

```
myStream.attachVideo(source! null [, snapShotMilliseconds])
```

`snapShotMilliseconds` パラメータは、1つのスナップショットを送る(0の値を与えることで)か、ビデオに、指定されたミリ秒数を加える正数を与えることで、継時画像のような一連のスナップショットを送るために使用されます。

`snapShotMilliseconds` パラメータを指定すると、記録されたフレーム間で再生中どれだけ時間経過させるか管理できません。

`doc__snapshot fla` の、以下のクライアント・サイド ActionScript コードは、サーバーに接続しローカルでカメラ出力を再生します。ユーザーがスナップショットを撮ると、イベント・ハンドラ `doRecord` が `out_ns.attachVideo(client_cam,0)` をコールします。2つめのパラメータ、0は1フレームだけを記録します。

```
#include "NetDebug.as"

// 記録状態の変数
RecState_box.text = 0;

// スナップショット数
numSnaps = 0;

// ボタン・ラベルの初期化
Record_btn.setLabel("Record");

// snapshot appに接続し、接続状況を取得します
function doConnect() {
    client_nc = new NetConnection();
```

```

client_nc.onStatus = function(info) {
trace("Level: " + info.level + newline + "Code: " + info.code);
};

client_nc.connect("rtmp://doc_snapshot/room_01");
}

// 記録しスナップショットを取得するストリームの作成
function initStreams() {
// 記録するストリーム
out_ns = new NetStream(client_nc);
out_ns.onStatus = function(info) {
trace("Level: " + info.level + newline + "Code: " + info.code);
};

// 再生するストリーム
in_ns = new NetStream(client_nc);
in_ns.onStatus = function(info) {
trace("Level: " + info.level + newline + "Code: " + info.code);
};
}

// カメラのインスタンスを取得し、ローカルのビデオ、Live_videoにアタッチ、ストリームを出力します
client_cam = Camera.get();
Live_video.attachVideo(client_cam);

// パブリッシュしスナップショットを表示させるボタンのイベント・ハンドラ
function doRecord() {
// 記録していないなら記録を始めます
if (RecState_box.text == 0) {
// スナップショット・ウィンドウのクリア
Snapshot_mc.removeMovieClip();
// スナップショットを撮ります
out_ns.attachVideo(client_cam, 0);
out_ns.publish("myRecording"+numSnaps, "record");
// ラベルを"Stop"にします
Record_btn.setLabel("Stop");
// 記録状況の更新
RecState_box.text = 1;
// 記録中ならストップします
} else {

```

```

// 記録されたストリームのパブリッシュをストップします
out_ns.publish(false);

//ストリームを閉じると同じものを使ってまたパブリッシュできます
out_ns.close();

// ラベルを"Record"にします
Record_btn.setLabel("Record");

// 記録状況の更新
RecState_box.text = 0;

showSnapshot();

}

}

// SnapView_videoビデオ・オブジェクトにストリームをアタッチして再生するためにスナップショットを表示します
function showSnapshot() {
//出力可能なムービー・クリップで、ビデオ・オブジェクトSnapView_videoを含むView_mcから新しいムービー・クリップを作成、
//新しい名前と深度(衝突を避けるために使用されるステージ上での他のView_mcに関連する数)を与えます。
_root.attachMovie("View_mc", "Snapshot_mc", numSnaps);
Snapshot_mc.SnapView_video.attachVideo(in_ns);
Snapshot_mc._x=375;
Snapshot_mc._y=225;
in_ns.play("myRecording"+numSnaps);
numSnaps++;
}
doConnect();
initStreams();

```

サムネイル

サムネイルは元の記録より小さな別物です。スナップショットのように、記録されたビデオ・ストリームの1フレーム分の画像を供給します。スナップショットの例では、データの1フレームを送るだけでした。このサンプルでは、記録されたストリームの最初のフレームのみ再生します。

前のサンプルでは、`NetStream.attachVideo` オブジェクトのプロパティ、`snapshotMilliseconds` を使って、ストリームのスナップショットを取得しました。このサンプルでは、ストリームの1フレームの画像の取得が必要です。ここではしかし、`NetStream.play:thumbStream.play("myRecording",0,0,true)` コールを使用します。

`NetStream.play` メソッドの最初のパラメータは、再生するもの、"myRecording"を記述します。2つめのパラメータは、ストリーム上の開始場所(0 は記録されたストリームの初めを意味します)です。3つめは、終了場所(0 は1フレームのみの再生を意味します)です。4つめは `Player` に以前の再生ストリームを消すように伝えます。詳細は、*Client-Side Communication ActionScript Dictionary* の `NetStream.play` 項を参照してください。

doc_thumbnails fla 内の以下のクライアント・サイド ActionScript コードでは、サーバーは、ユーザーが記録停止を選択するまで入ってくるストリームを記録します。その後、thumb_ns.play("myRecording",0,0,true)をコールして、記録の最初のフレームのみクライアントに戻します。

```
#include "NetDebug.as"

recState = 0;

Record_btn.setLabel("Record");

showThumb();

function doConnect()

{

client_nc = new NetConnection();

client_nc.onStatus = function(info) {

trace("Level: " + info.level + newline + "Code: " + info.code);

}

client_nc.connect("rtmp://doc_thumbnails/room_01");

}

function initStreams()

{

out_ns = new NetStream(client_nc);

out_ns.onStatus = function(info)

{

if (info.code == "NetStream.Publish.Success") {

var description = "You have published.";

trace(description);

} };

thumb_ns = new NetStream(client_nc);

}

function initMovie()

{

client_cam = Camera.get();

Live_video.attachVideo(client_cam);

out_ns.attachVideo(client_cam);

}

function doRecord()

{

if (recState == 0) {

out_ns.publish("myRecording", "record");
```

```

Record_btn.setLabel("Stop");

recState = 1;

} else {

out_ns.publish(false);

Record_btn.setLabel("Record");

recState = 0;

showThumb();

}

}

function showThumb()

{

ThumbView_vc.attachVideo/thumb_ns);

thumb_ns.play("myRecording", 0, 0, true);

}

doConnect();

initStreams();

initMovie();

```

CHAPTER 6

コミュニケーション・コンポーネントを使う

このチャプターでは、Macromedia Flash Communication Server MX 1.5 に含まれるコミュニケーション・コンポーネントについて説明します。

コミュニケーション・コンポーネントを使用した自作のアプリケーションを構築することができるようになるには、サーバーへの接続、再生、パブリッシュ、ストリームの記録、共有オブジェクトの使用など、基本的な機能を持った ActionScript をうまく書けるようになる必要があります。コン j ポーネントはアプリケーションを速く開発する助けとなりますが、Flash Communication Server とそのアプリケーションの核となるコンセプトをまず理解しなければなりません。コンポーネントを使用する前に、11 ページのチャプター 1 “始めよう”を読み、チュートリアルを完成させ、Flash Communication Server Welcome ページのサンプル・アプリケーションで練習してください。

コンポーネントを使用して開発を始めるに当たって、Macromedia は SimpleConnect コンポーネントを、他のコンポーネントについて学ぶ前に読むよう推奨しています。RoomList コンポーネントは最も上級で追加のサーバー・サイド・スク립ティングも必要ですから、このコンポーネントのサンプルで作業したいと思われるかも知れません。

このチャプターの各セクションは、コンポーネントの使用方法和必要なスクリプト(もしあれば)を説明し、作成してコンポーネントをテストできるアプリケーションのサンプルを紹介します。コンポーネントを使用した各セクションの次は、コンポーネントのアプリケーション・プログラミング(API)のセクションで、コンポーネントのオブジェクトとメソッドを説明します。コミュニケーション・コンポーネントは Macromedia Flash Player 6 とそれ以降でサポートされません。

コンポーネントのフレームワークについて

このチャプターで説明するコミュニケーション・コンポーネントは、Flash Communication Server と併用してアプリケーションを早く構築するために使用する一般的な機能を提供します。これらのコンポーネントは、Flash Communication Server コンポーネント・フレームワークの上に構築されています。フレームワークを使用して、自作のコンポーネントを作成したり、Flash Communication Server と併用してコンポーネントを拡張できます。

コンポーネント・フレームワークに関する詳細は、Macromedia Designer & Developer Center(www.macromedia.com/desdev/mx/flashcom/articles/frameworkd.html)のコンポーネント・フレームワークを理解するを参照してください。

簡単なアプリケーションでコンポーネントを使用する

開発環境をセットアップし、Flash Communication Server と基本的な ActionScript 関数に慣れたら、コンポーネントを使ったコミュニケーション・アプリケーション作成の準備は整いました。以下のサンプルでは、PeopleList コンポーネントを使い、アクティブ・ユーザーのリストを表示する、簡単なアプリケーションを作成します。

簡単なアプリケーションを作成する：

1. まだ作動していない場合は、Flash Communication Server を起動します。
Windows では、プログラム>Macromedia Flash Communication Server MX>Start Service を選択します。
UNIX では、シェル・ウィンドウを開き、Flash Communication Server インストール・ディレクトリに移動し、./server start と打ち込みます。その後 Admin Service をスタートさせるには、./adminserver start と打ち込みます。
2. Flash Communication Server MX アプリケーション・ディレクトリに com_test という名前のディレクトリを作成します。
3. Macromedia Flash MX で、ウィンドウ>コンポーネントを選択し、コンポーネント・ポップアップ・メニューから Communication Components を選択します。



4. コンポーネント・パネルから PeopleList コンポーネントをステージ上にドラッグします。
5. プロパティ・インスペクタで、インスタンスに peopleList_mc の名前をつけます。
6. サーバーに接続するには、タイムラインの最初のキーフレームを選択し、そのフレームのアクション・パネルに、以下のコードを書きます。

//新しいネットワーク・コネクションを作成し、ストップ・アクションを加えます

```
nc = new NetConnection();
```

```
stop();
```

//ステータス・ハンドラの作成

```
nc.onStatus = function(info) {trace(info.code);}
```

//リストをクリーン・アップするオプション・コール

```
peopleList_mc.close();
```

//アプリケーションに接続し、ユーザ名を与えます

```
nc.connect("rtmp:/com_test","Jane");
```

Note:Flash Communication Server が localhost で動作していない場合、このテスト・アプリケーションにはフル URI(Uniform Resource Identifier)が必要となります(例：rtmp://www.myflashcomdomain.com/com_test)。NetConnection.connect メソッドについての詳しい内容は、Macromedia Flash Communication Server MX ドキュメントを参照してください。

7. nc NetConnection インスタンスを使って、PeopleList_mc ムービー・クリップをサーバーへ接続させるには、タイムラインの1番目のキーフレームのアクション・パネルに以下のコードを加えます。

```
PeopleList_mc.connect(nc);
```

8 保存した後、制御>ムービープレビューを選択しテストします。

以下は onStatus メッセージを示しています。これはサーバーへの接続が成功した場合出力ウィンドウに表示されます。



この時、ユーザ名はリストに表示されていません。PeopleList コンポーネントはサーバーからユーザのリストを取得するだけで、サーバー・サイドの結果を取得する命令はまだ加わっていないのです。サーバー・サイド・スクリプトに何行か書き加えることで、コンポーネントはサーバーからユーザ名を取得できるようになります。

9 アプリケーション・ディレクトリに main.asc という新しいファイルを作成し、components.asc ファイルをロードする次のコードを記述します。

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

Note:サーバーは scriptlib ディレクトリにアクセスし、components.asc ファイルをロードする必要があります。

components.asc ファイルのロードに関する詳しい説明は、11ページの"Your main.asc.file"を参照してください。

10 クライアントからのユーザ名を受け取り、接続を許可する onConnect ハンドラを、以下のように作成します。

//newUserName はクライアント・サイドの nc.connect call からのパラメータ

```
application.onConnect = function(newClient,newUserName)
{
    //このファンクションを通るユーザ名でグローバルなユーザ名を設定する
    gFrameworkFC.getClientGlobals(newClient).username = newUserName;
    //ユーザからの接続を許可する
    application.acceptConnection(newClient);
}
```

11 main.asc ファイルを保存します。

Note:アプリケーション起動後 ASC ファイルを書き変えた時は、アプリケーションを再ロードさせる必要があります。再ロードは Communication App Inspector で行います。Flash MX で(SWF ウィンドウでなく Flash MX のウィンドウが開かれていることを確認してください)、ウィンドウ>Communication App Inspector を選択します。ログイン後 Active Apps インスタンスウィンドウから com_test¥_definst_を選択します。View Detail をクリック後、Reload App をクリックします。

1 2 Flash MX オーサリング環境で、制御>ムービープレビューを選択する。以下は接続した時の値を示します。



このサンプルは、コンポーネントを使用するといかに速くアプリケーションを作成できるかを示すために紹介しました。次のセクションでは、SimpleConnectコンポーネントを使って、さらに開発過程を簡単にします。

Tip: onConnectAcceptとonConnectRejectステートメント、それらをコミュニケーション・コンポーネントと併用することに関する情報は、アプリケーション・オブジェクトを参照してください。

SimpleConnect コンポーネントを使う

コミュニケーション・コンポーネントを使ってアプリケーションを作成する時は、通常いくつかのコンポーネントを加えて作成しますが、それぞれがサーバーへの接続確立を必要とします。サーバーに接続するアプリケーション内で、同じネットワーク接続インスタンスを使用し、全てのムービー・クリップを接続させることができます。SimpleConnect コンポーネントは、あらゆるコミュニケーション・オブジェクトの接続を制御します。加えて、SimpleConnect は作成したアプリケーションで、ユーザがログインするインターフェイスも提供します。

次のサンプルは com_test アプリケーションを使用し、SimpleConnect の働きを示すものです。PeopleList コンポーネントは NetConnection オブジェクトを使用せず、SimpleConnect の接続を使用してサーバーとやりとりします。SimpleConnect にサーバーへの接続を受け持たせることで、コードを能率的に機能させることができ、書かなくてはならない ActionScript を大幅に減らしてくれます。サンプルでは、SimpleConnect はサーバーへの接続を制御し、ActionScript を書くことなく PeopleList をその接続に結合させます。nc.connect コールの中にユーザ名を入れるという大変なコードを書かないで、実行させることができます。

SimpleConnect コンポーネントを使って接続するには :

1 Flash Communication Server が起動していることを確認します。

2 Flash Communication Server MX のアプリケーション・ディレクトリ(通常は¥flashcom¥applications)にアプリケーション・ディレクトリを作成し、com_test_simcon の名前をつけます。

3 com_test_simcon 内に main.asc の名前で新しいファイルを作成します。components.asc ファイルをロードする以下のコードを記述します。

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

Note:main.ASC ファイルは com_test_simcon.asc と名づけても使用できます。

4 Flash MX で Communication Components パネルが表示されていない場合は、ウィンドウ・メニュー>コンポーネントを選択し、コンポーネント・ポップアップ・メニューから Communication Components を選択します。

5 コンポーネント・パネルからステージ上に peopleList コンポーネントをドラッグします。

6 プロパティ・インスペクタで、peopleList コンポーネントのインスタンス名に peopleList_mc をつけます。

7 Flash Communication Server へ接続させるために、ステージ上に SimpleConnect コンポーネントをドラッグします。

8 プロパティ・インスペクタで、以下のパラメータを与えます。

Application Directory テキスト・ボックスに、rtmp://com_test_simcon と入力します。URI の com_test_simcon 部分は、アプリケーション名で、ステップ 2 で作成したディレクトリ名に一致させます。

Communication Components テキスト・ボックスをダブル・クリックし、現れた値ダイアログ・ボックスで、+印をクリックし、peopleList_mc と入力、OK をクリックします。

この値を与えることによって、SimpleConnect コンポーネントは PeopleList コンポーネントの接続を制御します。

SimpleConnect コンポーネントがサーバー接続に成功すれば、PeopleList コンポーネントは自動的にサーバーに接続します。

9 このテストファイルを com_test_simcon.fla 名で保存、パブリッシュします。

10 SWF ファイルを開くか、Flash MX オーサリング環境で制御>ムービープレビューを選択する。適当なユーザ名でログインする。

入力したユーザ名が、以下のように PeopleList コンポーネント内に表示される。



SimpleConnect コンポーネントはログイン時のインターフェイスを提供します。アクティブなユーザのリストを増やすには、このアプリケーションの別のインスタンスを開き、それぞれのインスタンスに新しいユーザ名を入れます。

ただ PeopleList コンポーネントをステージ上にドラッグし、peopleList_mc と名づけただけなので、com_test のサンプルに SimpleConnect を加えるのは大げさな気もしますが、以下のコードを加えます。

```
nc = new NetConnection();
nc.onStatus = function(info) { trace(info.code);}
peopleList_mc.close();//オプション、クリーンアップ・コール
nc.connect("rtmp:/com_test","Jane");
peopleList_mc.connect(nc);
```

しかし SimpleConnect は、作成するアプリケーションが大きくなればなるほど、使い勝手がよくなっていきます。新たなコミュニケーション・コンポーネントを加えるごとに、SimpleConnect コンポーネントのプロパティ・インスペクタで、Communication Component テキスト・ボックスをダブル・クリックして開き、新しいコンポーネントのインスタンス名を加えるだけでよいのです。

SimpleConnect コンポーネントを使用しない場合

SimpleConnect コンポーネントを使用しないなら、コミュニケーション・コンポーネントを正しく機能させるためには、サーバー・サイドの application.onConnect メソッドに以下のコードを加え、サーバーにユーザ名を登録しなければなりません。

```
GFrameworkdFC.getClientGlobals(newClient).username = newUserName;
```

サーバー・サイド・コミュニケーション ActionScript 辞書に記述されているように、onConnect メソッドを書いた時は常に新しいユーザの接続を明確に許可しなければなりません。

SimpleConnect コンポーネントを使用しない場合は、main.asc ファイルは以下のコードを含んでいる必要があります。

```
load("components.asc");
application.onConnect = function(newClient,newUserName)
{
    gFrameworkdFC.getClientGlobals(newClient).username = newUserName;
    application.acceptConnection(newClient);
}
```

SimpleConnect コンポーネントを使用する場合

SimpleConnect コンポーネントを使用する場合は、以下のコードを main.asc ファイルに書きます。

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

コンポーネントを使用して、クライアント-サーバー機能と最小のサーバー・サイド・スクリプトを持った簡単なアプリケーションを作成しました。もちろん、サーバー・サイド・スクリプトを使って、コミュニケーション・コンポーネントが提供する機能を拡張したり、また自作のコンポーネントを作成できます。自作のスクリプトを書き始める前に、23ページのチャプター2にある”Flash Communication Serverの構造概略”で、概念的な情報を通読してください。ここでは、Flash Communication Server開発者に有効な、オブジェクトについての重要な概略が説明されています。

コンポーネントを使った上級の開発

コンポーネントを使って楽々とコミュニケーション・アプリケーションが作成できるようになると、UIデザインを修正してコンポーネントに柔軟性を持たせたり、自作のコンポーネントを開発したり、APIを使ってルーカー・モードを可能にしたりできます。リスキニングやルーカー・モードの詳細は、このチャプター内各コンポーネントの説明の中で述べています。

コンポーネントのリスキニング

Flash ドキュメントにコンポーネントを加えると、ライブラリ・パネルに Commucation Components ディレクトリが加わります。このディレクトリには FlashCom UI Componets というサブディレクトリが含まれています。このサブディレクトリの中には、デザインを修正できる、UI 要素を含むそれぞれのコンポーネント用のスキン・サブディレクトリがあります。本ドキュメントにあるそれぞれのコンポーネントの記述には、コンポーネント・スキンの場所を示した “リスキニング” セクションも含まれています。スキンを編集しコンポーネントの見た目を変えることができます。ステージ上でコンポーネントを選択したり、右クリックしたり、「同じ位置で編集」を選択することで、コンポーネントのインスタンスのスキンを変更できます。またライブラリ・パレットのディレクトリ内でダブルクリックすることで、コンポーネント・スキンを編集することもできます。

自作コンポーネントの開発

上級開発者は Flash コミュニケーション・ムービー・クリップとサーバー・サイド・コードを作成することで、コミュニケーション・コンポーネントを作成することができます。クライアント・サイド部は Flash ムービー・クリップとして実行され、Flash クライアント環境で動作します。サーバー・サイド部は、コミュニケーション・コンポーネント基本クラスである FCComponent を拡張した ActionScript クラス(ファンクション・オブジェクト)です。コンポーネントの作成方法に関する情報は、Macromedia Flash MX web サイトを、コミュニケーション・コンポーネントの作成方法に関する情報は、Macromedia Designer & Developer Center

(http://www.macromedia.com/go/fcs_components)をご覧ください。

lurker モードの理解

コンポーネントの中には、接続した他のユーザーからユーザーを見えなくするユーザー・モードを提供するものもあります。このモードは *lurker mode* と呼ばれ、例えば司会者や参加者が必要な場面では有用に使えます。lurker モードでは、他のユーザに必ずしも知られず、全機能にアクセスする必要のないユーザが存在します。例えば Chat コンポーネントは、このルーカー(見ているだけの人)に、他のユーザの文字チャットを見せるが、メッセージを送ることは許可しない lurker mode をサポートしています。別のコンポーネントはまた別の lurker mode 機能を提供します。本ドキュメントに記述されている機能は、作成するアプリケーションが SimpleConnect を使用すると仮定し、ユーザがイグジットせずアプリケーションからログアウトすることを拒否します。コミュニケーション・コンポーネントに関する記述はどれも、lurker mode が可能かどうか、その機能について書かれています。

コンポーネント・リスト

Flash Communication Server には 16 のコンポーネントが内蔵されています。

SimpleConnect コンポーネント

ConnectionLight コンポーネント

VideoPlayback コンポーネント

VideoRecord コンポーネント

Chat コンポーネント

PeopleList コンポーネント

UserColor コンポーネント

Cursor コンポーネント

Whiteboard コンポーネント

PresentationText コンポーネント

PresentationSWF コンポーネント

AVPresence コンポーネント

AudioConference コンポーネント

VideoConference コンポーネント

SetBandwidth コンポーネント

RoomList コンポーネント

SimpleConnect コンポーネント

このコンポーネントは、Macromedia Flash Communication Server に接続したユーザや、サーバー・アプリケーションに接続したある特定のコンポーネントを制御します。またユーザ名を表示したり、ユーザが名前を変えるユーザ・インターフェイスも持っています。

SimpleConnect コンポーネントは、ユーザが初めてログインしたときに作成されるローカルの共有オブジェクトからデータを取得することで、ユーザを記録します。ローカルの共有オブジェクトを使用することで、接続と接続とを関連させてつなぎ合わせたり、ユーザがエグジットすることなくアプリケーションからログアウトすることを避けることができます。

このコンポーネントを使用するには、ムービーにドラッグし、プロパティ・インスペクタに表示されるコンポーネント・パラメータを編集します。rtml:/test といったアプリケーション・ディレクトリを準備する必要があり、自動的に接続させる他の全てのコミュニケーション・コンポーネントのリストを入力します。

このコンポーネントは、他の全てのコミュニケーション・コンポーネントと併用できるよう設計されており、NetConnection オブジェクトの管理を簡単なものになっています。ActionScript を少し書き加えた、もしくは全く書き加えないコミュニケーション・コンポーネントを使用して、新しいコミュニケーション・アプリケーションを構築することができます。

Tip: Simple Connect コンポーネントで登録したいコンポーネントは、Simple Connect コンポーネントが初期化するとき、ステージ上になければなりません。同じフレーム内でコンポーネントを見せたくない場合は、ステージ上にコンポーネントを置いて、Simple Connect コンポーネントと同じフレーム内で、見えなく(_visible = false)します。接続したいコン

ポーネントがステージ上にある場合は、コードに `componet.connect();`を明快に加えます。

SimpleConnect コンポーネントを使用する

以下はこのコンポーネントの使用例です。

映像会議：このコンポーネントと VideoConference コンポーネントを併用して、多くの機能を持ったアプリケーションを簡単に構築できます。

カスタム・コミュニケーション・コンポーネンツ：connect、close、setUsername などのメソッドを使用し、コミュニケーション・コンポーネントを作成します。SimpleConnect コンポーネントを使用して、ネットワーク接続を管理できます。

プロパティ・インスペクタ・パラメータ

名前	説明
Application Directory	ストリング；サーバーに接続する際使用するアプリケーション・ディレクトリの場所を特定するコンポーネント・パラメータ。
Communication Components	Array；FCSimpleConnect が接続する他のコミュニケーション・コンポーネントのリストを含むコンポーネント・パラメータ。

SetBandwidth コンポーネントのリスキニング

このコンポーネントのアセットは、ライブラリ・パネルの Core Assets-Developer Only¥Assets ディレクトリにあります。

ConnectionLight コンポーネント

このコンポーネントは、クライアントの接続状況を視覚的に戻す機能を提供します。ライトは、接続すれば緑、切断すれば赤、接続のレイテンシー(ネットワークを経由してサーバーからデータを送るのにかかる時間)が高すぎると赤になります。ライトは色を変える機能のほか、クリックするとトグル・ボタンとして、接続に関する詳細な情報をボックスの中に表示します。情報とは、レイテンシー・レート、アップロード、ダウンロードのレートです。

ConnectionLight コンポーネントは、SimpleConnection コンポーネントと併用すればスクリプティングの必要はなくなります。

ConnectionLight コンポーネントを使用する

このシンプルで使い勝手の良いコンポーネントは、どんなアプリケーションにでも使用でき、コミュニケーション・アプリケーションの標準的なコンポーネントといえます。

ConnectionLight コンポーネントの使い方は簡単で、ステージにドラッグしてサーバー・スクリプトで components.asc がロードされているかを確認するだけです。このコンポーネントを SimpleConnection コンポーネントと併用すれば、作業は終了です。SimpleConnection コンポーネントを使わない場合は、以下のクライアント・サイド ActionScript を記述する必要があります。

```
light_mc.connect(nc);
```

この ActionScript の中で、light_mc はムービー内にドラッグされたライトのインスタンス名で、nc は NetConnection オブジェクトです。

アプリケーションでコンポーネントを使う：

1 Flash Communication Server が起動していることを確認する。

2 Flash Communication Server MX のアプリケーション・ディレクトリ(通常は%flashcom%applications)にアプリケーション・ディレクトリを作成し、connect_test の名前をつける。

3 connect_test 内に main.asc の名前で新しいファイルを作成し、以下のコードを記述する。

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

Note:他のアプリケーション・ディレクトリからこのアプリケーション・ディレクトリへ、main.asc ファイルをコピーして使用することもできます。

4 Flash MX でコミュニケーション・コンポーネント・パネルが表示されていない場合は、ウィンドウ・メニュー>コンポーネントを選択し、コンポーネント・ポップアップ・メニューからコミュニケーション・コンポーネントを選択する。

5 コンポーネント・パネルから ConnectionLight コンポーネントをステージ上にドラッグする。プロパティ・インスペクタで light_mc のインスタンス名をつける。他のパラメータについては、29ページの"ConnectionLight コンポーネント・プロパティ・インスペクタ"をご覧ください。

6 サーバーに接続するには、タイムラインの一番めのキーフレームを選択し、そのフレームのアクション・パネルで以下のコードを記述する。

```
nc = new NetConnection();
```

```
nc.connect("rtmp://connect_test");
```

7 サーバーへ light_mc ムービー・クリップを接続させるには、以下のように、nc NetConnection インスタンスを使用する。

```
light_mc.connect(nc);
```

8 このファイルを connect_test fla 名で保存、パブリッシュする。

9 アプリケーション・ディレクトリで SWF ファイルを開くか、Flash MX オーサリング環境の場合は、制御>ムービープレビューを選択する。緑のボタンをクリックする。

すると以下のように表示される。



ConnectionLight コンポーネントをリスキニングする

このコンポーネントのアセットは、ライブラリ・パネルの Core Assets-Developer Only%ConnectionLight Assets ディレクトリにあります。

ConnectionLight コンポーネント・プロパティ・インスペクタ

プロパティ

解説

measurementInterval このプロパティは、帯域幅とレイテンシーの値をどれくらいの間隔で測定するかを制御する。デフォルト値は2秒。

latencyThreshold このプロパティは、ライトが黄色に変わるレイテンシーを決定する。デフォルト値は0.1秒。よって、接続レイテンシーが100ミリ秒より大きくなればライトは黄色に変わり、レイテンシーがしきい値以下に落ちるまでそのままのままでいる。

関係コンポーネント

“SimpleConnect コンポーネント” 64ページ。

FCConnectionLight オブジェクト

FCConnectionLight API は、サーバーに接続したりクリーンアップするオブジェクトに、クライアント・サイドとサーバー・サイドでの機能を提供します。またサーバーからの送信データを受け取る時間の長さを設定することもできます。

FCConnectionLight オブジェクトのプロパティ・サマリー

プロパティ

解説

FCConnectionLight.measurementInterval 帯域幅とレイテンシーの値をどれくらいの間隔で測定するかを制御する。

FCConnectionLight.latencyThreshold ライトが黄色に変わるレイテンシーを決定する。

FCConnectionLight オブジェクトのメソッド・サマリー

メソッド

解説

FCConnectionLight.connect 接続状況の監視を開始する。

FCConnectionLight.close 接続の監視をやめ、クリーンアップする。

FCConnectionLight.measurementInterval

利用

Flash Player 6

Flash Communication Server MX

使い方

```
connectLight_mc.measurementInterval = maxInterval
```

説明

プロパティ；帯域幅とレイテンシーの値をどれくらいの間隔で測定するかを制御する。これは取得、設定できるプロパティ。値 *maxInterval* は、帯域幅とレイテンシーの値をどれくらいの間隔で測定するかを決定する数値。デフォルト値は2秒。

FCConnectionLight.latencyThreshold

利用

Flash Player 6

Flash Communication Server MX

使い方

```
connectLight_mc.latencyThreshold = maxLatency
```

説明

プロパティ；ライトが黄色に変わるレイテンシーを決定する。これは取得、設定できるプロパティ。値 *maxLatency* は、レイテンシーが高くなると黄色に変わるのにかかる時間を決定する。デフォルト値は 0.1 秒。よって、接続レイテンシーが 100 ミリ秒より大きくなればライトは黄色に変わり、レイテンシーがしきい値以下に落ちるまでそのままのまである。

FCConnectionLight.connect

利用

Flash Player 6

Flash Communication Server MX

使い方

```
connectLight_mc.connect(nc)
```

パラメータ

nc アクティブな NetConnection オブジェクト。

戻り値

なし

説明

メソッド；接続状況の監視を開始する。

FCConnectionLight.close

利用

Flash Player 6

Flash Communication Server MX

使い方

```
connectLight_mc.close()
```

パラメータ

なし

戻り値

なし

説明

メソッド；クライアント上でこのコンポーネントが使用しているアセットを解放し、クリーンアップする。その後は接続監視をしない。

VideoPlayback コンポーネント

VideoPlayback コンポーネントによって、バッファされた音声映像ストリームの再生が可能になります。ムービーを一時停止、再生させたり、再生ヘッドをドラッグして好きな場所に飛んだり、また音声レベル設定もできます。

VideoPlayback コンポーネントは、SimpleConnect コンポーネントと併用すればスクリプティングの必要はなくなりま
す。VideoPlayback コンポーネントは、streamName 変数内に特定したストリームをロードし、playStream メソッド
によって再生できます。

VideoPlayback コンポーネントを使用する

このコンポーネントは、ストリームを記録したビデオを再生するツールとして使用できます。以下はこのコンポーネン
トの2つの使用例です。

ムービー再生：これまでに記録したコンテンツを再生します。

ビデオ・ノート：先に訪れたユーザが残したメッセージを再生します。

アプリケーション内でこのコンポーネントを使用するには、72ページの“VideoRecord コンポーネントを使う”をご覧
ください。

VideoPlayback コンポーネント・プロパティ・インスペクタ

名前	変数	説明
Default Stream Name	streamName	ストリング；デフォルト値は video。ストリーム名を特定し てロードし再生する。
Buffer Time	bufferTime	数値；デフォルト値は 2。再生時にバッファするビデオ・ データの毎秒の長さを特定する。

VideoConference コンポーネントのリスキニング

このコンポーネントの資産は、ライブラリ・パネルの Core Assets-Developer Only¥VideoPlayback Assets デイレ
クトリにあります。

関係するコンポーネント

“SimpleConnect”コンポーネント 64 ページ、 “VideoRecord”コンポーネント 72 ページ

FCVideoPlayback オブジェクト

FCVideoPlayback オブジェクトは、サーバーに接続したりクリーンアップしたりするオブジェクトに、クライアント・
サイドやサーバー・サイドでの機能を提供します。

FCVideoPlayback オブジェクトのメソッド・サマリー

メソッド	説明
FCVideoPlayback.connect	クライアント・サイドとサーバ・サイドで VideoPlayback コンポーネントが必要と するアセットをセットアップする。
FCVideoPlayback.close	クリーン・アップし、通常のマウス・ポインタに戻す(Mouse.show)

FCVideoPlayback.connect

利用

Flash Player 6

Flash Communication Server MX

使い方

`vidPlayback_mc.connect(nc)`

パラメータ

nc アクティブな NetConnection オブジェクト

戻り値

なし

説明

メソッド；クライアント・サイドとサーバ・サイドで VideoPlayback コンポーネントが必要とするアセットをセットアップする。

FCVideoPlayback.close

利用

Flash Player 6

Flash Communication Server MX

使い方

`vidPlayback_mc.close()`

パラメータ

なし

戻り値

なし

説明

メソッド；このクライアント上でこのコンポーネントが使用していたアセットを解放することでコンポーネントを閉じ、クリーン・アップする。

VideoRecord コンポーネント

VideoRecord コンポーネントは、バッファされたストリームをサーバーに記録します。カメラとマイクの出力結果が記録されます。

VideoRecord コンポーネントを使用する

このコンポーネントは、サーバーにストリームを保存するビデオ・レコーダーとして使用できます。以下はこのコンポーネントを使用した2つの例です。

ムービー・レコーダー：このコンポーネントを使ってサーバーにビデオをキャプチャーし、後で再生する。

ビデオ・ノート：他のユーザがメッセージを記録し、後で見る。

アプリケーションで VideoRecord コンポーネントを使用する

1 Flash Communication Server が起動していることを確認する

2 Flash Communication Server MX のアプリケーション・ディレクトリ(通常は`flashcom\applications`)にアプリケーション・ディレクトリを作成し、`recplay_test`の名前をつける。

3 ステージ上に `videoRecord` コンポーネントをドラッグし、インスタンス名 `record_mc` をつける。他のパラメータについては、74 ページの `videoRecord` コンポーネント・プロパティ・インスペクタをご覧ください。

4 ステージ上に `videoPlayback` コンポーネントをドラッグし、インスタンス名 `playback_mc` をつける。他のパラメータについては、70 ページの `videoPlayback` コンポーネント・プロパティ・インスペクタをご覧ください。

5 ステージ上に `SimpleConnect` コンポーネントをドラッグする。

6 `SimpleCpnnection` 用のプロパティ・インスペクタで以下のパラメータを入力する。

Application Directory テキスト・ボックスで、ステップ 2 で作成したアプリケーションの URI、例えば、`rtmp:/recplay_test` を入力する。

Communication Components テキスト・ボックスをダブル・クリックする。現れる値ダイアログ・ボックスで、+ 印をクリックし、再生と記録のコンポーネントのインスタンス名にそれぞれ `playback_mc`、`record_mc` と入力し、OK をクリックする。

これらの値を与えることで、`SimpleConnect` コンポーネントは `videoPlayback` コンポーネントと `videoRecord` コンポーネントの接続を制御できる。`SimpleConnect` コンポーネントがサーバーへの接続に成功したら、他のコンポーネントは自動的にサーバーに接続する

7 アプリケーション・ディレクトリ内に `recplay_test` 名で保存、パブリッシュする。

8 このアプリケーション・ディレクトリ内に `main.asc` の名前で新しいファイルを作成し、以下のコードを記述する。

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

Note:他のアプリケーション・ディレクトリからこのアプリケーション・ディレクトリへ、`main.asc` ファイルをコピーして使用することもできます。

9 アプリケーション・ディレクトリ内にある SWF ファイルを開くか、または Flash MX オーサリング環境では、制御>ムービープレビューを選択しログインする。

VideoRecord コンポーネントの Record ボタンをクリックし、コンポーネントの Default Stream Name プロパティで定義されたデフォルトのストリーム名 video を記録する。カメラとマイクの出力は、下記のように VideoRecord コンポーネント上の右矢印の Record ボタンがクリックされたら記録される。



再生する video ストリームは、playback コンポーネントの 1 番目のパラメータで定義される。記録を止めた後、playback コンポーネント上の右矢印の Playback ボタンをクリックすると、下記のように記録されたビデオ・ストリームが再生される。



VideoRecord コンポーネント・プロパティ・インスペクタ

名前	変数	説明
Default Stream Name	streamName	ストリング；デフォルト値は video。最初にロードし再生するストリーム名を特定する。
Default Stettings	[setLow	ストリング；デフォルト値は setHigh。コンポーネントが初期化されるとき、使用するカメラの設定を指定する。SetLow、setMed、setHigh の 3 種類の設定がある。
Buffer Time	bufferTime	数値；デフォルト値は 10。録画でバッファするビデオ・データの毎秒の長さを指定する。
Low Quality Setting	setLow	オブジェクト；カメラ設定で使用する以下の値を持つ。 幅、水平値 160 ピクセル 高さ、垂直値 120 ピクセル fps、毎秒 10 ビデオ・フレーム 帯域幅、使用する最大の帯域幅、0(0は無制限を示す) 品質、ビデオの圧縮品質、50(0は無制限を示す) キー、キーフレームの間隔、10キーフレーム レート、キロヘルツ単位のオーディオ品質、11KHz
Medium Quality Setting	setMed	オブジェクト；カメラ設定で使用する以下の値を持つ。 幅、水平値 240 ピクセル 高さ、垂直値 180 ピクセル fps、毎秒 12 ビデオ・フレーム 帯域幅、使用する最大の帯域幅、0(0は無制限を示す) 品質、ビデオの圧縮品質、80(0は無制限を示す) キー、キーフレームの間隔、12キーフレーム レート、キロヘルツ単位のオーディオ品質、22KHz
High Quality Setting	setHigh	オブジェクト；カメラ設定で使用する以下の値を持つ。 幅、水平値 320 ピクセル 高さ、垂直値 240 ピクセル fps、毎秒 15 ビデオ・フレーム 帯域幅、使用する最大の帯域幅、0(0は無制限を示す) 品質、ビデオの圧縮品質、90(0は無制限を示す) キー、キーフレームの間隔、5キーフレーム レート、キロヘルツ単位のオーディオ品質、44KHz

VideoRecord コンポーネントのリスキニング

このコンポーネントのアセットは、ライブラリ・パネルの Core Assets-Developer Only¥VideoRecord Assets ディレクトリにあります。

関係するコンポーネント

“SimpleConnect”コンポーネント 64 ページ、“VideoPlayback”コンポーネント 70 ページ

FCVideoRecord オブジェクト

VideoRecord コンポーネントは、SimpleConnect コンポーネントと併用すればスクリプティングの必要はなくなります。

streamName プロパティを設定しない場合は、コンポーネントはデフォルトのストリーム名で録画します。

FCVideoRecord オブジェクトのメソッド・サマリー

メソッド	説明
FCVideoRecord.connect	クライアント・サイドとサーバー・サイドで VideoRecord コンポーネントが必要とするアセットをセットアップする。
FCVideoRecord.close	クリーン・アップし、通常のマウス・ポインタに戻す(Mouse.show)

FCVideoRecord.connect

利用

Flash Player 6

Flash Communication Server MX

使い方

`vidRecord_mc.connect(nc)`

パラメータ

`nc` アクティブな NetConnection オブジェクト

戻り値

なし

説明

メソッド；クライアント・サイドとサーバー・サイドで VideoRecord コンポーネントが必要とするアセットをセットアップする。

FCVideoRecord.close

利用

Flash Player 6

Flash Communication Server MX

使い方

`vidRecord_mc.close()`

パラメータ

なし

戻り値

なし

説明

メソッド；このクライアント上のこのコンポーネントが使用していたアセットを解放することで、コンポーネントを閉じ、クリーン・アップする。

Chat コンポーネント

このコンポーネントは一般的な文字チャットのウィンドウを持ち、lurker mode をサポートします。SimpleConnect コンポーネントと一緒に使用すれば、ログインしないユーザがテキスト・ボックスに入力しても Send ボタンを押しても、lurker mode により表示されません。見てだけのユーザも他の人の文字チャットを見ることはできます。ユーザ名を入力してログインすると(SimpleConnect コンポーネントを使用するか、1 番目のパラメータとして、nc.connect をコールするか)、ユーザは文字を送信できます。UserColor コンポーネントも使用すれば、ユーザの文字が個別に色分けされます。Chat コンポーネントは SimpleConnect コンポーネントを併用すればスクリプティングの必要がなくなります。

このコンポーネントを UserColor コンポーネントと一緒に使用することもできます。ユーザはいつでも新しい色を選択でき、サーバーサイドではグローバルな色として更新され、全てのクライアントでも色は更新されます。

Chat コンポーネントを使用する

このコンポーネントは、チャットルーム・アプリケーション作成のために使用したり、以下に記すように大きなアプリケーションの内部で動作するコンポーネントとしても使用できます。

一般的なチャットルーム:このコンポーネントを UserColor コンポーネントや PeopleList コンポーネント、SimpleConnect コンポーネントと併用してチャットルームを作成できます。

あらゆるアプリケーションにおいて使いやすい:コミュニケーション・アプリケーションの中に Chat コンポーネントを含めることは有益です。文字のコミュニケーションは扱いやすく、例えばユーザのコンピュータの接続環境が遅い場合など、音声・映像でのコミュニケーションがしづらい場合の代案になります。

Chat コンポーネントを使うには

1 Flash Communication Server が起動していることを確認する

2 Flash Communication Server MX のアプリケーション・ディレクトリ(通常は`flashcom/applications`)にアプリケーション・ディレクトリを作成し、chat_test の名前をつける。

3 アプリケーション・ディレクトリ内に main.asc の名前で新しいファイルを作成し、以下のコードを記述する。

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

Note:他のアプリケーション・ディレクトリからこのアプリケーション・ディレクトリへ、main.asc ファイルをコピーして使用することもできます。

4 Flash MX 画面で、ステージ上に Chat コンポーネントをドラッグし、プロパティ・インスペクタで、chat_mc のインスタンス名をつける。

5 ステージ上に UserColor コンポーネントをドラッグし、プロパティ・インスペクタで、color_mc のインスタンス名をつける。

6 ステージ上に SimpleConnect コンポーネントをドラッグする。

7 SimpleCpnnection 用のプロパティ・インスペクタで以下のパラメータを入力する。

Application Directory テキスト・ボックスで、ステップ 2 で作成したアプリケーションの URI、例えば、rtmp:/chat_test を入力する。

Communication Components テキスト・ボックスをダブル・クリックする。現れる値ダイアログ・ボックスで、+印をクリックし、Chat コンポーネントのインスタンス名 chat_mc を入力、さらに+印をクリックし、UserColor コンポーネントのインスタンス名 color_mc を入力し、OK をクリックする。

これらの値を与えることで、SimpleConnect コンポーネントは Chat コンポーネントと UserColor コンポーネントの接続を制御できる。SimpleConnect コンポーネントがサーバーへの接続に成功したら、Chat と UserColor コンポーネントは自動的にサーバーに接続する。

8 chat_test のファイル名でアプリケーション・ディレクトリ内に保存、パブリッシュする。

9 アプリケーション・ディレクトリ内にある SWF ファイルを開くか、または Flash MX オーサリング環境では、制御 >ムービープレビューを選択しログインする。

多くの SWF ファイル・インスタンスを開くことができます。違うユーザとしてログインしたり、下記に示すように違う文字カラーを選ぶこともできます。



Chat コンポーネントのリスキニング

Chat コンポーネントのアセットは、ライブラリ・パネルの Core Assets-Developer Only\Chat Assets ディレクトリにあります。

関係するコンポーネント

“SimpleConnect コンポーネント” 6 4 ページ、“UserColor コンポーネント” 6 5 ページ、“PeopleList コンポーネント” 3 4 ページ。

FCChat オブジェクト

FCChat API は、サーバーに接続したりクリーンアップするオブジェクトに、クライアント・サイドでの機能を提供します。またプログラミングで、クライアントからチャットの履歴を消すこともできます。このコンポーネントのサーバー・サイド API については 2 5 P の“FCChat サーバー・サイド・オブジェクト”をご覧ください。

FCChat オブジェクトのメソッド・サマリー

メソッド	説明
FCChat.connect	クライアント・サイド、サーバー・サイドで、コンポーネントの必要とするアセットを全てセット・アップし、チャット履歴を取得する。
FCChat.setUsername	アクティブなユーザのリストを現わすためユーザ名を取得する。
FCChat.clearHistory	チャット履歴を消去し、全てのクライアント上で履歴を更新する。
FCChat.close	クリーンアップする。

FCChat.connect

利用

Flash Player 6

Flash Communication Server MX

使い方

`chat_mc.connect(nc)`

パラメータ

`nc` アクティブな NetConnection オブジェクト

戻り値

なし

説明

メソッド ; クライアント・サイド、サーバー・サイドで、コンポーネントの必要とするアセットを全てセットアップし、チャット履歴を取得する。

FCChat.setUsername

利用

Flash Player 6

Flash Communication Server MX

使い方

`chat_mc.setUsername(newName)`

パラメータ

`newName` スtring

`newName` パラメータが null か undefined でない場合は入力テキスト・ボックスと Send ボタンを表示する。

`newName` パラメータが null の場合は、何もしない(lurker mode)

戻り値

なし

説明

メソッド ; アクティブなユーザのリストに表示するためにユーザ名を取得する

FCChat.clearHistory

利用

Flash Player 6

Flash Communication Server MX

使い方

`chat_mc.clearHistory()`

パラメータ

なし

戻り値

なし

説明

メソッド ; チャット履歴を消去し、全てのクライアント上で履歴を更新する。この命令は、クライアントによるクリアを、サーバー・サイドの設定(デフォルトでは実行可能)が許可して初めて成功する。さらに詳しい情報は 27 P の “ FCChat.allowClear ” をご覧ください。

FCChat.close

利用

Flash Player 6

Flash Communication Server MX

使い方

`chat_mc.close()`

パラメータ

なし

戻り値

なし

説明

メソッド ; クライアント上でコンポーネントが使用しているアセットを解放しクリーンアップする。

FCChat サーバー・サイド オブジェクト

FCChat サーバー・サイド API は、履歴プロパティの接続と履歴のクリアを可能にするサーバー・サイドの機能を提供します。

FCChat サーバー・オブジェクトのメソッド・サマリー

メソッド

説明

FCChat.clearHistory 履歴をクリアする。

FCChat サーバー・オブジェクトのプロパティ・サマリー

以下のサーバー・サイド・プロパティは、コンポーネントのふるまいを修正するために設定することができます。

プロパティ

説明

FCChat.histlen チャット履歴の可能な大きさを定義する。

FCChat.persist チャット履歴がアプリケーションの実行中に持続するかどうかを定義する。

FCChat.allowClear チャット履歴がクライアントからクリアできるかどうかを定義する。

FCChat.histlen

利用

Flash Player 6

Flash Communication Server MX

使い方

```
FCChat.prototype.histlen = maxHistory;
```

説明

プロパティ； チャット履歴の可能な大きさを定義する。取得、設定できるプロパティ。値 *maxHistory* は、サーバーが保有する文字メッセージの最大長を示す数値。デフォルト値は 250。

FCChat.persist

利用

Flash Player 6

Flash Communication Server MX

使い方

```
FCChat.prototype.persist = true;
```

説明

プロパティ； チャット履歴がアプリケーションのある間ずっと保持されるかどうかを定義する。履歴はアプリケーションがリスタートしても持続する。これは取得、設定できるプロパティ。値 *true* と *false* は、サーバーが履歴を保存するかどうかを決める Boolean 値。デフォルト値は *true*。

FCChat.allowClear

利用

Flash Player 6

Flash Communication Server MX

使い方

```
FCChat.prototype.allowClear = true;
```

説明

プロパティ； チャット履歴がクライアントからクリアできるかどうかを定義する。これは取得、設定できるプロパティ。値 true と false はユーザが履歴をクリアできるかどうかを決定する Boolean 値。デフォルト値は true。

FCChat.clearHistory

利用

Flash Player 6

Flash Communication Server MX

使い方

```
FCChat.prototype.clearHistory()
```

パラメータ

なし

説明

メソッド； 履歴をクリアする。クライアント・サイドでなくサーバー・サイドでこのメソッドをコールすると成功する。

PeopleList コンポーネント

このコンポーネントは、コミュニケーション・アプリケーションに接続しているユーザのリストを表示します。ユーザ名を入力したユーザだけがリストに表示されます。

このコンポーネントは、SimpleConnect コンポーネントと併用すれば、スクリプティングの必要はなくなります。

PeopleList コンポーネントを使用する

このコンポーネントはコミュニケーション・アプリケーションの中で標準的なもので、現在ログインしているユーザのリストを提供します。

以下はこのコンポーネントの使用法を 2 つ紹介したものです。

チャット； PeopleList コンポーネントと Chat、SimpleConnect コンポーネントを合わせて使用することで、チャットルーム・アプリケーションの基本部分を作成できます。

仮想会議； このコンポーネントを使用して、仮想会議部屋を作成し、現在会議に参加している全ユーザを表示できます。

PeopleList コンポーネントのリスキニング

このコンポーネントのアセットは、ライブラリ・パネルの Core Assets-Developer Only\PeopleList Assets ディレクトリにあります。作成するアプリケーションに合わせて、ポインタ矢印やラベルの文字を編集できます。

関係するコンポーネント

“SimpleConnect コンポーネント” 6 4 p、 “Chat コンポーネント” 2 0 P。

FCPeopleList オブジェクト

FCPeopleList API は、サーバーに接続したりクリーンアップするオブジェクトに、クライアント・サイドとサーバー・サイドでの機能を提供します。またプログラミングを施せば lurker mode を設定することもできます。

変数

名前	変数	説明
Lurkers	lurkers	整数 ; 新しいユーザが接続したり切断すると必ず更新される変数。アプリケーションに接続して
		いるユーザ数と等しいが、ユーザ名は供給しない。この変数は取得できるが、設定はできない。
Uers	users	整数 ; lurkers と類似した変数。アプリケーションに接続しているユーザ数と等しく、ユーザ名を供給する値。この変数は取得できるが、設定はできない。

FCPeopleList オブジェクトのメソッド・サマリー

メソッド	説明
FCPeopleList.connect	クライアント・サイドとサーバー・サイドでコンポーネントが必要とする全てのアセットをセットアップし、サーバーが提供する名前で setUsername ファンクションをコールする。
FCPeopleList.setUsername	アクティブなユーザのリストに表示するユーザ名を設定する。
FCPeopleList.close	クリーンアップする。

FCPeopleList.connect

利用

Flash Player 6

Flash Communication Server MX

使い方

peopleList_mc.connect(nc)

パラメータ

nc アクティブな NetConnection オブジェクト

戻り値

なし

説明 ;

メソッド ; クライアント・サイドとサーバー・サイドでコンポーネントが必要とする全てのアセットをセットアップし、サーバーが提供する名前で setUsername ファンクションをコールする。

FCPeopleList.setUsername

利用

Flash Player 6

Flash Communication Server MX

使い方

peopleList_mc.setUsername(newName)

パラメータ

newName 新しいユーザ名を記述するオプションのストリング。*newName* が null または undefined でない場合、*newName* はユーザのリストに表示される。*newName* が null の場合は何も変更されず、ユーザは lurker mode に入る。

戻り値

なし

説明；

メソッド；アクティブなユーザのリストに表示されるユーザ名を設定する。

FCPeopleList.close

利用

Flash Player 6

Flash Communication Server MX

使い方

peopleList_mc.close()

パラメータ

なし

戻り値

なし

説明；

メソッド；クライアント上のこのコンポーネントが使用しているアセットを解放、クライアントのユーザ・リストからユーザを消去し、クリーン・アップする。

UserColor コンポーネント

このコンポーネントによって、ユーザは Flash Communication Server アプリケーション内にあるコンポーネントの選択した色を変更できます。ユーザがメニューから色を選択すると、例えば Chat コンポーネントのように、色を表示する他のコンポーネントが、自動的に色を変更します。

UserColor コンポーネントは、ユーザのコンピュータ上にあるローカルの共有オブジェクトに選択した色を保管します。もし、初めての訪問などで、ローカルの共有オブジェクトが見つからない場合は、リストからランダムに色が選ばれ表示されます。

UserColor コンポーネントは、SimpleConnect コンポーネントと併用すれば、スクリプティングの必要はなくなります。

UserColor コンポーネントを使用する

このシンプルで視覚的にパワフルなコンポーネントは、集中管理的ツールで、他のコンポーネントに色を加えます。

以下はこのコンポーネントの2つの使用例を示したものです。

読みやすいチャット：UserColor コンポーネントなしに Chat コンポーネントを使用すると、全ユーザの文字色は黒になります。UserColor コンポーネントを加えることで、チャットの文字は読みやすくなり、ユーザにカスタマイズ
の機会を与えることができます。

ユーザに仮想アイデンティティが必要な場合はいつでも、ユーザが特定した色はそのユーザ名のように働きます。

UserColor コンポーネントにより、多くのコーディングをする苦勞なしに、アイデンティティが拡張された感覚を味わわせることができます。

SimpleConnect、UserColor、Corsor、Chat コンポーネントを合わせて使用してみてください。ユーザは自分の名前や色、場所を表示する文字とカーソルをステージ上に見ることができます。ユーザの色はより豊かなユーザ体験をもたらします。

アプリケーションでこのコンポーネントを使用するには、20ページの“Chat コンポーネント”をご覧ください。

コンポーネントのリスキニング

このコンポーネントの色を変えたい場合は、ライブラリ・パネルでダブル・クリックし、ColorCombo ボックスをクリックして、プロパティ・インスペクタを開き、Data array 内で値を変更します。カラー・パレットは自動的に更新されます。Label array と Data array の要素数は同じにしておくよう注意してください。

関係するコンポーネント

“Chat コンポーネント”20ページ。

FCUserColor オブジェクト

FCUserColor API は、サーバーに接続したりクリーンアップしたりするオブジェクトに、クライアント・サイドとサーバー・サイドでの機能を提供します。

FCUserColor オブジェクト・メソッド・サマリー

メソッド	説明
FCUserColor.connect	保存された FCUserColor のローカル共有オブジェクトを探し、ユーザの最後の色を取得します。
FCUserColor.close	クリーン・アップします。

FCUserColor.connect

利用

Flash Player 6

Flash Communication Server MX

使い方

`userColor_mc.connect(nc)`

パラメータ

`nc` アクティブな NetConnection オブジェクト

戻り値

なし

説明；

メソッド；保存された FCUserColor のローカル共有オブジェクトを探し、ユーザの最後の色を取得する。ローカル共有

オブジェクトが存在する場合、色は選択される。そうでない場合は、そのユーザの色がランダムに選択され、ローカル共有オブジェクト内に保存される。

FCUserColor.close

利用

Flash Player 6

Flash Communication Server MX

使い方

`userColor_mc.close()`

パラメータ

なし

戻り値

なし

説明；

メソッド；このクライアント上でこのコンポーネントが使用したアセットを解放し、クリーン・アップする。

UserColor コンポーネントとコンポーネントを併用する

コンポーネントを書き、UserColor コンポーネントと併用させたいときは、以下のステップで見ている人の色を変更します。

1 下記のように、ocColorChange メソッドを記述します。

```
MyCompnentClass.prototype.onColorChange = function()  
{  
    this.setColor(gFlashCom.userprefs.color);  
}
```

gFlashCom.userprefs グロ - バル・オブジェクトは、新しい色を保持した FCUserColor コンポーネントによって更新さ

れます。

コンポーネントの this.setColor メソッドは、色の変更を処理します。onColorChange と setColor メソッドを論理的に

分けるのはよいことです。そうすることでonColorChange を呼ぶUserColor コンポーネントを使用しなくても、setColor メソッドを使用できます。

2 色も含む userprefs オブジェクトが変更されたことを知らせるため、リスナーのリストにコンポーネント・クラスを

加えます。コンポーネントの#endinitclip ステートメントの後に、以下を入力します。

```
GFlashCom.userprefs.addListener(this);
```

これらのステップで、色の変更に FCUserColor を使用すればいつでも、onColorChange メソッドがコールされます。

Cursor コンポーネント

このコンポーネントは、アプリケーションに接続しているそれぞれのユーザに、マウス・ポインタ（カーソル）を表示します。コミュニケーション・アプリケーション内でユーザがマウスを動かせば、その動きは他のユーザの画面上に反映されます。ユーザ名がそのユーザのカーソル近くに表示され、ユーザは自分用のポインタの色を選べます。

Cursor コンポーネントは、SimpleConnect コンポーネントと併用すればスクリプティングの必要はありません。

UserColor コンポーネントとも合わせて使用できます。

Note:このコンポーネントがコミュニケーション・アプリケーションの一部で、ユーザがステージ上でポインタを初めて動かすと、ポインターは見えなくなります。ユーザがログインすれば、他のログインしているユーザと同様に、そのユーザのポインタも見えるようになります。

Cursor コンポーネントを使う

このコンポーネントはアバターのシンプルな手法です。アバターとは、複数のユーザが同時操作できる仮想のステージで、ユーザに代わる象徴を意味します。このコンポーネントを使用すると、コミュニケーション・アプリケーション内のステージ上のポインタがそれぞれのユーザの場所を代わって表します。

以下はこのコンポーネントの使用法をいくつか示したものです。

教育アプリケーション：このコンポーネントを e-ラーニングのアプリケーションに組み込むことで、プレゼンテーションや討論、質問中に、全てのユーザがグラフや地図、式などを指し示すことができるようになります。

使い方の研究：例えば web サイトなどのアプリケーションにこのコンポーネントを追加することで、人々がどのようにこれを使用するか見ることができます。

新しい双方向的モデルでの視覚的ヘルプ：多くのユーザが同時操作するアプリケーションで、ステージ上の全てのマウス・ポインタの場所を明らかにすることで、他のユーザの動向をよく知ることができる。

アプリケーションでコンポーネントを使用する

1 Flash Communication Server が起動していることを確認する

2 Flash Communication Server MX のアプリケーション・ディレクトリ(通常は`flashcomapplications`)にアプリケーション・ディレクトリを作成し、`cursor_test`の名前をつける。

3 アプリケーション・ディレクトリ内に `main.asc` の名前で新しいファイルを作成し、以下のコードを記述する。

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

Note:他のアプリケーション・ディレクトリからこのアプリケーション・ディレクトリへ、`main.asc` ファイルをコピーして使用することもできます。

4 ステージ上に Chat コンポーネントをドラッグする。

5 プロパティ・インスペクタで、`cursor_mc` のインスタンス名をつける。

6 ステージ上に SimpleConnect コンポーネントをドラッグする。

7 SimpleCpnnection 用のプロパティ・インスペクタで以下のパラメータを入力する。

Application Directory テキスト・ボックスで、ステップ 2 で作成したアプリケーションの URI、例えば、`rtmp:/cursor_test` を入力する。

Communication Components テキスト・ボックスをダブル・クリックする。現れる値ダイアログ・ボックスで、+ 印をクリックし、Cursor コンポーネントのインスタンス名 `cursor_mc` を入力し、OK をクリックする。

この値を与えることで、SimpleConnect コンポーネントは Cursor コンポーネントの接続を制御できる。SimpleConnect コンポーネントがサーバーへの接続に成功したら、Cursor コンポーネントは自動的にサーバーに接続する。

8 アプリケーション・ディレクトリ内に保存、パブリッシュする。

9 アプリケーション・ディレクトリ内にある SWF ファイルを開くか、または Flash MX オーサリング環境では、制御 >ムービープレビューを選択しログインする。Auto チェック・ボックスをクリックする。

下記のようにポインタとログイン名が表示される。さらにアプリケーションのインスタンスを開くと、またポインタが現れる。



Cursor コンポーネントのリスキニング

このコンポーネントの資産は、ライブラリ・パネルの Core Assets-Developer Only¥Cursor Assets ディレクトリにあります。作成するアプリケーションに合わせて、ポインタの矢印やラベルの文字を編集できます。

関係するコンポーネント

“SimpleConnect コンポーネント” 6 4 p、 “UseColor コンポーネント” 6 5 P。

FCCursor オブジェクト

FCCursor API は、FCConectionLight API は、サーバーに接続したりスクリーンアップするオブジェクトにクライアント・サイドとサーバーサイドでの機能を提供します。またプログラミングでユーザのポインタの色を設定できます。

FCCursor オブジェクトのメソッド・サマリー

メソッド	説明
FCCursor.connect	クライアント・サイドとサーバー・サイドでコンポーネントが必要とするアセットを全てセットアップし、サーバーが提供する名前で setUsername をコールする。
FCCursor.setUsername	表示し、サーバーへ渡すユーザ名を設定する。
FCCursor.close	クリーン・アップし、Mouse.show メソッドをコールしてマウス・ポインタを通常の状態に戻す。
FCCursor.setColor	ユーザのポインタの色を変更する。

FCCursor.connect

利用

Flash Player 6

Flash Communication Server MX

使い方

cursor_mc..connect(nc)

パラメータ

nc アクティブな NetConnection オブジェクト

戻り値

なし

説明

メソッド；クライアント・サイドとサーバー・サイドでコンポーネントが必要とするアセットを全てセットアップしサーバーが提供する名前で setUsername をコールする。

FCCursor.setUsername

利用

Flash Player 6

Flash Communication Server MX

使い方

cursor_mc..setUsername(newName)

パラメータ

newName 新しいユーザ名を特定するオプションのストリング。*newName* パラメータが null または undesine でない場合、このメソッドはユーザへの新しい参照を作成し、*newName* への参照名を設定、`Mouse.hide` メソッドをコールすることで、通常のマウス・ポインタを隠す。*newName* が null の場合は、何もせずユーザは lurker mode に入る。

戻り値

なし

説明

メソッド；表示、サーバーへ渡すユーザ名を設定する。

FlashCursor.close

利用

Flash Player 6

Flash Communication Server MX

使い方

```
cursor_mc.close()
```

パラメータ

なし

戻り値

なし

説明

メソッド； クリーン・アップし、このクライアント上でコンポーネントが必要としたアセットを解放し、`Mouse.show` メソッドをコールしてマウス・ポインタを前の状態に戻す。

FlashCursor.setColor

利用

Flash Player 6

Flash Communication Server MX

使い方

```
cursor_mc.setColor(newColor)
```

パラメータ

newColor 16進法を示すストリング(例：0xFC00F3)

戻り値

なし

説明

メソッド； ユーザのポインタの色を特定の色に変更する。`Cursor` コンポーネントは、`UserColor` コンポーネントと合わせて使用することもでき、`gFlashCom.usercolor` の値の変更を監視するように作られている。この値への変更はコールされる `setColor` によるものであり、全クライアントへの色も更新される。

Whiteboard コンポーネント

このコンポーネントでは、リアルタイムの共有環境で、テキストやボックス、ラインの編集ができます。シェイプを作るにはツールを選択し、ホワイトボード領域上でクリックします。またシェイプを新しい場所にドラッグしたりテキストを編集したりできます。デリート・キーを押すと、選択されたアクティブなアイテムが削除されます。

Whiteboard コンポーネントは、SimpleConnect コンポーネントと併用するとスクリプトを書く必要がなくなります。

Whiteboard コンポーネントを使用する

このコンポーネントを使って、アイデア開発などにおける共有環境を作ることができます。以下はこのコンポーネントの3つの使用例です。

共有ホワイトボード：他のユーザにアイデアを示す

組織図：会社のおおまかな組織構成。

ワークフロー：プロジェクトの進捗状況を示す。次々要素を追加できるので、ホワイトボード上でそれを確認でき、全てのユーザが同時に見ることができる。

アプリケーションでコンポーネントを使用する

1 Flash Communication Server が起動していることを確認する

2 Flash Communication Server MX のアプリケーション・ディレクトリ(通常は`flashcomapplications`)にアプリケーション・ディレクトリを作成し、`whiteboard_test` の名前をつける。

3 ステージ上に Whiteboard コンポーネントをドラッグする。

4 プロパティ・インスペクタでインスタンス名 `whiteboard_mc`

5 ステージ上に SimpleConnect コンポーネントをドラッグする。

6 SimpleCpnnection 用のプロパティ・インスペクタで以下のパラメータを入力する。

Application Directory テキスト・ボックスで、ステップ 2 で作成したアプリケーションの URI、例えば、`rtmp:/whiteboard_test` を入力する。

Communication Components テキスト・ボックスをダブル・クリックする。現れる値ダイアログ・ボックスで、+ 印をクリックし、`whiteboard_m` と入力、OK をクリックする。

7 アプリケーション・ディレクトリ内に `white_test` 名で保存、パブリッシュする。

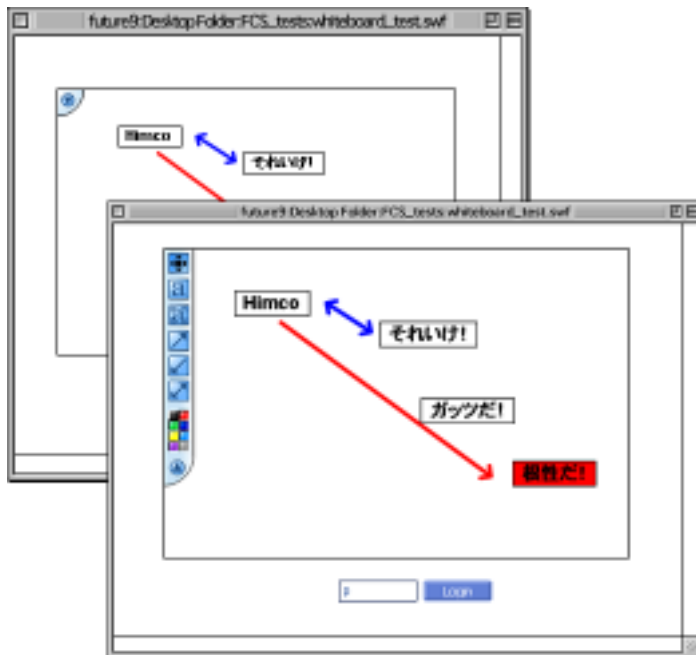
8 このアプリケーション・ディレクトリ内に `main.asc` の名前で新しいファイルを作成し、以下のコードを記述する。

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

Note:他のアプリケーション・ディレクトリからこのアプリケーション・ディレクトリへ、`main.asc` ファイルをコピーして使用することもできます。

9 アプリケーション・ディレクトリ内にある SWF ファイルを開くか、または Flash MX オーサリング環境では、制御>ムービープレビューを選択しログインする。

下図のように、左にいくつかのツールが並ぶホワイトボードが現れる。さらにアプリケーションのインスタンスを開けば、それぞれのインスタンスは他のインスタンスのシェイプやテキストを表示する。オブジェクトを選択、移動させたり、文字を打ったり、テキスト・ボックスや矢印を作成したり、色を選択したりできる。



Whiteboard コンポーネントのリスキニング

このコンポーネントの資産は、ライブラリ・パネルの Core Assets-Developer Only\Whiteboard Assets ディレクトリにあります。

関係するコンポーネント

“SimpleConnect”コンポーネント 64 ページ

FCWhiteboard オブジェクト

FCWhiteboard API は、サーバーに接続したりクリーンアップするオブジェクトに、クライアント・サイドとサーバー・サイドの機能を提供します。

FCWhiteboard オブジェクト・メソッド・サマリー

メソッド

説明

FCWhiteboard.connect	クライアント・サイドとサーバーサイドでコンポーネントが必要とする全てのアセットをセット・アップする。
FCWhiteboard.close	このクライアント上でこのコンポーネントが使用していたアセットを解放し、クリーンアップする。

FCWhiteboard.connect

利用

- Flash Player 6
- Flash Communication Server MX

使い方

`wb_mc.connect(nc)`

パラメータ

`nc` アクティブな NetConnection オブジェクト

戻り値

なし

説明

メソッド；クライアント・サイドとサーバー・サイドでコンポーネントが必要とするアセットをセットアップする。

FCWhiteboard.close

利用

- Flash Player 6
- Flash Communication Server MX

使い方

`wb_mc.close()`

パラメータ

なし

戻り値

なし

説明

メソッド；このクライアント上のこのコンポーネントが使用していたアセットを解放することで、クリーン・アップする。

PresentationText コンポーネント

このコンポーネントは、共有した文字でのプレゼンテーションを可能にします。PresentationSWFのように、PresentationText コンポーネントは、`speaker mode` または `default mode` で動作します。`speaker mode` では、ユーザはリアルタイムでスライドの編集や、現在のスライドの変更ができ、さらにスライドを加えたり消去したりできます。`default mode` では、ユーザはプレゼンテーションを見られますが、プレゼンターが達していないスライドへ進むことはできません。

PresentationText コンポーネントは、SimpleConnect コンポーネントと合わせて使えば、スクリプティングの必要はありません。

PresentationText コンポーネントを使う

このコンポーネントは、様々なプレゼンテーション・アプリケーションに追加して使用することができます。

以下はこのコンポーネントの2つの使用例です。

トレーニング・セミナー：トレーニング・アプリケーションで、メモをとったり、サンプルの概略説明に用います。

ミーティング・ノート：ミーティング・アプリケーションで、ユーザを現在の討議事項に注目させます。

以下の例では、2つのクライアント・ファイルを作成します。PresentText_test_speaker はプレゼンテーションのスライドや概略を制御するプレゼンター用のもので、presentText_test は、プレゼンターとまらない参加者が見るためのものです。これらのクライアントは両方とも同じアプリケーション、presentText_test に向かいますが、speaker mode のクライアントだけがプレゼンターになることができます。

話者アプリケーションで PresentationText コンポーネントを使用する

1 Flash Communication Server が起動していることを確認する

2 Flash Communication Server MX のアプリケーション・ディレクトリ(通常は`flashcom` applications)にアプリケーション・ディレクトリを作成し、presentText_test の名前をつける。

3 アプリケーション・ディレクトリ内に main.asc の名前で新しいファイルを作成し、以下のコードを記述する。

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

Note:他のアプリケーション・ディレクトリからこのアプリケーション・ディレクトリへ、main.asc ファイルをコピーして使用することもできます。

4 Flash MX で、ステージ上に presentationText コンポーネントをドラッグし、プロパティ・インスペクタで、presentText_mc のインスタンス名をつける。

Note:プロパティ・インスペクタでの唯一のプロパティは、speaker mode。デフォルトでの設定は true。これによりクライアントはプレゼンターになることができる。

5 ステージ上に SimpleConnect コンポーネントをドラッグする。

6 SimpleCpnnection 用のプロパティ・インスペクタで以下のパラメータを入力する。

Application Directory テキスト・ボックスで、ステップ 2 で作成したアプリケーションの URI、例えば、`rtmp:/presentText_test` を入力する。

Communication Components テキスト・ボックスをダブル・クリックする。現れる値ダイアログ・ボックスで、+ 印をクリックし、PresentationText コンポーネントのインスタンス名 presentText_mc を入力し、OK をクリックする。

この値を与えることで、SimpleConnect コンポーネントは PresentationText コンポーネントの接続を制御できる。SimpleConnect コンポーネントがサーバーへの接続に成功したら、PresentationText コンポーネントは自動的にサーバーに接続する。

7 アプリケーション・ディレクトリ内に presentText_test_speaker.fla として保存、パブリッシュする。

視聴者アプリケーションで Presentation コンポーネントを使用する

- 1 PresentText_test_speaker.fla をコピーしファイル名を presentText_fla にする。
- 2 presentText_fla 内で、PresentationText コンポーネントを選択し、プロパティ・インスペクタで speaker mode のパラメータを false に変更する。
- 3 presentText_test.fla を保存する。

両方のモードで PresentationText コンポーネントをテストする

1 アプリケーション・ディレクトリで presentText_test_speaker.swf を開くか、Flash MX オーサリング環境では、制御>ムービープレビューを選択し、ログインする。

2 アプリケーション・ディレクトリで presentText_test.swf を開くか、Flash MX オーサリング環境では、制御>ムービープレビューを選択し、別のユーザ名でログインする。

presentText_test_speaker.swf では、ユーザはプレゼンターとなり、プレゼンテーション中新しいスライドを作成したり、先へ進んだり戻ったりできる。またプレゼンテーションのアウトライン情報も見ることができる。

presentText_test.swf では、ユーザは視聴者となる。スライドを見ることはできるが、プレゼンターがスライドを進めるようにしか見ることはできない。

以下は、後ろにあるのがプレゼンター・クライアント・アプリケーションで、前にあるのが視聴者プレゼンテーション。



PresentationText コンポーネントのリスキニング

このコンポーネントの資産は、ライブラリ・パネルの Core Assets-Developer Only¥PresentationText Assets ディレクトリにあります。

関係するコンポーネント

“SimpleConnect コンポーネント”

FCPresentationText オブジェクト

FCPresentationText API は、サーバーに接続したりクリーンアップしたりするオブジェクトにクライアント・サイドとサーバーサイドでの機能を提供します。またプログラミングでスライドを操作することもできます。

FCPresentationText オブジェクトのメソッド・サマリー

メソッド	説明
FCPresentationText.connect	クライアント・サイドとサーバー・サイドでコンポーネントが必要とする全てのアセットをセットアップします。
FCPresentationText.setUsername	表示しサーバーに渡すユーザ名を設定します。
FCPresentationText.close	クリーン・アップし接続を切断します。
FCPresentationText.nextSlide	リストの次のスライドへ移動します。
FCPresentationText.backSlide	リストの1つ前のスライドへ戻ります。
FCPresentationText.newSlide	新しいスライドを作り、スライドのリストに加えます。
FCPresentationText.deleteSlide	現在のスライドを消去します。

FCPresentationText.connect

利用

Flash Player 6

Flash Communication Server MX

使い方

```
presentText_mc.connect(nc)
```

パラメータ

nc アクティブな NetConnection オブジェクト

戻り値

なし

説明；

メソッド；クライアント・サイドとサーバー・サイドでコンポーネントが必要とする全てのアセットをセットアップします。

FCPresentationText.setUsername

利用

Flash Player 6

Flash Communication Server MX

使い方

```
presentText_mc.setUsername(newName)
```

パラメータ

newName 新しいユーザ名を特定するオプションのストリング。*newName* が null または undefined でない場合、ユーザはコンポーネントの全ての機能を使用できる。*newName* が null の場合は何も変更されず、ユーザは lurker mode に入る。

戻り値

なし

説明；メソッド；表示しサーバーに渡すユーザ名を設定する。

FCPresentationText.close

利用

Flash Player 6

Flash Communication Server MX

使い方

presentText_mc.close()

パラメータ

なし

戻り値

なし

説明；

メソッド；このクライアントが使用していたアセットを解放し、このコンポーネントの接続を切断することで、クリーンアップする。

FCPresentationText.nextSlide

利用

Flash Player 6

Flash Communication Server MX

使い方

presentText_mc.nextSlide()

パラメータ

なし

戻り値

なし

説明；

メソッド；リスト内の次のスライドへ移動する。

FCPresentationText.backSlide

利用

Flash Player 6

Flash Communication Server MX

使い方

presentText_mc.backSlide()

パラメータ

なし

戻り値

なし

説明；

メソッド；リスト内の前のスライドへ移動する。

FCPresentationText.newSlide

利用

Flash Player 6

Flash Communication Server MX

使い方

presentText_mc.newSlide()

パラメータ

なし

戻り値

なし

説明；

メソッド；新しいスライドを作成しスライドのリストに加える。

FCPresentationText.deleteSlide

利用

Flash Player 6

Flash Communication Server MX

使い方

presentText_mc.deleteSlide()

パラメータ

なし

戻り値

なし

説明；

メソッド；現在のスライドを削除する。

PresentationSWF コンポーネント

PresentationSWF コンポーネントは、共有する別の SWF ファイルを見せるプレゼンテーションを可能にします。

このコンポーネントは2つのモードで動作が可能です。*speaker mode*ではユーザがプレゼンターとなり、SWF ファイルを制御すると同時に、全てのユーザが同じフレームを見ます。*default mode*では、ユーザは視聴者となり、Next ボタンと Back ボタンを使うことで SWF ファイルを行き来し、非同期にファイルを見ることができます。しかし、プレゼンターがまだ行っていないプレゼンテーションを先に行って見ることはできません。

PresentationSWF コンポーネントは、SimpleConnection コンポーネントと合わせて使用すれば、スクリプティングの必要はなくなります。

presentationSWF ファイルを設定する

このコンポーネントでは、プレゼンテーション用 SWF としてロードする別の SWF ファイルが必要になります。以下のサンプルでは、Flash Communication Server MX をインストールした `flashcom¥applications¥sample_broadcast` ディレクトリにある、`simple_preso.swf` ファイルをコピーして使用します。

このコンポーネントに表示するプレゼンテーションを作成するときは、SWF ファイルを適切に設定する必要があります。

ファイルは、メインのタイムライン上にそれぞれのフレームがひとつのプレゼンテーション・スライドとしてある構成になっていなくてはなりません。

1 番めのフレームに stop アクションが書かれていることを確認します。

プレゼンテーション SWF ファイルには ActionScript を含まないようにしてください。メインのタイムラインを行き来するようなアクションは、コンポーネントに含まれています。

PresentationSWF コンポーネントを使用する

以下はこのコンポーネントの2通りの使用方法を記したものです。

セールス・プレゼンテーション：チャートや他の情報を示しながら、オンラインでプレゼンテーションを行います。

ミーティング・アジェンダ：会議中ユーザの注意を引きつけ、別のアジェンダに導きます。

以下のサンプルでは、2つのクライアント・ファイルを作成します。`presentSWF_test_speaker` はスライドを制御するプレゼンター用として、`presentSWF_test` は、プレゼンターではない参加視聴者用として作成します。どちらも `presentSWF_test` という同じアプリケーションに注目させますが、*speaker mode* にあるクライアントだけがプレゼンターになることができます。

話者アプリケーションでこのコンポーネントを使用する

1 Flash Communication Server が起動していることを確認する

2 Flash Communication Server MX のアプリケーション・ディレクトリ(通常は`flashcom¥applications`)にアプリケーション・ディレクトリを作成し、`presentSWF_test` の名前をつける。

3 アプリケーション・ディレクトリ内に `main.asc` の名前で新しいファイルを作成し、以下のコードを記述する。

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

Note:他のアプリケーション・ディレクトリからこのアプリケーション・ディレクトリへ、`main.asc` ファイルをコピーして使用することもできます。

4 Flash MX で、ステージ上に `presentationSWF` コンポーネントをドラッグし、プロパティ・インスペクタで、

presentSWF_mc のインスタンス名をつける。

5 プロパティ・インスペクタで、デフォルトの presentationSWF ファイル名、simple_preso.awf は変更しない。他のパラメータについては"PresentationSWF コンポーネント・プロパティ・インスペクタ"をご覧ください。

6 ステージ上に SimpleConnect コンポーネントをドラッグする。

7 SimpleCpnnection 用のプロパティ・インスペクタで以下のパラメータを入力する。

Application Directory テキスト・ボックスで、ステップ 2 で作成したアプリケーションの URI、例えば、rtmp:/presentSWF_test を入力する。

Communication Components テキスト・ボックスをダブル・クリックする。現れる値ダイアログ・ボックスで、+ 印をクリックし、PresentationSWF コンポーネントのインスタンス名 presentSWF_mc を入力し、OK をクリックする。

この値を与えることで、SimpleConnect コンポーネントは PresentationSWF コンポーネントの接続を制御できる。SimpleConnect コンポーネントがサーバーへの接続に成功したら、PresentationSWF コンポーネントは自動的にサーバーに接続する。

8 アクション・パネルの 1 番めのキーフレームにムービーの進行を止め、speaker mode に設定する以下のコードを記述する。

```
stop();  
_global.speakerMode = true;
```

9 アプリケーション・ディレクトリ内に presentSWF_test_speaker 名で保存、パブリッシュする。

視聴者アプリケーションで PresentationSWF コンポーネントを使用する

1 presentSWF_test_speaker fla をコピーし、presentSWF_test fla 名で保存する。

2 presentSWF_test fla 内で、アクション・パネルの 1 番めのキーフレームに、ユーザがプレゼンターにならないよう、stop コール後に以下のコードを記述する。

```
_global.speakerMode = false;
```

3 presentSWF_test fla を保存する。

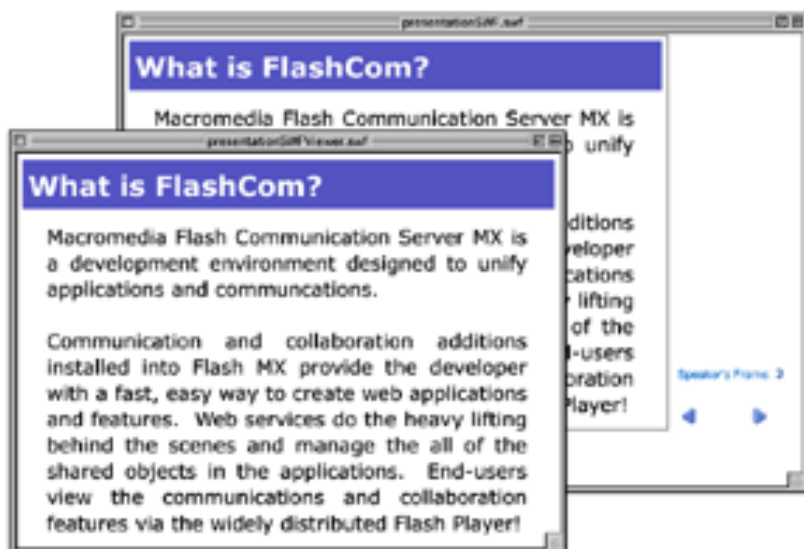
両方のモードで PresentationSWF コンポーネントをテストする

1 アプリケーション・ディレクトリで presentSWF_test_speaker.swf ファイルを開くか、Flash MX オーサーリング環境の場合は、制御>ムービプレビューを選択し、ログインする。

2 アプリケーション・ディレクトリで presentSWF_test.swf ファイルを開くか、Flash MX オーサーリング環境の場合は、制御>ムービプレビューを選択し、別のユーザとしてログインする。

presentSWF_test_speaker.swf では、ユーザはプレゼンターとなる。新しいスライドへ移動し、プレゼンテーション中先へ進んだり、戻ったりできる。presentSWF_test.swf では、ユーザは視聴者となる。スライドを見ることはできるが、プレゼンターが進んだところまでしか移動することはできない。

以下は、後ろにあるのがプレゼンター・クライアント・アプリケーションで、前にあるのが視聴者プレゼンテーション。



PresentationSWF コンポーネント・プロパティ・インスペクタ

名前	変数	説明
PresentationSWF	swFile	ストリング；デフォルト値は simple_preso.swf。プレゼンテーションで ロードするファイルを特定する。LoadSWF メソッドを使って動作中このファイルを変更することもできる。
Viewer Button Enabled	viewerButtons	Boolean 値；デフォルト値は true。true なら speaker mode でないユーザもプレゼンテーション・ファイルを行き来できる。

PresentationSWF コンポーネントのリスキニング

このコンポーネントのアセットは、ライブラリ・パネルの Core Assets-Developer Only\PresentationSWF Assets ディレクトリにあります。

関係するコンポーネント

“SimpleConnect コンポーネント” 6 4 p

FCPresentation オブジェクト

FCPresentation API は、サーバーに接続したりクリーンアップするオブジェクトに、クライアント・サイドとサーバーサイドでの機能を提供します。またプログラミングでフレームを操作することもできます。

FCPresentationSWF オブジェクト・メソッド・サマリー

メソッド

	説明
FCPresentationSWF.connect	クライアント・サイドとサーバー・サイドでコンポーネントが必要とする全てのアセットをセットアップします。
FCPresentationSWF.close	クリーン・アップし接続を切断します。
FCPresentationSWF.loadSWF	プレゼンテーションで使用する新たな SWF ファイルを設定するためにプレゼンターがオプションで使用します。
FCPresentationSWF.next	1 フレームプレゼンテーション SWF を進めます。
FCPresentationSWF.back	プレゼンテーション SWF 内で 1 フレーム戻ります。
FCPresentationSWF.sync	視聴者のフレームとプレゼンターのフレームを再同調させます。

変数

グローバル変数

	説明
<code>_global.speakerMode</code>	Boolean 値 ; デフォルト値は true。true の場合、他のユーザが見ている今のスライドを変更できます。false なら、プレゼンターが見せるスライドを見るだけです。この変数をプログラミングで設定するには、 <code>_global.speakerMode = [true または false]</code> に設定します。

`_global.speakerMode`

利用

Flash Player 6

Flash Communication Server MX

使い方

```
_global.speakerMode = true;
```

説明 ;

グローバル変数 ; speaker mode を制御する Boolean 値。true(デフォルト値)ならユーザは他のユーザが見ている現在のスライドを変更できる。False ならユーザは、プレゼンターが選んで見せるスライドのみ見ることができる。この変数を設定するには、`_global.speakerMode` を true か false に設定する。

FCPresentationSWF.connect

利用

Flash Player 6

Flash Communication Server MX

使い方

```
presentSWF_mc.connect(nc)
```

パラメータ

`nc` アクティブな NetConnection オブジェクト

戻り値

なし

説明；

メソッド；クライアント・サイドとサーバー・サイドでコンポーネントが必要とする全てのアセットをセットアップする。

FCPresentationSWF.close**利用**

Flash Player 6

Flash Communication Server MX

使い方

presentSWF_mc.close()

パラメータ

なし

戻り値

なし

説明；

メソッド；クライアント上でこのコンポーネントが使用したアセットを解放し、クリーンアップする。

FCPresentationSWF.loadSWF**利用**

Flash Player 6

Flash Communication Server MX

使い方

presentSWF_mc.loadSWF(swfFile)

パラメータ

swfFile SWF ファイルへの相対または絶対パスを指定するストリング。

戻り値 なし

説明；

メソッド；プレゼンテーションで使用する新たな SWF ファイルを設定するためにプレゼンターがオプションで使用する。このメソッドは、視聴者がファイルをロードするためには使用できない。SWF ファイルはプレゼンターが新しい

フ

ファイルをロードすると自動的にロードされる。

FCPresentationSWF.next**利用**

Flash Player 6

Flash Communication Server MX

使い方

presentSWF_mc.next()

パラメータ

なし

戻り値

なし

説明；

メソッド； 1フレームだけプレゼンテーション SWFを進める。speaker mode であれば、フレームは全ての視聴者と同調して進む。speaker mode でなく視聴者のボタンが実行可能な場合は、視聴者はローカルで（他のユーザには影響を与えず）フレームを進めることができるが、プレゼンターの現在のフレームを超えて先を見ることはできない。

FCPresentationSWF.back

利用

Flash Player 6

Flash Communication Server MX

使い方

`presentSWF_mc.back()`

パラメータ

なし

戻り値

なし

説明；

メソッド； プレゼンテーション SWF 内で1フレーム戻る。ユーザが speaker mode であれば、フレームは全ての視聴者と同調して戻る。ユーザが speaker mode でなく視聴者のボタンが実行可能な場合は、ローカルで変更でき、他の視聴者には影響しない。

FCPresentationSWF.sync

利用

Flash Player 6

Flash Communication Server MX

使い方

`presentSWF_mc.syncs()`

パラメータ

なし

戻り値

なし

説明；

メソッド； 視聴者のフレームと話者のフレームを再同調させる。

AVPresence コンポーネント

AVPresence コンポーネントは、作成したアプリケーションで音声や映像の送受信を可能にします。このコンポーネントは、接続したユーザなら誰でも使用できる、プレゼンター・シートの役目を果たします。ユーザが音声か映像、もしくはその両方をコンポーネントのインスタンスから送信すると、他のユーザは自動的にその音声や映像を見たり聞いたりできます。コンポーネントのインスタンスが使用されていない場合、ユーザはそれをクリックして音声や映像を送信できます。ユーザは音声や映像、その両方を切断するだけでなく、送信も受信もできるコントロールを持ちます。ユーザが、入ってくるストリーム上でこのコントロールを使うと、結果はローカル上に現れます。自分自身の音声や映像のインスタンス上でこのコントロールを使うと、接続している全てのユーザのインターフェイス上にその効果は現れます。このように、見られているか聞かれているかだけでなく、誰を見たり、聞くかをどのユーザも自分でコントロールできるのです。

このコンポーネントは、SimpleConnect コンポーネントと併用するとスクリプティングの必要はなくなります。ステージ上にドラッグするコンポーネント・インスタンスにはどれも唯一のインスタンス名が必要です。AVPresence コンポーネントには変更可能ないくつかのコンポーネント・パラメータがあります。

AVPresence コンポーネントを使用する：

このコンポーネントは多くの用途に使用できるコミュニケーション・コンポーネントであり、ユーザに仮想空間を提供するために使用できます。以下はこのコンポーネントの使用例です。

パネル・ディスカッション：人々が聴衆の前で討論を交わしたり、ディベート、プレゼンテーションを行う。

ビデオ電話：1つのアプリケーションに2つの AVPresence を使うことで、ビデオ電話アプリケーションを作成できます。

アプリケーションでコンポーネントを使用するには

1 Flash Communication Server が起動していることを確認する

2 Flash Communication Server MX のアプリケーション・ディレクトリ(通常は`flashcom/applications`)にアプリケーション・ディレクトリを作成し、例えば `avPresence_test` の名前をつける。

3 アプリケーション・ディレクトリ内に `main.asc` の名前で新しいファイルを作成し、以下のコードを記述する。

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

Note:他のアプリケーション・ディレクトリからこのアプリケーション・ディレクトリへ、`main.asc` ファイルをコピーして使用することもできます。

4 Flash MX で、ステージ上に AVPresence コンポーネントをドラッグする。

5 プロパティ・インスペクタで、以下の値を入力する。

コンポーネントのインスタンス名に `avPresence_mc` をつける。

パラメータの定義は、18ページの“AVPresence コンポーネント・プロパティ・インスペクタ”を参照のこと。

6 ステージ上に Flash Communication Server への接続を設定する SimpleConnect コンポーネントをドラッグする。

7 SimpleConnect コンポーネントのプロパティ・インスペクタで、以下の2つのパラメータを入力する。

Application Directory テキスト・ボックスで、ステップ2で作成したアプリケーションのURI、例えば、`rtmp://AVPresence_test`を入力する。

Communication Components テキスト・ボックスをダブル・クリックする。現れる値ダイアログ・ボックスで、+印をクリックし、AVPresence コンポーネントのインスタンス名 `avPresence_mc` を入力し、OK をクリックする。

この値を与えることで、SimpleConnect コンポーネントは AVPresence コンポーネントの接続を制御できる。SimpleConnect コンポーネントがサーバーへの接続に成功したら、AudioConference コンポーネントは自動的にサーバーに接続する。

8 `myAVPresence` のファイル名でアプリケーション・ディレクトリ内に保存、パブリッシュする。

9 アプリケーション・ディレクトリ内にある SWF ファイルを開く(または Flash MX オーサリング環境では、制御>ムービープレビューを選択する)。ログインし Send Audio/Video ボタンをクリックする。

以前に Flash Player のプライバシー設定を行っていない場合は、カメラ、マイクへのアクセスを許可するかどうか聞かれる。

オーディオ・デバイスデバイスが作動している場合、喋ると音声レベルが上下する。ビデオ・デバイスが作動していると、ビデオ画面にその出力結果を見ることができる。マウス・ポインタをビデオ画面上で動かすと、マイクとカメラのUIを見ることができる。これは音声や映像データへの変換を止めるボタンとしても働く。

AVPresence コンポーネント・プロパティ・インスペクタ

コンポーネント・インスタンスのプロパティ・インスペクタで以下の変数を設定できます。

名前	変数	説明
Presenter SharedObject	<code>soName</code>	ストリング：デフォルト値： <code>av</code> 。動作中コンポーネントを制御するため、サーバー上で使用される共有 オブジェクト名を決定する。
Sync Speed	<code>updateFps</code>	数値：デフォルト値：3。このコンポーネントが毎秒何回メッセージを送るかを設定する。この値は音声、映像クオリティとは独立しており、インスタンスが、音声のアップデート・レートや、送受信する新しいストリームにどれくらい速く応答するかを制御する。
Video Width	<code>vidWidth</code>	数値：デフォルト値：120。ローカルでのカメラ設定で画像の幅をピクセル単位で設定する。 <code>vidWidth</code> は <code>FCSetBandwidth</code> が使用されたら上書きされる。
Video Height	<code>vidHeight</code>	数値：デフォルト値：120。ローカルでのカメラ設定で画像の高さをピクセル単位で設定する。 <code>vidHeight</code> は <code>FCSetBandwidth</code> が使用されたら上書きされる。
Video Bandwidth	<code>vidBandwidth</code>	数値：デフォルト値：12,000。ローカルでのカメラ設定で帯域幅の最大値をビット単位で設定する。 <code>VidBandwidth</code> は <code>FCSetBandwidth</code> が使用されたら上書きされる。
Video Quality	<code>vidQuality</code>	数値：デフォルト値：0。ローカルでのカメラ設定で最大品質を設定する。0は

FCAVPresence.setUsername

利用

Flash Player 6

Flash Communication Server MX

使い方

avPresence_mc.setUsername(newName)

パラメータ

newName 新しいユーザ名を記述するオプションのストリング。*NewName* パラメータが null、もしくは undefined でない場合、Flash はそのユーザ用の新しい参照を作り、名前を *newName* に設定する。*NewName* が null の場合は、何も変化せず、ユーザは lurker mode となる。

戻り値

なし

説明

メソッド； アクティブなユーザのリストで表示するためのユーザ名を取得する

FCAVPresence.close

利用

Flash Player 6

Flash Communication Server MX

使い方

avPresence_mc.close()

パラメータ

なし

戻り値

なし

説明

メソッド； クライアント上でコンポーネントが使用しているアセットを解放しクリーンアップする。

AudioConference コンポーネント

AudioConference コンポーネントを使用すると、多数のクライアントによる同時ストリーミングに対応する、マルチプレックスなアプリケーションが作成できます。このアプリケーションはまた、アプリケーションに今ログインしているユーザをリストアップするユーザ・インターフェイス(UI)を提供します。UI は接続している各ユーザをライトで示し、ユーザが音声を送るとライトは緑色に変わります。ユーザは会話に加わるには Talk ボタンをクリックするか、喋ると自動的に音声を送る Auto チェック・ボックスを選択します。このコンポーネントでは、リストに表示されたり他のユーザと話をするには、まずログインすることが必要となります。

このコンポーネントは、SimpleCpconnect と併用すればスクリプティングの必要はなくなります。

AudioConference コンポーネントを使う

以下はこのコンポーネントについてのいくつかの使用例です。

パーティ・ライン：ユーザのグループがある話題を同時に討論する。

仮想電話会議：電話をかけてスピーカーフォンで行う会議の代わりに行うオンライン本社会議。このコンポーネントを使って、企業イメージを使ったカスタム・アプリケーションを簡単に作成することができます。

テクニカル・サポート：トレーニング器具や複雑なアプリケーションを扱う場合、問題解決フォーラムでこのコンポーネントを使用する。

アプリケーションでコンポーネントを使用するには

1 Flash Communication Server が起動していることを確認する

2 Flash Communication Server MX のアプリケーション・ディレクトリ(通常は`flashcom` applications)にアプリケーション・ディレクトリを作成し、例えば `audioconf_test` の名前をつける。

3 アプリケーション・ディレクトリ内に `main.asc` の名前新しいファイルを作成する。`components.asc` ファイルをロードする以下のコードを記述する。

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

Note:他のアプリケーション・ディレクトリからこのアプリケーション・ディレクトリへ、`main.asc` ファイルをコピーして使用することもできます。

4 Flash MX で、ステージ上に AudioConference コンポーネントをドラッグする。

5 プロパティ・インスペクタで、AudioConference コンポーネントのインスタンス名に `audioconf_mc` をつける。

6 ステージ上に SimpleConnect コンポーネントをドラッグする。

7 プロパティ・インスペクタで、以下の2つのパラメータを入力する。

Application Directory テキスト・ボックスで、ステップ2で作成したアプリケーションの URI、例えば、`rtmp:/audioconf_test` を入力する。

Communication Components テキスト・ボックスをダブル・クリックする。現れる値ダイアログ・ボックスで、+ 印をクリックし、AudioConference コンポーネントのインスタンス名 `audioconf_mc` を入力し、OK をクリックする。

この値を入力することで、SimpleConnect コンポーネントは AVPresence コンポーネントの接続を制御できる。

SimpleConnect コンポーネントがサーバーへの接続に成功したら、AudioConference コンポーネントは自動的にサーバーに接続する。

8 `myAudioApp` のファイル名でアプリケーション・ディレクトリ内に保存、パブリッシュする。

9 アプリケーション・ディレクトリ内にある SWF ファイルを開く(または Flash MX オーサリング環境では、制御>ムービープレビューを選択する)。ログインし Auto チェック・ボックスをクリックする。

オーディオ入力デバイスが動作していれば、話すときライトが緑に変わる。ライトはそれぞれのユーザ名の隣に現れる。話していなければライトはくすんだ色になる。



AudioConference コンポーネントをリスキニングする

このコンポーネントの-assets は、ライブラリ・パネルの Core Assets-Developer Only\AudioConference Assets ディレクトリにあります。

関係するコンポーネント

“SimpleConnect コンポーネント” 64 ページ

FCAudioConference オブジェクト

FCAudioConference オブジェクトでは、SimpleConnect コンポーネントと併用すればスクリプティングの必要はありません。ユーザは、リストに現れ話すには、ログインしなければなりません。

FCAudioConference API は、サーバーに接続したりクリーンアップするオブジェクトに、クライアント・サイドとサーバー・サイドでの機能を提供します。

FCAudioConference オブジェクトのメソッド・サマリー

メソッド

記述

FCAudioConference.connect	サーバー上のアプリケーションに接続する
FCAudioConference.setUsername	ユーザ名を設定し、アクティブなユーザのリストに表示する
FCAudioConference.close	クリーンアップし、音声会議からそのユーザを消去する

FCAudioConference.connect

利用

Flash Player 6

Flash Communication Server MX

使い方

myAudioConf_mc.connect(nc)

パラメータ

nc アクティブな NetConnection オブジェクト

戻り値

なし

説明

メソッド；サーバー上のアプリケーションに接続する。また、クライアント・サイドやサーバー・サイドでコンポーネントが必要とする全てのアセットをセット・アップし、サーバーが提供する名前で setUsername メソッドをコールする。

FCAudioConference.setUsername

利用

Flash Player 6

Flash Communication Server MX

使い方

myAudioConf_mc.setUsername([newName])

パラメータ

newName リストに表示するユーザ名を含むストリング。*newName* が null または *undefined* でない場合、Talk オプションと Auto オプションが選択可能になり、ユーザ名がリストに表示される。*newName* が null の場合は何も変化せず、このユーザは lurker mode(見ているだけ)となる。これは他のユーザがこのユーザの存在に気づかないことを意味する。

戻り値

なし

説明

メソッド：アクティブなユーザのリストに表示するためユーザ名を設定する。

FCAudioConference.close

利用

Flash Player 6

Flash Communication Server MX

使い方

myAudioConf_mc.close()

パラメータ

なし

戻り値

なし

説明

メソッド：クライアントのコンポーネントで使用されていたアセットを解放し、音声会議からユーザを消去することでクリーンアップする。

VideoConference コンポーネント

このコンポーネントは、AVPresence コンポーネントを用いて、多数のユーザが互いに音声や映像を使用することを可能にします。VideoConference コンポーネントは SimpleConnection コンポーネントと併用すれば、スクリプティングの必要はなくなります。

VideoConference コンポーネントを使用する

アプリケーション内にこの多機能なコミュニケーション・コンポーネントを使用することで、ビデオ会議スペースを作成

することができます。

VideoConference、SimpleConnect、SetBandwidth、ConnectionLight の各コンポーネントをステージ上にドラッグして

ビデオ会議アプリケーションを作成します。

アプリケーションにコンポーネントを使用する

1 Flash Communication Server が起動していることを確認する

2 Flash Communication Server MX のアプリケーション・ディレクトリ(通常は`flashcomapplications`)にアプリケーション・ディレクトリを作成し、`video_test`の名前をつける。

3 Flash MX で、ステージ上に `videoConference` コンポーネントをドラッグする。

4 プロパティ・インスペクタで、インスタンス名 `videoconf_mc` をつける。他のパラメータについては、68 ページの `videoConference` コンポーネント・プロパティ・インスペクタをご覧ください。

5 ステージ上に `SimpleConnect` コンポーネントをドラッグする。

6 `SimpleCpnnection` 用のプロパティ・インスペクタで以下のパラメータを入力する。

Application Directory テキスト・ボックスで、ステップ2で作成したアプリケーションの URI、`rtmp://video_test` を入力する。

Communication Components テキスト・ボックスをダブル・クリックする。現れる値ダイアログ・ボックスで、+ 印をクリックし、`videoConference` コンポーネントのインスタンス名 `videoconf_mc` を入力し、OK をクリックする。

この値を与えることで、`SimpleConnect` コンポーネントは `videoConference` コンポーネントの接続を制御できる。

`SimpleConnect` コンポーネントがサーバーへの接続に成功したら、`videoConference` コンポーネントは自動的にサーバーに接続する

7 アプリケーション・ディレクトリ内に `video_test` 名で保存、パブリッシュする。

8 このアプリケーション・ディレクトリ内に `main.asc` の名前で新しいファイルを作成し、以下のコードを記述する。

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

Note:他のアプリケーション・ディレクトリからこのアプリケーション・ディレクトリへ、`main.asc` ファイルをコピー

して使用することもできます。

9 アプリケーション・ディレクトリ内にある SWF ファイルを開くか、または Flash MX オーサリング環境では、制御>ムービープレビューを選択しログインし、Auto チェック・ボックスをクリックする。

以前に Flash Player のプライバシー設定を行っていない場合は、カメラ、マイクへのアクセスを許可するかどうか聞かれる。

オーディオ・デバイスが作動している場合、喋ると音声レベルが上下する。ビデオ・デバイスが作動していると、ビデオ画面にその出力結果を見ることができる。マウス・ポインタをビデオ画面上で動かすと、マイクとカメラの UI を見ることができる。これは音声や映像データへの変換を止めるボタンとしても働く。

dragSharing プロパティが true に設定されていると、ビデオ画面を動かすことができる。

VideoConference コンポーネント・プロパティ・インスペクタ

名前	変数	説明
Show Boundary	showBoundary	Boolean 値；デフォルト値は false。ビデオ会議エリアの境界線を示すかどうかを決定する。
Show Background	showBackground	Boolean 値；デフォルト値は false。ビデオ会議に背景を設定するか、透明にするかを決定する。
Clip Mask	clipMask	Boolean 値；デフォルト値は false。ビデオ・ウィンドウがビデオ会議エリアに切り取られるかどうかを決定する。
Drag Sharing	dragSharing	Boolean 値；デフォルト値は true。ビデオ・ウィンドウのポジションを共有するかどうかを決定する。共有することによりビデオウィンドウは離れたユーザがドラッグすると同じように移動する。どのようにドラッグしても他のユーザに見える。

VideoConference コンポーネントのリスキニング

このコンポーネントのアセットは、ライブラリ・パネルの Core Assets-Developer Only\VideoConference Assets ディレクトリにあります。

関係するコンポーネント

“SimpleConnect”コンポーネント 64 ページ、“AVPresence”コンポーネント 17 ページ、“SetBandwidth”コンポーネント。

FCVideoConference オブジェクト

FCVideoConference API は、オブジェクトがサーバーに接続したりクリーンアップしたりする、クライアント・サイドやサーバー・サイドの機能を提供します。

FCVideoConference オブジェクト・メソッド・サマリー

メソッド

説明

FCVideoConference.connect

ビデオ会議を開始する。

FCVideoConference.setUsername

newName パラメータが null または undefined でない場合、メソッドがユーザのアイデンティティを変更する。そうでなければカメラとマイ

ク

のパブリッシュを止める。

FCVideoConference.close

ビデオ会議への参加を止める。

FCVideoConference.connect

利用

Flash Player 6

Flash Communication Server MX

使い方

myVideoConf_mc.connect(*nc*)

パラメータ

nc アクティブな NetConnection オブジェクト

戻り値

なし

説明 ; メソッド ; ビデオ会議を開始する。ユーザ名が設定されていれば、このメソッドはカメラとマイクをパブリッシュ

シ
ユする。

FCVideoConference.setUsername

利用

Flash Player 6

Flash Communication Server MX

使い方

myVideoConf_mc.setUsername(*newName*)

パラメータ

newName スtring

戻り値

なし

説明 ;

メソッド ; *newName* が null または undefind でなければ、このメソッドはユーザ・アイデンティティを変更する。

NewName が null または undefined の場合は、カメラとマイクをパブリッシュするストリームは止まる。

FCVideoConference.close

利用

Flash Player 6

Flash Communication Server MX

使い方

```
myVideoConf_mc.close()
```

パラメータ

なし

戻り値

なし

説明；

メソッド；このクライアント上でこのコンポーネントが使用しているアセットを解放し、ビデオ会議内でのこのクライアントの参加を終わらせることで、クリーン・アップする。

SetBandwidth コンポーネント

このコンポーネントにより、ユーザはアップロードとダウンロードの帯域幅を特定させることができます。この新しい設定が、コンポーネントが操作するマイクやカメラの品質を左右します。これにより複数のマイクやカメラを制御することができ、帯域幅に優先順位をつけて割り振られます。

ユーザはアップロード、ダウンロードで Modem、DSL、LAN、Custom を選択できます。Custom を選択すると、現れるダイアログ・ボックスで、ユーザは固定された帯域幅数を選ぶか、帯域幅数を入力できます。帯域幅は対称か非対称かどちらかに設定されます。ユーザが非対称オプションを選んだ場合は、アップロードとダウンロードの数字は等しくなりません。

SetBandwidth コンポーネントは、ユーザが選択したアップロードとダウンロードの帯域幅を制限する、サーバー・サイドの複製を使用します。帯域幅の制限により、接続に可能な帯域幅を Flash Communication Server は認知し、一般的に音声と映像の品質を向上させます。

Optimal settings でのパブリッシュが望まれます。これは、接続者が受信するより高いレートにはしない、という設定です。例えば、オンラインのプレゼンターは高スピードで接続していますが、視聴者はモデムや DSL 接続で見

て

いるといった場合、モデムのスピードより速い設定でパブリッシュすべきではありません。なぜならばモデムのスピードは結果として、品質を著しく落としかねないからです。プレゼンターは DSL 設定でパブリッシュしたいかも知れま

せ

んが、モデムユーザの見るインターフェイスでは、動きの品質がはなはだ損なわれることとなります。

SetBandwidth コンポーネントは、SimpleConnect コンポーネントと併用すれば、スクリプティングに必要ななくなります。

SetBandwidth コンポーネントを使用する

シンプルでパワフルなこのコンポーネントは、可能なアップロード帯域幅でパブリッシュされた、マイクやカメラの品質に自動的に合わせます。このコンポーネントは、ライブ音声や映像をパブリッシュするあらゆるアプリケーションに使用できます。

以下はこのコンポーネントの2つの使用例です。

映像会議：FCVideoConference は、FCSetBandwidth と併用できるように作成されている AVPresence コンポーネントを使用するので、マイクやカメラの品質は自動的に調整されます。

プレゼンテーション：プレゼンテーション・アプリケーションのコンセプトに、1人のプレゼンターに多くの視聴者というのがあります。そういったアプリケーションで、AVPresence コンポーネントは、ビデオ・プレゼンテーションを提供します。こうしたアプリケーションで SetBandwidth コンポーネントと共用することは効果的です。

アプリケーションでコンポーネントを使用する

1 Flash Communication Server が起動していることを確認する

2 Flash Communication Server MX のアプリケーション・ディレクトリ(通常は`flashcom/applications`)にアプリケーション・ディレクトリを作成し、`setBand_test` の名前をつける。

3 アプリケーション・ディレクトリ内に `main.asc` の名前で新しいファイルを作成し、以下のコードを記述する。

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

Note:他のアプリケーション・ディレクトリからこのアプリケーション・ディレクトリへ、`main.asc` ファイルをコピーして使用することもできます。

4 Flash MX で、ステージ上に SetBandwidth コンポーネントをドラッグする。

5 プロパティ・インスペクタで、インスタンス名 `setBand_mc` をつける。他のパラメータについては、61ページの FCSetBandwidth オブジェクト・プロパティ・サマリーをご覧ください。

6 ステージ上に SimpleConnect コンポーネントをドラッグする。

7 SimpleCpnnection 用のプロパティ・インスペクタで以下のパラメータを入力する。

Application Directory テキスト・ボックスで、ステップ2で作成したアプリケーションの URI、例えば、`rtmp:/setBand_test` を入力する。

Communication Components テキスト・ボックスをダブル・クリックする。現れる値ダイアログ・ボックスで、+ 印をクリックし、SetBandwidth コンポーネントのインスタンス名 `setBand_mc` を入力し、OK をクリックする。

この値を与えることで、SimpleConnect コンポーネントは SetBandwidth コンポーネントの接続を制御できる。SimpleConnect コンポーネントがサーバーへの接続に成功したら、SetBandwidth コンポーネントは自動的にサーバーに接続する。

8 アプリケーション・ディレクトリ内に `SetBand_tst` 名で保存、パブリッシュする。

9 アプリケーション・ディレクトリ内にある SWF ファイルを開くか、または Flash MX オーサリング環境では、制御 >ムービープレビューを選択しログインする。

下記のように、違った帯域幅を設定できる。



SetBandwidth コンポーネントのリスキニング

このコンポーネントのアセットは、ライブラリ・パネルの Core Assets-Developer Only\SetBandwidth Assets ディレクトリにあります。

関係するコンポーネント

“AVPresence” コンポーネント 1 7 P、“SimpleConnect コンポーネント”

FCSetBandwidth オブジェクト

FCSetBandwidth オブジェクトは、manage と unmanage メソッドをもつグローバルな Quality Manager(gFlashCom.quality)を設定することで、カメラやマイクの品質を管理します。

以下のサンプルは、マイクとカメラ 2 セットが登録されることを示しています。1 つめと 2 つめのパラメータは、それぞれ登録されるマイクとカメラの名前を付けます。3 つめのパラメータは、オプションで、カメラのアスペクト比(高さ/幅)を設定します。最後のパラメータは、帯域幅の共有量を決定します。サンプルでは、mic1/cam1 が 1/3、mic2/cam2 が 2/3 を得ます。このパラメータを指定しないと、帯域幅は等しく分けられます。

```
gFlashCom.quality.manage(mic1, cam1, 1,100);
```

```
gFlashCom.quality.manage(mic2, cam2, .75,200);
```

マイクとカメラの管理を止めるには、以下のように記述します。Quality Manager が自動的に品質を調整します。

```
gFlashCom.quality.manage(mic2, cam2);
```

アプリケーションが SetBandwidth コンポーネントを使用していて、AVPresence コンポーネントを使用していない場合は、Quality Manager を使用してマイクとカメラを登録しなければなりません。AVPresence s コンポーネントは、自動的に Quality Manager オブジェクトをコールします。

FCSetBandwidth オブジェクト・メソッド・サマリー

メソッド

説明

FCSetBandwidth.connect
現在選択されている帯域幅にしたがって、特定の NetConnection オブジェクトの帯域幅制限を設定する。

FCSetBandwidth.close
クリーン・アップする

FCSetBandwidth オブジェクト・プロパティ・サマリー

FCSetBandwidth.modemUp
Modem オプションを選択したとき、アップロードの帯域幅を決定する。デフォルトは 3 3 Kbps アップ。

FCSetBandwidth.modemDown
Modem オプションを選択したとき、ダウンロードの帯域幅を決定する。デフォルトは 3 3 Kbps ダウン。

FCSetBandwidth.dslUp
DSL オプションを選択したとき、アップロードの帯域幅を決定する。デフォルトは 1 2 8 Kbps アップ。

FCSetBandwidth.dslDown
DSL オプションを選択したとき、ダウンロードの帯域幅を決定する。デフォルトは 2 5 6 Kbps ダウン。

FCSetBandwidth.lanUp
LAN オプションを選択したとき、アップロードの帯域幅を決定する。デフォルトは 1 0 0 0 Kbps アップ。

FCSetBandwidth.lanDown
LAN オプションを選択したとき、ダウンロードの帯域幅を決定する。デフォルトは 1 0 0 0 Kbps ダウン。

FCSetBandwidth.modemUp

利用

Flash Player 6

Flash Communication Server MX

使い方

setBand_mc.modemUp = maxSpeed

説明 ;

プロパティ。Modem オプションを選択したとき、アップロードの帯域幅を決定する。これは設定、取得できるプロパティ。変数 *maxSpeed* は Kbps 単位のアップロードのスピード。デフォルトは 3 3 Kbps。

FCSetBandwidth.modemDown

利用

Flash Player 6

Flash Communication Server MX

使い方

setBand_mc.modemDown = maxSpeed

説明；

プロパティ。Modem オプションを選択したとき、ダウンロードの帯域幅を決定する。これは設定、取得できるプロパティ。変数 *maxSpeed* は Kpbs 単位のダウンロードのスピード。デフォルトは 3 3 Kbps。

FCSetBandwidth.dslUp

利用

Flash Player 6

Flash Communication Server MX

使い方

setBand_mc.dslUp= maxSpeed

説明；

プロパティ。DSL オプションを選択したとき、アップロードの帯域幅を決定する。これは設定、取得できるプロパティ。変数 *maxSpeed* は Kpbs 単位のアップロードのスピード。デフォルトは 1 2 8 Kbps。

FCSetBandwidth.dslDown

利用

Flash Player 6

Flash Communication Server MX

使い方

setBand_mc.dslDown = maxSpeed

説明；

プロパティ。DSL オプションを選択したとき、ダウンロードの帯域幅を決定する。これは設定、取得できるプロパティ。変数 *maxSpeed* は Kpbs 単位のダウンロードのスピード。デフォルトは 2 5 6 Kbps。

FCSetBandwidth.lanUp

利用

Flash Player 6

Flash Communication Server MX

使い方

setBand_mc.lanUp= maxSpeed

説明；

プロパティ。LAN オプションを選択したとき、アップロードの帯域幅を決定する。これは設定、取得できるプロパティ。変数 *maxSpeed* は Kpbs 単位のアップロードのスピード。デフォルトは 1 0 0 0 Kbps。

FCSetBandwidth.lanDown

利用

Flash Player 6

Flash Communication Server MX

使い方

setBand_mc.lanDown = maxSpeed

説明；

プロパティ。LAN オプションを選択したとき、ダウンロードの帯域幅を決定する。これは設定、取得できるプロパティ。変数 *maxSpeed* は Kpbs 単位のダウンロードのスピード。デフォルトは 1 0 0 0 Kbps。

FCSetBandwidth.connect

利用

Flash Player 6

Flash Communication Server MX

使い方

`setBand_mc.connect(nc)`

パラメータ

`nc` アクティブな NetConnection オブジェクト。

戻り値

なし

説明；

メソッド。現在選択されている帯域幅にしたがって、`nc` パラメータの帯域幅制限を設定する。

FCSetBandwidth.close

利用

Flash Player 6

Flash Communication Server MX

使い方

`setBand_mc.close()`

パラメータ

なし

戻り値

なし

説明；

このクライアント上でこのコンポーネントが使用するアセットを解放し、クリーン・アップする。

SetBandwidth コンポーネントとコンポーネントを合わせて使用する

カメラやマイクの品質が変わったとき、サーバーにそれを知らせるイベントをコンポーネントに書き込むことができます。これらのイベントに常に注視するには、下記のステップを踏みます。

1 以下のように `ocChange` メソッドを与える。

```
listener.onChange = function(microphone, camera)
```

```
{
```

```
    //セッティングの再調整など
```

```
}
```

2 以下のように Quality Manager にリスナーを登録する。

```
GFlashCom.quality.addListener(listener);
```

これで、SetBandwidth コンポーネントがマイクやカメラの設定を調整すると常に `onChange` メソッドがコールされます。

RoomList コンポーネント

RoomList コンポーネントは、チャット・アプリケーションのように、ユーザが部屋を作成、参加し、削除する上級のコンポーネントです。このコンポーネントは、他のアプリケーションへのユーザ・アクセスを管理する使い勝手の良いロビー・アプリケーションに不可欠です。

Create Room をクリックすると、部屋名と説明のフィールドのあるポップ・アップ・メニューが現れます。部屋を選択し、Join をクリックすることで、新しいブラウザ・ウィンドウが開き、選択した部屋、つまり別のアプリケーション・インスタンスが現れます。誰もいなければユーザはリストから部屋を消去できます。コンポーネントの全機能を使うには、ユーザはユーザ名を入力してサーバーに接続する必要があります。

RoomList コンポーネントを使用する

このコンポーネントではロビーと部屋の2つのアプリケーションを作成します。ロビー・アプリケーションはユーザのスタート地点となり、部屋アプリケーションは、ユーザの行き先となります。

Note: ユーザはローカル、リモート環境に関わらず、web ブラウザを使用してこのアプリケーションにアクセスします。例えば、典型的な開発環境では、<http://localhost> でアプリケーションをテストします。

RoomList コンポーネントは、SimpleConnect コンポーネントを使用すれば、クライアント・サイドでのスクリプティングの必要はなくなります。ロビー・アプリケーションには、components.asc ファイルをロードする main.asc ファイルを作成する必要があります。

部屋アプリケーションでは、components.asc ファイルをロードする main.asc ファイルの中で、サーバー・サイド・スクリプトを記述する必要があります。それによってアプリケーションは、ユーザ名と部屋名を得、部屋数とユーザ数を管理します。この main.asc ファイルには、前後にパラメータを渡すサーバー・サイドのコールを含みますが、このコールにはユーザが実際に部屋アプリケーションに接続しているかどうかを制御するコールは含みません。サーバー・サイドの機能は、ただロビー・アプリケーションの部屋で表示されている人数を更新し、アプリケーションにアプリケーション

シ
ョン名を渡すだけです。このことによってそれぞれにインスタンスにおいて、ダイナミックに部屋名が表示されるのです。

RoomList コンポーネントから開いた部屋アプリケーションは、送信する username と appInstance パラメータを使用する必要がありますが、SimpleConnect アプリケーションを使用すれば、この過程は自動的に処理されます。例えば、Flash Communication Server MX とともにインストールされる sample_room アプリケーションをご覧ください。

ロビー・アプリケーションでは、以下の手順を踏みます。

- 1 SimpleConnect コンポーネントを加える
- 2 RoomList コンポーネントを加える
- 3 ロビー・アプリケーションを拡張する components.asc ファイルをロードする。

この後、部屋アプリケーションでは、以下の手順を踏みます。

- 4 以前作成したチャット・ルーム・アプリケーションに手を加え、行き先にします。
- 5 ロビー・アプリケーションからチャット・ルームに値を渡すため、HTML を変更します。

以下は、このコンポーネントを使用した2つの例です。

チャット・ロビー：ユーザが集まる中央ロビー、会議場所を作成します。その後参加するには部屋を選択します。

アプリケーション・インスタンス・ランチャー：仮想センターを作成し、そこからユーザはアプリケーションの多様なインスタンスを生み出します。

*Note:*以下のサンプルでは、21ページの“Chat コンポーネントを使用する”のサンプルを作成したものと仮定しています。

ロビー・アプリケーションでコンポーネントを使用する

1 Flash Communication Server が起動していることを確認する

2 Flash Communication Server MX のアプリケーション・ディレクトリ(通常は`flashcomapplications`)にアプリケーション・ディレクトリを作成し、`lobby_com_test`の名前をつける。

3 Flash MX で、ステージ上に SimpleConnect コンポーネントをドラッグする。

4 SimpleConnection 用のプロパティ・インスペクタで以下のパラメータを入力する。

Application Directory テキスト・ボックスで、ステップ 2 で作成したアプリケーションの URI、例えば、`rtmp:/lobby_com_test` を入力する。

Communication Components テキスト・ボックスをダブル・クリックする。現れる値ダイアログ・ボックスで、+ 印をクリックし、RoomList コンポーネントのインスタンス名 `roomList_mc` を入力し、OK をクリックする。

5 RoomList コンポーネントをステージ上にドラッグする。

6 RoomList コンポーネント用のプロパティ・インスペクタで、下記の値を入力する。

インスタンス名に `room_list_mc` をつける。この値は部屋アプリケーションの `main.asc` ファイル内で使用される。

Room Application Path テキスト・ボックスで、部屋アプリケーションの相対位置を入力する。例えば、`./chat_room_test/chat_room_test.html` を入力する。これは `chat_room_test fla` をパブリッシュしたときに生成される部屋アプリケーションの HTML ファイル名。

7 `lobby_com_test fla` として、web サーバーのアプリケーション・ディレクトリに保存し、パブリッシュする。

8 `lobby_com_test` アプリケーション・ディレクトリで、`main.asc` を作成し、以下のコードを記述する。

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

ロビー・アプリケーションの行き先として機能する部屋アプリケーションを作成する

1 “チャットコンポーネントを使う”で作成した `chat_test` アプリケーションをコピーする。Web サーバーのアプリケーション・ディレクトリ名を `chat_room_test` に変え、コピーした FLA ファイル名を `chat_room_test fla` に変える。

2 `chat_room_test fla` ファイルで、ステージ上の Simple Connect コンポーネントを選択し、Application Directory の設定を `rtmp:/chat_room_test` に変更する。

3 SimpleConnect 用のプロパティ・インスペクタで、コンポーネント・インスタンス名を `connector_mc` にする。この値は行き先の部屋の `main.asc` ファイルからコールされる。

4 ファイルを `chat_room_test` アプリケーション・ディレクトリに保存しパブリッシュする。アプリケーションをパブリッシュすると、Flash MX は `chat_room_test.html` ファイルを生成する。

5 HTML エディタで chat_room_test.html を開き、<BODY>...</BODY>タグ内を全て中に収まるように置き換える。

```
<BODY bgcolor="#FFFFFF">
<script language=JavaScript1.1>
<!--
    var appURL = String(document.location);
    if (appURL.indexOf("?") != -1){
        var appParams = appURL.substr(appURL.indexOf("?"));
    }else{
        var appParams = "";
    }
    document.write('<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000">');
    document.write('codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#versi
on=6,0,0,0" ');
    document.write(' WIDTH="550" HEIGHT="400" id="chat_room_test" ALIGN="">');
    document.write('<PARAM NAME=movie VALUE="chat_room_test.swf' + appParams + ">');
    document.write('<PARAM NAME=quality VALUE=high> <PARAM NAME=bgcolor VALUE=#FFFFFF>');
    document.write('<EMBED src="chat_room_test.swf' + appParams + " quality=high bgcolor=#FFFFFF');
    document.write(' swLiveConnect=FALSE WIDTH="550" HEIGHT="300" NAME="chat_room_test"
ALIGB=""');
    document.write('TYPE="application/x-shockwave-flash"
PLUGINSOURCE="http://www.macromedia.com/go/getflashplayer">');
    document.write('</embed>');
    document.write('</object>');
    //-->
</script></BODY>
```

*Note:*HTML は URL からパラメータを引き出し、部屋アプリケーションに渡す。これは web サーバーから、例えば <http://localhost> で、ページを見たときのみ機能する。またエレメントの値は行き先の部屋アプリケーションと、SWF ファイル名に合っていないと機能しない。前の HTML では、例えば chat_room_test と chat_room_test.swf という風に。Flash MX からまたムービーをパブリッシュしたら、この HTML ファイルは上書きされてしまう。

6 次に、chat_room_test アプリケーションの main.asc ファイルに、ロビー・アプリケーションに行き先へ向かわせるコードを加える。

Note: chat_room_test アプリケーション・ディレクトリの main.asc ファイルは、すでに以下のサーバー・サイド ActionScript を含んでいる。

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

chat_room_test アプリケーションの main.asc ファイル内に、ユーザをクライアントのグローバル・リストに加え、ユーザの接続を許可し、部屋インスタンス名を取得、部屋数を更新する onConnect ハンドラを記述する。

```
application.onConnect = function(newClient,username,password){  
    gFrameworkFC.getClientGlobals(newClient).username = username;  
    application.acceptConnection(newClient);  
    if(this.name.indexOf("/") != -1){  
        newClient.room = this.name.substr(this.name.lastIndexOf("/")+1);  
        roomConnect(newClient);  
    }  
}
```

7 roomConnect コールの結果を受け取り、結果を SimpleConnect コンポーネントに渡す roomResult ハンドラを記述する。

```
function roomResult(newClient){  
    this.onResult = function(roomName){  
        newClient.call("FCSimpleConnect/connector_mc/roomName",null,roomName);  
    }  
}
```

8 ロビーに接続し、ユーザが選択した部屋名を取得、戻り値で roomResult をコールする。

```
function roomConnect(newClient,room){
    lobby_nc = new NetConnection();
    lobby_nc.onStatus = function(infoStatus){
        if(infoStatus.code == "NetConnection.Connect.Success"){
            lobby_nc.call("FCRoomList/roomlist_mc/roomConnect",new
                roomResult(newClient),newClient.room);
        }
    };
    lobby_nc.connect("rtmp://localhost/lobby_com_test");
}
```

9 ロビー・アプリケーションに通知する onDisconnect ハンドラを記述する。

```
application.onDisconnect = function(client){
    if(client.room != null){
        roomDisconnect(client.room);
    }
}
```

10 部屋数を合わせ、表示からユーザを消す。

```
function roomDisconnect(room){
    lobby_nc = new NetConnection();
    lobby_nc.onStatus = function(infoStatus){
        if(infoStatus.code == "NetConnection.Connect.Success"){
            lobby_nc.call("FCRoomList/roomlist_mc/roomDisconnect",null,room);
        }
    }
    lobby_nc.connect("rtmp://localhost/lobby_com_test");
}
```

11 main.asc を保存し、Communication App Inspector からアプリケーションをリロードする

RoomList コンポーネントをテストする

1web ブラウザで、lobby_com_test.html を開き、ログインする。

2 Create Room を選択し、部屋リストに部屋を加える。

3 作成した部屋をクリックし、Join Room をクリックする。

4 チャットができ、文字色も変更できる。

5 lobby_com_test アプリケーションの別のインスタンスを開き、大勢のユーザでチャットしているかのように、別の名前でログインする。

RoomList コンポーネント・プロパティ・インスペクタ

名前	変数	説明
----	----	----

Room Application Path	roomPath	ストリング；部屋に加わったとき出すアプリケーションへのパスを与える。パスは絶対でも相対でもかまわない。
-----------------------	----------	---

RoomList コンポーネントのリスキニング

このコンポーネントのアセットは、ライブラリ・パネルの Core Assets-Developer Only\RoomList Assets ディレクトリにあります。

関係するコンポーネント

“SimpleConnect コンポーネント” 6 4 p

FCRoomList オブジェクト

FCRoomList API は、サーバーに接続したりクリーンアップしたりするオブジェクトに、クライアント・サイドとサーバー・サイドの機能を提供します。またプログラミングで、部屋を操作することもできます。

FCRoomList オブジェクトのメソッド・サマリー

メソッド	説明
FCRoomList.connect	クライアント・サイドとサーバー・サイドでコンポーネントが必要とするアセットを全てセットアップし、サーバーが与える名前で setUsername メソッドをコールします。
FCRoomList.setUername	表示し、サーバーに渡す名前を設定します。
FCRoomList.close	クリーン・アップしコンポーネントを切断します。
FCRoomList.createRoom	Create Room ダイアログ・ボックスを開き、一時的に Join、Create、Delete の各ボタンを使用不可にします。
FCRoomList.deleteRoom	現在選択されている部屋を消去するよう、サーバーにコールします。
FCRoomList.joinRoom	新しいブラウザ・ウィンドウを出し、選択された部屋のインスタンスを開きます。

FCRoomList.connect

利用

Flash Player 6

Flash Communication Server MX

使い方

roomList_mc.connect(nc)

パラメータ

nc アクティブな NetConnection オブジェクト

戻り値

なし

説明；

メソッド；クライアント・サイドとサーバー・サイドでコンポーネントが必要とするアセットを全てセットアップし、サーバーが与える名前ですetUsername メソッドをコールする。

FCRoomList.setUsername

利用

Flash Player 6

Flash Communication Server MX

使い方

roomList_mc.setUsername(newName)

パラメータ

newName 新しいユーザを特定するオプションのストリング。*NewName* が null または undefined でない場合、ユーザは部屋のリストを見、参加、作成、消去はできる。*newName* が null の場合は、何も変化せずユーザは lurker mode に入る。

戻り値

なし

説明；

メソッド；表示し、サーバーに渡すユーザ名を設定する。

FCRoomList.close

利用

Flash Player 6

Flash Communication Server MX

使い方

roomList_mc.close()

パラメータ

なし

戻り値

なし

説明；

メソッド；クライアント上でこのコンポーネントが使用していたアセットを解放し、サーバーからコンポーネントを切断することで、クリーンアップする。

FCRoomList.createRoom**利用**

Flash Player 6

Flash Communication Server MX

使い方

roomList_mc.createRoom()

パラメータ

なし

戻り値

なし

説明；

メソッド； Create Room ダイアログ・ボックスを開き、一時的に Join、Create、Delete の各ボタンを使用不可にする。

FCRoomList.deleteRoom**利用**

Flash Player 6

Flash Communication Server MX

使い方

roomList_mc.deleteRoom()

パラメータ

なし

戻り値

なし

説明；

メソッド；現在選択されている部屋を消去するよう、サーバーにコールする。

FCRoomList.joinRoom

利用

Flash Player 6

Flash Communication Server MX

使い方

roomList_mc.joinRoom()

パラメータ

なし

戻り値

なし

説明；

メソッド； 新しいブラウザ・ウィンドウを出し、選択された部屋のインスタンスを開く。アプリケーション・インスタンスとユーザ名は、パラメータとして新しいウィンドウに送られる。

CHAPTER 7

アプリケーション・サーバーの接続

このチャプターでは、Macromedia Flash Remoting サービスを使って、Flash Communication Server MX 1.5 にアプリケーション・サーバー接続を加える方法を説明します。Macromedia Flash Remoting は J2EE アプリケーション・サーバーや Microsoft Windows .NET サーバーにコミュニケーション・アプリケーションを接続させるゲートウェイです。Flash Remoting は、Macromedia Flash MX ムービーをサーバー・サイド・ゲートウェイにリンクさせる、NetServices という、クライアント・サイドの ActionScript ライブラリに依存します。Flash Remoting スクリプト・ファイルは、(既定で)scriptlib ディレクトリにあります。Scriptlib ディレクトリに関する詳細は、Flash Communication Server が使用する File タイプを参照してください。Netservices API は、*Using Flash Remoting* に記述されており、Macromedia Support Center サイト (www.macromedia.com/go/flashremoting)で PDF ファイルとして提供されています。

このチャプターのサンプルでは、サーバー・サイド・コミュニケーション ActionScript でどのようにして Flash Remoting をコールし、NetServices にアクセスするかを説明します。この状況では Flash Communication Server は Flash remoting のクライアントとしてふるまいます。これらのサンプルは、Flash Remoting を含む Macromedia ColdFusion MX とともに動作します。他のアプリケーション・サーバーと Flash Remoting を使用するには、Flash Remoting サイト (www.macromedia.com/go/flashremoting)を参照してください。

Flash Remoting を通して接続する

外部のデータ・ソースに接続には、Flash Communication Server を使用してクライアントとサーバーに接続する異なる 2 通りの方法があります。サブセクションの“外部ソースへの接続”では、Flash Communication Server が他のサービスに接続するのを示します。クライアントは Flash Communication Server 上でメソッドを呼び出し、その結果 Flash Communication Server は追加されたサーバー上でメソッドを呼び出します。結果はアプリケーション・サーバーから Flash Communication Server へ移動し、最後にクライアントに戻ります。

Flash Communication Server を Flash Remoting NetServices ライブラリと併用し、いくつかのパラメータを渡して、クライアントから Flash Communication Server メソッドを呼び出します。Flash Communication Server は、その後 NetServices メソッドを呼び出します。NetServices メソッドは Flash Communication Server にその結果を渡し、サーバーはクライアントにそれを戻します。

サンプルの再作成

サンプルは、Flash Communication Server と ColdFusion MX が動作する同じローカル・ホスト上で、Flash MX オーサリング環境があるものと仮定しています。サンプル関連ファイル(FLA、SWF、HTML、ASC)は、インストールされた Macromedia Flash MX オーサリング・ディレクトリ下か Flash Communication Server インストール・ディレクトリの /flashcom_help/help_collateral ディレクトリのサブディレクトリにあります。

初めのチャプターでサンプルを作成したように、クライアント・サイド・コミュニケーション ActionScript を使用して FLA ファイルにユーザー・インターフェイスを作成します。FLA ファイルのクライアント・サイド ActionScript は、また作成するサーバー・サイド ASC ファイル内でメソッドを呼び出します。

アプリケーションに Flash Remoting サービスを加えるには、作成する ASC ファイル内にサーバー・サイド ActionScript を使用します。サーバー・サイド・コードは、また作成する ColdFusion CFC ファイル内でメソッドを呼び出すことで、Flash Remoting へのクライアントとしてふるまいます。

アプリケーション・ディレクトリに FLA と ASC ファイルを配置し、同じディレクトリに NetServices を可能にする netservices.asc ファイルを置きます。他の ColdFusion スクリプト・ファイルを置いた既定のディレクトリに、CFC か CFM ファイルを配置します。

最後に、SWF ファイルをパブリッシュし動作させてムービーをテストします。テストする前に、Flash Communication Server と ColdFusion MX が作動していることを確認してください。

Note: サンプルは RTMP を使用して、アプリケーション・サーバーに接続します。安全な接続を使って接続したい場合は、HTTPS を使用します。Macromedia RTMP を使用した、サーバーからサーバーへの接続は不可能です。

サンプル 1：簡単なリモートリング

このサンプルでは、クライアント・アプリケーションが計算を求め、アプリケーション・サーバー上でなされた計算の結果がストリング型で戻されます。クライアントは Flash Communication Server にコールを呼び出します。その結果、Flash Communication Server は、Flash Remoting サービスのクライアントとしてふるまいます。ここでは、CFC スクリプトであり、バックエンドのサーバーによってなされる作業と結果を戻すよう求めるスクリプトです。

サンプルについて

ユーザーが Run Test ボタンをクリックすると、画面は Boolean 値、月の配列、追加操作といった、サーバーからのメッセージで更新されます。

サンプルの再作成

doc_remoting fla ファイルは、クライアント、Flash Communication Server、ColdFusion MX Server の間で接続性をテストするユーザー・インターフェイスを提供します。main.asc ファイルは、クライアントからの接続を許可し、その結果、Flash Remoting を使って ColdFusion に接続します。simple.cfc ファイルは、Boolean 値、配列、合計を取得するスクリプトを含みます。

サンプルを再作成する前に、Creating your working environment を参照してください。

サンプルのユーザー・インターフェイスを作成する

1. ツールボックスからテキスト・ツールを選択し、ダイナミック・テキスト・ボックスを描きます。プロパティ・インスペクタ(ウィンドウ>プロパティ)で、テキスト・ボックスのタイプにダイナミック・テキストを選択し、インスタンス名を ResultBox にします。
2. テスト動作用のボタンを追加するには、コンポーネント・パネル(ウィンドウ>コンポーネント)を開き、ステージ上に PushButton をドラッグします。プロパティ・インスペクタで、インスタンス名を Run_btn、ラベルを Run Tests、click handler を runTests に設定します。
3. doc_remoting fla として保存します。
4. Flash Communication Server アプリケーション・ディレクトリに doc_remoting という名前のディレクトリを作成します。

サンプルのクライアント・サイド ActionScript を書く

1. Flash Communication Server から帰ってくるステータス情報をトレースする以下のデバッグコードを加えます。

```
function traceStatus(info) {  
    trace("Level: " + info.level + " Code: " + info.code);  
}  
  
NetConnection.prototype.onStatus = traceStatus;  
NetStream.prototype.onStatus = traceStatus;  
SharedObject.prototype.onStatus = traceStatus;
```

2. Run_btn のイベント・ハンドラを作成します。RunTests へのコールに注目してください。これは後でサーバー・サイド・コールに定義する関数を呼び出します。2 つめのパラメータ new Result()は、runTests へのコールの結果を受け取る機能を提供します。

```
function runTests() {  
    function Result() {  
        this.onResult = function(str) {  
            _root.ResultBox.text = str;  
        }  
    }  
  
    main_nc.call("runTests", new Result());  
}
```

3. 新しいネットワーク接続を作成し、アプリケーションに接続します。

```
main_nc = new NetConnection();  
main_nc.connect("rtmp://doc_remoting/room_01");
```

4. Flash Communication Server がテストの結果を受け取るためにコールするハンドラ機能を与えます。

```
NetConnection.prototype.postResults = function(result) {  
    _root.ResultBox.text += "¥n" + result;  
}
```

サンプルのサーバー・サイド ActionScript を書く

1. サーバー・サイド ActionScript エディターを使って、新しいファイルを作成し、Flash Remoting が Flash Communication Server を通して使用可能になる、netservices.asc ファイルを書きます。

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

2. onAppStart 関数で、Flash Remoting サービスを場所を定義し、接続します。

```
application.onAppStart = function() {  
    trace("***** on app start");  
    NetServices.setDefaultGatewayUrl("http://localhost:8500/flashservices/gateway");  
    this.gatewayConnection = NetServices.createGatewayConnection();  
}
```

3. onConnect関数で、クライアントの接続を許可します。

```
application.onConnect = function (clientObj) {  
    trace("***** on connect");  
    this.acceptConnection(clientObj);  
    return true;  
}
```

4. クライアントがコールしてテストを呼び出せるプロトタイプ関数を与えます。

```
Client.prototype.runTests = function() {  
    trace("***** runTests");
```

5. 同じ関数で、Flash Remoting サービスのインスタンス、testServiceを作成します。2つめのパラメータ、new TestResult(this)は、このファイルの下の方で定義するハンドラで、Flash Remotingサービスから返ってくるデータをキャッチします。

```
var testService = application.gatewayConnection.getService("simple.simple",new TestResult(this));
```

6. testServiceメソッドをコールし、Flash Remoting メソッドのgetBoolean、getArray、addNumbersからの結果を取得し、関数を閉じます。

```
testService.getBoolean(true);  
testService.getArray();  
testService.addNumbers(1,3);  
return "Ran the service on the server";  
}
```

7. Flash Remotingサービスから返ってくるデータに、ハンドラとしてTestResult関数を与えます。

```
function TestResult(client) {  
    this.client = client;  
}
```

- 各テスト用に、結果resultを受けます。その後クライアント定義関数postResultsをコールし、クライアントにresultデータを渡します。

```
TestResult.prototype.getBoolean_Result = function(result)
{
  trace("***** getBoolean_Result: " + result);
  this.client.call("postResults", null, result);
}

TestResult.prototype.addNumbers_Result = function(result) {
  trace("***** addNumbers_Result: " + result);
  this.client.call("postResults", null, result);
}

TestResult.prototype.getArray_Result = function(result) {
  trace("***** getArray_Result: " + result);
  this.client.call("postResults", null, result);
}
```

Note: このチャプターのサンプルでは、main.ascファイル内にnetservices.ascを含むことにより、リモート・サービスを読み込んでいます。main.ascファイルにもリモート・サービスから返ってくる結果値を管理するresultメソッドを与える必要があります。このチャプターのサンプルはFlash Remotingから[methodName]_Resultコールバックの慣例を使用しています。例えば、getBooleanをコールするには、getBoolean_Resultメソッドにコールバックさせています。

- /applications/doc_remotingディレクトリにmain.ascファイルとして保存します。

サンプルのColdFusionコンポーネントを書く

- 標準的なエディターを使って、新しいファイルを作成し、アプリケーションに名前をつけ、アクセス方法を指示するコードを加えます。

```
<cfcomponent name="simple" access="remote">
```

- Flash Communication Serverが呼び出すaddNumbersメソッドを作成します。

```
<cffunction name="addNumbers" output="false"
description="add two numbers" access="remote">
<cfargument name="num1" required="true" type="numeric"
description="the first number">
<cfargument name="num2" required="true" type="numeric"
description="the second number">
<cfreturn num1+num2>
</cffunction>
```

- Flash Communication Serverが呼び出すgetBooleanメソッドを作成します。

```
<cffunction name="getBoolean" output="false" description="Returns a
```

```
boolean that you specify." access="remote">
<cfargument name="bool" required="true" type="boolean"
description="The boolean to return">
<cfreturn bool>
</cffunction>
```

4. Flash Communication Serverが呼び出すgetArrayメソッドを作成します。

```
<cffunction name="getArray" output="false" description="Creates and
returns an array of 3 items" access="remote">
<cfset months = ArrayNew(1)>
<cfset months[1] = "January">
<cfset months[2] = "February">
<cfset months[3] = "March">
<cfreturn months>
</cffunction>
</cfcomponent>
```

5. ファイルをsimple.cfcとして、ColdFusion MXファイルをパブリッシュするディレクトリ下のsimpleディレクトリに保存します。

サンプル 2 : メールを送る

このサンプルでは、多用途のCFMAILタグを用いてメールを送る方法を説明します。Flash Communication Serverデータを配信する際このタグは非常に重要なものとなります。このサンプルの再作成をする前に、ColdFusionでのメール・サーバーの構成を確認してください。確認するには、サーバーの設定> メールサーバーを選択し、使用したいSMTPサーバーのIPアドレスを指定します。

サンプルについて

ユーザーは、To、From、Subject、message本文の4つのフィールドに記入します。ユーザーがSendボタンをクリックすると、Flash Playerが、ユーザーの4つのフィールドに入力した内容をパラメータとして、Flash Communication ServerにsendMailコマンドを送ります。その後Flash Communication Serverは、メールを送るようFlash Remotingサービス上でコールします。

サンプルの再作成

doc_sendmail flaファイルは、メール・メッセージを送信するシンプルなユーザー・インターフェイスを提供します。クライアント・サイドActionScriptはメッセージを送信するよう要求しますが、クライアントに結果は返されません。

サンプルを再作成する前に、Creating your workng environmentを参照してください。

サンプルのユーザー・インターフェイスを作成する

1. Flash MXオーサリング環境で、ファイル>新規を選択し、新しいファイルを開きます。
2. ツールボックスからテキストツールを選択し4つのテキスト・ボックスを描きます。
3. テキスト・ボックスそれぞれに、プロパティ・インスペクタ(ウィンドウ>プロパティ)で、テキスト・ボックスの種類からテキスト入力を選択し、インスタンス名、FromText、ToText、SubjectText、BoduTextを入力します。

4. メールを送信するボタンを加えるには、コンポーネント・パネル(ウィンドウ>コンポーネント)からステージ上に PushButton をドラッグします。プロパティ・インスペクタでインスタンス名 Send_btn とラベル名 Send、click handler 名 sendMail を設定します。
5. ファイルを doc_mailto fla として保存します。
6. Flash Communication Server アプリケーション・ディレクトリに doc_mailto と名づけたディレクトリを作成します。

サンプルのクライアント・サイド ActionScript を書く

1. Flash Communication Server から帰ってくるステータス情報をトレースする以下のデバッグコードを加えます。

```
function traceStatus(info) {  
    trace("Level: " + info.level + " Code: " + info.code);  
}
```

```
NetConnection.prototype.onStatus = traceStatus;
```

```
NetStream.prototype.onStatus = traceStatus;
```

```
SharedObject.prototype.onStatus = traceStatus;
```

2. send_btn のイベント・ハンドラを作成します。sendMail へのコールに注目してください。これは後でサーバー・サイド・コールに定義する関数を呼び出します。2 つめのパラメータ null は、Flash Communication Server にクライアントに結果を返さなくてもよいことを伝えます。

```
function sendMail() {  
    main_nc.call("sendMail", null, ToText.text, FromText.text,  
    SubjectText.text, BodyText.text);  
}
```

3. 新しいネットワーク接続を作成しアプリケーションに接続します。

```
main_nc = new NetConnection();
```

```
main_nc.connect("rtmp://doc_mailto/room_01");
```

サンプルのサーバー・サイド ActionScript を書く

1. サーバー・サイド ActionScript エディターを使用して新規ファイルを作成し、Flash Communication Server を通して Flash Remoting の使用を可能にする netservices.asc ファイルをロードします。

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

2. onAppStart 関数で、Flash Remoting サービスの場所を定義し、接続します。

```
application.onAppStart = function() {  
    trace("***** on app start");  
    NetServices.setDefaultGatewayUrl("http://localhost:8500/flashservices/gateway");  
    var gatewayConnection = NetServices.createGatewayConnection();  
    this.mailSendService = gatewayConnection.getService("mail.sendMail", this);  
}
```

3. onConnect関数で、クライアントの接続を許可します。

```
application.onConnect = function (clientObj) {  
    trace("***** on connect");  
    this.acceptConnection(clientObj);  
    return true;  
}
```

4. クライアントがメールを送るコールをプロトタイプ関数に与えます。

```
Client.prototype.sendMail = function(to, from, subject, body) {  
    trace("***** sending mail");  
    application.mailSendService.sendMail(to, from, subject, body);  
}
```

5. ファイルを/applications/doc_mailtoディレクトリにmain.ascとして保存します。

サンプルのColdFusionコンポーネントを書く

1. 標準のエディターを使用し、新規ファイルを作成、アプリケーションの名前をつけアクセス方法を指示するコードを加えます。

```
<cfcomponent name="sendMail" access="remote">
```

2. sendMail関数を作成し、各パラメータを定義します。

```
<cffunction name="sendMail" output="false"  
description="send mail to a client" access="remote">  
<cfargument name="to" required="true" type="string"  
description="recipient of the email">  
<cfargument name="from" required="true" type="string"  
description="sender of the email">  
<cfargument name="subject" required="true" type="string"  
description="subject heading">  
<cfargument name="body" required="true" type="string"  
description="body text of the email">
```

3. メッセージを送信するColdFusionメソッドcfmailをコールします。

```
<cfmail to = "#to#" from = "#from#" subject = "#subject#">#body#</cfmail>  
<cftrace category="UDF End" inline = "True"  
text = "Email was sent" var = "MyStatus">  
</cffunction>  
</cfcomponent>
```

4. ファイル名をsendmail.cfcとし、ColdFusion MXをパブリッシュするディレクトリ下のmailアプリケーション・ディレクトリに保存します。

サンプル3：レコードセット

データベースへ簡単にアクセスできると、Flash Communication Serverにデータの保存や検索を加えられます。このシンプルなテストは、データベースから全レコードセットを検索します。作業はCFMファイル内部の1つのSQLステートメントが行います。

サンプルについて

ユーザーがGet Recordsボタンをクリックすると、データベースにある全データがリストボックスに表示されます。

サンプルの再作成

doc_rset flaファイルは、ユーザーにただGet Recordsボタンを選択させ、レコードセットを返させるインターフェイスを提供します。

サンプルを再作成する前に、Creating your working environmentを参照してください。

サンプルのユーザー・インターフェイスを作成する

1. Flash MXオーサリング環境で、ファイル>新規を選択し新しいファイルを開きます。
2. レコードをリストする要素を加えるには、コンポーネント・パネル(ウィンドウ>コンポーネント)を開き、ステージ上にList Boxコンポーネントをドラッグします。そしてインスタンス名records_listをつけます。
3. サーバーに接続するボタンを加えるには、コンポーネント・パネルからステージ上にPushButtonをドラッグします。プロパティ・インスペクタで、インスタンス名GetRecs_btn、ラベルにGet Records、click handlerにdoGetRecordsを設定します。
4. doc_rset flaで保存します。
5. Flash Communication Serverアプリケーション・ディレクトリ内にdoc_rsetと名づけたディレクトリを作成します。

サンプルのクライアント・サイドActionScriptを書く

1. 新しいネットワーク接続を作成し、Flash Communication Serverに接続します。

```
nc = new NetConnection();
nc.onStatus = function(info) {
trace(info.level + ": " + info.code + " " + info.description);
}
```

```
nc.connect("rtmp://doc_rset/room_01");
```

2. 共有オブジェクトを取得し、共有オブジェクトからのデータで、リスト・ボックス(records_list)を更新します。

```
recset_so = SharedObject.getRemote("records", nc.uri, false);
recset_so.onSync = function(list) {
records_list.removeAll();
records_list.addItem(getColumnString(this.data.__COLUMNS__), -1);
```

3. 同じメソッド内で、getRecordStringへのコールから返ってくるデータに、recstrの変数名をつけ、データでリスト・ボックスをポピュレートします。

```
for (var i in this.data) {  
    if (i == "__COLUMNS__")  
        continue;  
    var recstr = getRecordString(i,this.data[i], this.data.__COLUMNS__);  
    records_list.addItem(recstr, i);  
}  
records_list.sortItemsBy("data", "ASC");  
}
```

4. 共有オブジェクトに接続します

```
recset_so.connect(nc);
```

5. 全コラム・データを取得するgetColumnString関数を作成します。

```
function getColumnString(cols) {  
    var colstr = "";  
    if (cols.length > 0)  
        colstr += cols[0];  
    for (var i = 1; i < cols.length; i++)  
        colstr += " " + cols[i];  
    return colstr;  
}
```

6. getRecordString関数を作成します。

```
function getRecordString(inx, recs, cols) {  
    var recstr = "";  
    if (cols.length > 0)  
        recstr += recs[cols[0]];  
    for (var i = 1; i < cols.length; i++)  
        recstr += " " + recs[cols[i]];  
    return recstr;  
}
```

7. GetRecs_btnボタンのイベント・ハンドラを作成します。ユーザーがボタンをクリックすると、doGetRecordsメソッドがコールされます。この関数は、サーバー・サイドActionScriptで定義されるgetRecordsコールを含みます。

```
function doGetRecords() {  
    nc.call("getRecords", null);  
}
```

サンプルのサーバー・サイドActionScriptを書く

1. サーバー・サイドActionScriptエディターを使用して新規ファイルを作成、Flash Communication Serverを通してFlash Remotingを使用可能にするnetservices.ascファイルをロードします。

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

2. records共有オブジェクトを取得します。

```
gRecords = SharedObject.get("records", false);
```

```
NetServices.setDefaultGatewayUrl("http://localhost:8500/flashservices/gateway");
```

```
var gRemotingGateway=NetServices.createGatewayConnection();
```

3. レコードセットを保持するグローバル・オブジェクトを作成します。

```
gFoo = {};
```

4. fooサービスへの参照を取得します。

```
gFoo.service = gRemotingGateway.getService("foo", gFoo);
```

5. Flash Remotingサービスが求める、onResultコールバックを作成します。

```
gFoo.bar_Result = function(result) {
```

```
var cols = result.getColumnNames();
```

```
trace("Columns: " + cols);
```

```
gRecords.setProperty("__COLUMNS__", cols);
```

```
var reclen = result.getLength();
```

```
trace("number of records " + reclen);
```

```
for (var i = 0; i < reclen; i++) {
```

```
trace(i + "] " + result.getItemAt(i));
```

```
gRecords.setProperty(i, result.getItemAt(i));
```

```
}
```

```
}
```

6. onConnect関数内で、クライアントの接続を許可します。

```
application.onConnect = function(client) {
```

```
trace(application.name + " connect from " + client.ip);
```

```
application.acceptConnection(client);
```

```
}
```

7. クライアントがレコード取得をコールするプロトタイプ関数を与えます。

```
Client.prototype.getRecords = function() {
```

```
var result = gFoo.service.bar();
```

```
trace("gFoo.service.bar returned " + result);
```

```
}
```

8. main.ascファイル名で/application/doc_rsetディレクトリに保存します。

サンプルのColdFusionコンポーネントを書く

1. 標準的なエディターを使って、新規ファイルを作成し、全レコードを照会するコードを加えます。

```
<cfquery name="flash.result" datasource="ExampleApps">
```

```
SELECT * FROM tblItems
```

```
</cfquery>
```

2. ファイルをbar.cfm名で、ColdFusion MXファイルのパブリッシュするディレクトリ下のfooディレクトリに保存します。

APPENDIX

MP3 ファイルを使った作業

MP3ファイルの再生

Macromedia Flashファイルで、既定の再生形式はFLVです。アプリケーション内で、クライアント・サイドActionScriptを使って、MP3オーディオ・ファイルとMP3ファイルのID3タグを再生し、サーバー・サイドActionScriptを使ってストリーム上にMP3ファイルをパブリッシュできます。

そのためには、登録したアプリケーション・ディレクトリ内の`/streams/application_instance/`サブディレクトリにアプリケーションで使うMP3ファイルをアップロードします(Macromedia Flash Communication Sever MX 1.5は、ストリームを記録すると、`/streams`サブディレクトリを作成します。`/streams`サブディレクトリが存在しない場合は、手作業で作成できます)。例えば、Flash Communication Severアプリケーション・ディレクトリ内にCDPlayerAppという名前のアプリケーションがあったなら、`/application/CDPlayerApp/streams/application_instance`にそのアプリケーションのMP3ファイルをアップロードします。

ディレクトリに共有MP3ファイルを置き、アプリケーションが使用するVhost.xmlファイルの`<Streams>`タグに、仮想ディレクトリとして、この共有ディレクトリの場所を指定することで、全てのアプリケーション・インスタンスでMP3ファイルを共有することができます。その後、`Stream.play`ステートメントで、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>`タグに関しては、*Managing Flash Communication Server*の“The Vhost.xml file”を参照してください。

MP3ファイルを再生するには、VideoオブジェクトにNetStreamオブジェクトをアタッチし、`play`メソッドをコールするか、MovieClipオブジェクトにアタッチして`attachAudio`メソッドをコールするかします。パラメータで`streamname`を指定し、再生するものに関しては、`mp3:`をMP3ファイルの前に書かなくてはなりません。以下のサンプルは、ネットワーク・ストリーム`mystream`でファイル`bolero.mp3`を再生する2つの方法を示します。

```
vidObj.attachVideo(mystream);
mystream.play("mp3:bolero");
movieObj.attachAudio(mystream2);
mystream2.play("mp3:mp3dir/bolero");
```

Tip: `NetStream.play`ステートメントで既定のビデオ/オーディオ・ファイル形式のFLVを明示しなかったとしても、MP3ファイルは常に指定しなければなりません。つまり、“`flv:granada`”と“`granada`”は両方とも`granada.flv`ファイルを再生します。“`mp3:bolero`”はファイル`bolero.mp3`を再生します。

MP3のID3タグを再生するには、id3:をストリーム名の前に書きます。そしてID3データをキャプチャーするコールバック関数を定義します。例えば、bolero.mp3のID3タグを表示するには、

```
mystream.play("id3:bolero");  
mystream.onId3 = function(info){  
  for (i in info){  
    trace(i + ":" + info[i]);  
  }  
}
```

サポートされるID3タグのバージョンについて

Flash Communication Serverは、UTF-8、UTF-16、ISO-8859-1形式でID3テキスト・タグの再生をサポートし、ID3バージョン1.0、2.3、2.4をサポートします。歌の題名、歌手名、コメント、録音年など、テキスト・データを含むタグのみをサポートします。

サーバー・サイド・コミュニケーションActionScriptを使用する

サーバー・サイドのStreamオブジェクトのメソッドを使用して、MP3ファイルを再生(.play)したり、MP3ファイルの長さを取得(.length)できます。MP3ファイルで使うメソッドに関する情報は、*Server-Side Communication ActionScript Dictionary*のStream.play項、Stream.length項を参照してください。

サーバー・サイドActionScriptを使ってMP3ファイルを削除するには、Application.clearStreamsメソッドを使用します。

詳細は、*Server-Side Communication ActionScript Dictionary*のApplication.clearStreams項を参照してください。

ストリーム上にMP3ファイルを、またはMP3ファイルに関連するID3タグ情報をパブリッシュするには、以下のサンプルのように、Stream.playメソッドを使用します。

```
application.myStream = Stream.get( "music" );  
if (application.myStream)  
{  
  application.myStream.play("mp3:bolero", 0, -1);  
}
```

Stream.playメソッドを使ってID3タグのテキストをキャプチャーし再生するには、以下のようにします。

```
application.myStream = Stream.get( "description" );
```

```
application.myStream.onId3 = function(info)
```

```
{
```

```
  for (i in info)
```

```
  {
```

```
    trace(i + ": " + info[i]);
```

```
  }
```

```
}
```

```
if (application.myStream)
```

```
{
```

```
  startTime parameter
```

```
  application.myStream.play("id3:bolero", 0, -1);
```

```
}
```


GLOSSARY

Flash Communication Server 用語

アプリケーション・サーバー サーバー・サイド・スクリプトやタグを含むWebページを、webサーバーが処理するのを助けるソフトウェア。そういったページが要求されると、ブラウザにページを送る前に、webサーバーは処理をするアプリケーション・サーバーにページを送ります。

よく知られるアプリケーション・サーバーは、Macromedia ColdFusion MX Server、Macromedia Jrun、Microsoft.NET Framework、IBM WebSphere、Apache Tomcatなどです。

コミュニケーション・コンポーネント サーバーで使用する機能を供給するMacromedia Flash Communication Server 1.5をインストールすると、Macromedia Flash MXオーサリング環境に加えられる、ドラッグ・アンド・ドロップのインターフェイス要素。

HTTP(ハイパーテキスト・トランスファー・プロトコル) World Wide Web上のサーバーに接続するために使用されるコミュニケーション・プロトコル。HTTPの主な機能は、webサーバーとの接続を確立させ、クライアント・ブラウザにHTMLページを送ることです。

NetConnection オブジェクト Flash Playerにサーバー上のアプリケーションに接続するよう伝えるオブジェクト。

リアル・タイム・メッセージング・プロトコル(RTMP) Flash PlayerクライアントとFlash Communication Server間での双方向コミュニケーションに、永続するソケット接続を提供するコミュニケーション・プロトコル。

共有オブジェクト ローカルやリモートにデータを保持するために使用されるオブジェクト。

ストリーム・オブジェクト Flash Communication Serverで各ストリーム管理を可能にさせるオブジェクト。

ストリーミング・オーディオ データ・ネットワーク上でのオーディオの連続的な送信。

ストリーミング・ビデオ データ・ネットワーク上でのビデオの連続的な送信。

thin client ローカルに保持せず、サーバーからプログラムをダウンロードするユーザーのコンピュータ。通常のコンピュータ処理を実行するが、データはサーバーに保存します。

web client webのクライアント側(ユーザー側)のこと。Webクライアントは、ユーザーのwebブラウザであり、プラグインであり、ブラウザをサポートするほかのアプリケーションでもあり得ます。

web server webブラウザからの要求に応じて、webページを送るソフトウェア。