Министерство образования и науки Российской Федерации

Федеральное агентство по образованию

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Д.Г. Николаев, Д.Г. Штенников

Web-программирование. Серверный ActionScript

Учебное пособие

Санкт-Петербург

2006

УДК 681.3

Николаев Д.Г., Штенников Д.Г. Web-программирование. Серверный ActionScript.
Учебное пособие. – СПб., 2006. - 116 с.

Рецензенты: Л.С. Лисицына, к.т.н., доцент, зав. каф. КОТ СПбГУ ИТМО,
А.А. Бобцов, к.т.н., доцент каф. СУиИ СПбГУ ИТМО

Учебное пособие подготовлено на кафедре «Компьютерные образовательные технологии» (КОТ) факультета ИТиП СПбГУИТМО и предназначено для студентов специальностей 230202 – «Информационные технологии в образовании» и 230201 – «Информационные системы и технологии», изучающих курс «Web-программирование», а также для разработчиков Интернет-ресурсов, слушателей курсов повышения квалификации учителей информатики и информационных технологий. Пособие служит руководством к выполнению лабораторных работ по курсу. Практикум формирует навыки разработки и использования серверных приложений, применяемых при создании мультимедийных и интерактивных образовательных интернет-ресурсов.

Печатается по решению УМС факультета ИТиП СПбГУ ИТМО, протокол № 2 от 26.09.2006.

Оглавление

Лабораторная работа №1: Создание приложения

В прошлой части методического пособия были рассмотрены вопросы, касаемые клиентского программирования на ActionScript. В данном пособии будет рассказано о применении серверного ActionScript'a, используемого для Flash Communication Server или Flash Media Server.

Macromedia Flash получило свое развитие как метод легкого создания и распространения простейшей анимации и графики в сети для платформ мощных приложений. По информации от Macromedia, Flash Player 6 уже в июне 2004 был доступен более чем 94 % автоматизированным рабочим местам в Канаде, Соединенных Штатах, и Европе с выходом в Интернет. С согласия пользователя, Flash movie может записывать в реальном времени аудио и видео от микрофона или от камеры и передавать это на Flash Communication Server MX (сервер связи).

Сервер может перераспределять потоки данных другим пользователям, имеющих Flash Player на своем компьютере. Связь в реальном времени делает возможным разрабатывать широкий диапазон приложений.

Функции Flash и Flash Communication Server MX могут быть использованы для создания:

Различных видео конференций, корпоративных конференций, и чатов с общими составляющими, такими как: общий чат, whiteboards, графиками.

Приложений с Video on demand и данных с пользовательскими интерфейсами, которые могут включать закрытые заголовки и средства управления оболочкой.

Прямой трансляции с помощью настраиваемого пользовательского взаимодействия, такого как небольшой чат и диалог в форме вопросов и ответов.

Многопользовательских игр, моделирования, и других приложений с добавлением аудио и видео, если необходимо.

Клиенты и Серверы

FlashCom- это серверное приложение, которое установлено на хосте машины подобно серверу сети; однако, FlashCom работает иначе. Вместо принятия множества локаничных соединений от браузеров, требующих web-страницу или другой ресурс, FlashCom принимает продолжительные подключения от Flash Movie, выполняемые во Flash Player. Каждый Flash Movie может совместно использовать данные с другим Flash Movie через сервер, используя Протокол Обмена сообщениями в реальном времени (RTMP). В отличие от модели HTTP (запрос/ответ), используемой браузерами, чтобы связаться с web- серверами, RTMP ведет подключение к FlashCom Серверу беспрерывно, таким образом, нет необходимости в специальных шагах для поддержки сеанса передачи информации. Как только сервер принимает подключение клиента, соединение может использоваться, для обмена аудио, видео, и ActionScript данными, пока или клиент или сервер не разъединяться.

Flash Player может выполняться в пределах Standalone Player или в пределах web-браузера. Flash Player (и любой Movie проигрывающий в его пределах) считается клиентом. FlashCom не может инициировать подключение к Movie, подключение должно быть начато от Flash Player'a, выполняющегося на стороне клиента. Архитектура клиент/сервер для приложений FlashCom показана на рисунке 1.

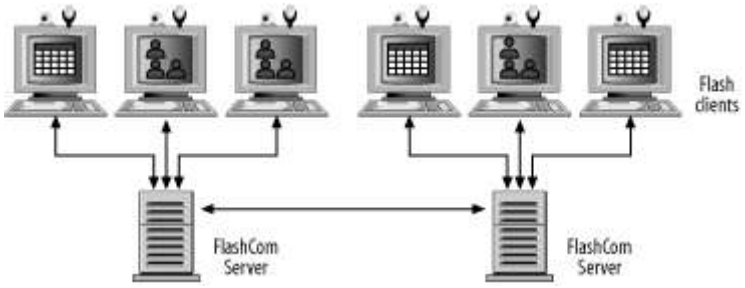

Рисунок 1. Архитектура клиент/сервер.

Через Web-браузер или через Flash Player загружаются файлы Flash Movie (файлы с расширением .swf), затем передаются .swf файл в Player, для его выполнения. Flash Movie обеспечивает пользовательский интерфейс и может попытаться соединиться через Player с любым FlashCom Сервером. После соединения, Flash Movie может поддерживать связь с сервером. Кроме того, он может связываться через сервер с Movie, играющими на других Flash клиентах. Flash Movie может передавать аудио и видео на FlashCom Сервер так, чтобы другие клиенты Flash с доступом к тому же самому серверу могли проиграть записи, сохраненные на сервере и живые потоки от других клиентов.

Живой поток - это тот поток, который выложен на сервер одним клиентом так, чтобы другие клиенты могли иметь доступ к нему. Таким образом, данные клиента прибывают на сервер, сервер дублирует их и пересылает каждому клиенту, где эти данные можно просмотреть и прослушать. Записанные потоки сохраняются на сервере, и могут проигрываться, начиная с любой точки в пределах потока, быть приостановлены, и перезапущены. Также возможно остановить записанный поток, найти любую точку в его приделах, и начать проигрывать снова.

Если множество FlashCom Серверов связаны друг с другом, клиенты, соединенные с одним сервером, могут связаться с клиентами с других серверов. Способность взаимодействия между серверами и клиентами, связанными с ними, делает возможным крупномасштабные приложения, типа прямых трансляций для многочисленной аудитории.

FlashCom может принимать много различных приложений. Несколько запросов приложения могут быть обработаны в одно и тоже время. Каждому запросу дают его собственное уникальное имя. И когда клиент соединяется с сервером, он всегда указывает имя запроса приложения. Например, множество отдельных запросов приложения, названных chatRoom могут быть доступными. Каждый запрос имеет свое собственное уникальное имя и может обеспечивать уникальные ресурсы для клиента.

Рисунок 2 иллюстрирует трех клиентов, связанных с одним и тем же самым запросом chatRoom приложения.

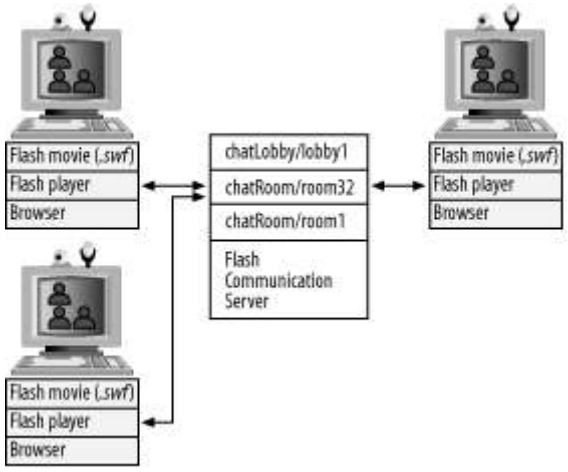

Рисунок 2. Клиенты и chat-room.

Создание Приложения

Благодаря природе приложений связи типа клиент/сервер (распределенная модель, в которой имеются два типа приложений: приложения-клиенты, посылающие запросы (или вызывающие соответствующие события)) разработчик обычно создает на стороне клиента Flash Movie, чтобы управлять пользовательским взаимодействием и отдельной серверной частью приложения FlashCom, с которой происходит соединение. На стороне клиента Flash Movie может быть написан на ActionScript 1.0, 2.0 или 3.0.

Серверное приложение FlashCom написанно на (SSAS), который подобен клиентскому ActionScript. Для создания приложения FlashCom сначала необходимо создать домашнюю директорию на сервере.

ActionScript файлы с исходным кодом должны быть помещены в домашнюю директорию приложения, которая дает каждому приложению его уникальный серверный режим.

Разработка Movie Flash почти всегда включает программирование с ActionScript, язык создания сценария Flash, основан на ECMA стандарте (подобно JavaScript). В дополнение к типичным объектам, ActionScript поддерживает специальный MovieClip тип данных. Movie Cliры – это основные блоки строительства для Flash анимации и основа для компонентов более высокого уровня типа Button, DataGrid, и Tree.

Для многих приложений с видео, Вы можете использовать заготовку FLVPlayer, который не требует никакой авторизации Flash, доступного на сайте:
`http://www.peldi.com/blog/FLVPlayer.html`

FLVPlayer's может быть настроен, для соединения с любым приложением и включения автоматической пропускной способности и выбора потока. Видео Player также доступен как часть Macromedia Video Kit:
`http://www.macromedia.com/software/studio/flashvideokit`

Вы можете создавать простые приложения соединения, такие как видео конференц-приложения используя заготовку компонентов типа SimpleConnect, PeopleList, и VideoConference компонентов в Macromedia Flash. А также Вы можете перетащить компоненты из панели Flash's Components на Сцену, чтобы создать интерфейс пользователя. При использовании панели Flash's Properties (состояние), Вам станет доступна группировка компонентов, чтобы использовать их вместе и присваивать им адреса запроса приложения на FlashCom Сервере, для подключения.

Для создания более разнообразных приложений, необходимо использовать ActionScript, чтобы создавать или настраивать компоненты и разрабатывать уникальные интерфейсы пользователя. Связанные между собой ActionScript классы делают использование компонентов и приложений более легким. На стороне клиента, они включают NetConnection, NetStream, Camera, SharedObject, и Microphone классы. ActionScript классы на стороне сервера включают Application, Client, Stream, and SharedObject классы.

Протокол обмена сообщениями в реальном времени

The Flash Player соединяется с FlashCom используя Протокол Передачи сообщений в реальном времени (RTMP). RTMP использует TCP (протокол управления передачей) для передачи пакетов между Flash Player и сервером. TCP - надежный протокол, который гарантирует доставку каждого пакета данных. RTMP может транспортировать аудио закодированное в MP3 и Nellymoser форматы, видео, закодированное в формате Flash Video (FLV), и данные закодированные в формат AMF. AMF обеспечивает эффективное преобразование в последовательную и параллельную формы данных так, чтобы и простые и сложные ActionScript данные могли быть переданы между клиентом и сервером без необходимости вручную кодировать или расшифровать их.

FlashCom Против Традиционных Media серверов.

Использование TCP как основного протокола для Flash связи упрощает управление транспортировкой аудио, видео, и ActionScript данных, передающихся между клиентом и сервером. Однако, TCP - протокол PPP (двухпунктовый), который подразумевает, что каждый клиент требует отдельного TCP клиента/сервера подключения; следовательно, он не может передавать пакеты от сервера ко многим клиентам на сетевом уровне. Если живой аудио поток был послан одновременно множеству клиентам, сервер должен послать дубликаты аудио данных каждому клиенту через дискретные подключения. Традиционные серверы средств информации, предназначенные прежде всего для того, чтобы распределить потоки средств информации от сервера к клиенту, обычно обеспечивают возможность послать поток, используя UDP (протокол пользовательских датаграмм) также как TCP. UDP может использоваться для многонаправленных или однонаправленных потоков, когда допускается, что единственная копия потока передана, и любой клиент, может получить ее. Мультивещание имеет два огромных преимущества: уменьшает загрузку на сервере и пропускную способность, требуемую, чтобы послать потоки большому числу клиентов.

К сожалению, огромное большинство системных служб Internet (ISPs) блокирует multicast в их сетях из-за беспокойства за безопасность. Следовательно, серверы средств информации обеспечивают аварийный переход к однонаправленной передачи данных по UDP. Однонаправленная передача означает, что сервер должен дублировать данные потока и посылать их в отдельном потоке каждому клиенту. Если по некоторым причинам клиент не принимает однонаправленный поток, UDP серверы средств информации естественно снижают скорость передачи дублированного потока данных каждому клиенту по TCP, почти такой же способ использует FlashCom .

Даже без широковещания, UDP имеет некоторые преимущества для традиционных серверов средств информации. UDP - не "надежный" протокол. Он не гарантирует, что каждый пакет, посланный от сервера до клиента дойдет. Для аудио и видео потоков, потеря некоторых пакетов не может оказать значимого влияния на качество воспроизведения. Чем более перегружена сеть, тем большее количество пакетов будет потеряно, так, что качество потока может ухудшиться без большого замедления доставки потока. Напротив, TCP регулирует сетевую перегрузку и пропускную способность, замедляя передачу данных и, снова посылая потерянные пакеты. Чтобы поддерживать передачу медиа данных по TCP, количество посылаемых данных должно быть динамически отрегулировано в ответ на сетевую пропускную способность и перегрузку. RTMP разработан, чтобы регулировать количество передаваемой видео и аудио информации, отбрасывая аудио сообщения и видео фреймы в ответ на недостаточную сетевую пропускную способность. На перезагруженных сетях это не столь же эффективно, как на сетях, основанных на UDP- протоколах для передачи медиа-потоков, но на современных сетях это соответствует очень хорошему выполнению.

RTMP поддерживает больше чем протоколы передачи медиа от традиционных серверов средств информации. Он также поддерживает динамическую передачу многочисленных потоков, которые могут содержать аудио, видео, и ActionScript данные и от сервера до клиента и от клиента на сервер. RTMP управляет передачей аудио, видео, и ActionScript данных отдельно. ActionScript данные никогда не будут потеряны, потому что любая потеря может иметь катастрофическое влияние на приложение. Аудио и видео данные буферизированы отдельно на сервере.

Если аудио данные в аудио буфере подходят к некоторому порогу, все данные в буфере сбрасываются, и вновь прибывшие данные могут начать накапливаться в буфере, которые нужно переслать каждому клиенту. Видео данные управляются подобным способом за исключением того, что данные в буфере сбрасываются, когда новый ключевой кадр прибывает. Сбрасывание видео данных, когда ключевой кадр прибывает, гарантирует, что клиент никогда не получит частичные модификации фрейма для

искаженного ключевого кадра. Иначе видео изображение могло бы состоять из мозаики блоков размерами 8*8пикселей от 2-х разных фреймов. RTMP также располагает приоритетами по данным. Аудио дается самый высокий приоритет, потому что это очень необходимо для общения в реальном масштабе времени. Видео дается самый низкий приоритет, и ActionScript дается промежуточный приоритет между аудио и видео.

Классы Соединения

В то время как RTMP удобно управляет передачей многочисленных потоков данных через сети, реальная сила Flash и FlashCom для разработчиков реализуется в классах высокого уровня, которые делают соединение с сервером, распределение информации, и передачу аудио и видео, простым и гибким.

Соединение с сервером.

NetConnection класс соединяет клиента с запросом приложения на сервере. В самом простом случае, Вы можете создавать подключение, вызывая функцию NetConnection.connect () с URI отдаленного приложения:

```
nc = new NetConnection();
nc.connect ("rtmp://echo.ryerson.ca/campusCameras/connector ");
```

Для того, чтобы определить, было ли успешно установлено подключение, нужно запросить onStatus() метод на объекте NetConnection перед вызовом функции connect():

```
nc = new NetConnection ();
nc.onStatus = function (info) {
 trace (" код подключения: " + info.code);
};
nc.connect (" rtmp://echo.ryerson.ca/campusCameras/connector ");
```

В этом примере, адрес RTMP включает хост (echo.ryerson.ca), имя приложения (campusCameras), и название запроса (connector).

Передача Аудио, Видео, и ActionScript данных

Как только Вы устанавливаете подключение, используя объект NetConnection в клиенте. Вы можете использовать его, чтобы посылать или получать поток, содержащий аудио и/или видео. Предположим, что Вы знаете, что зарегистрированный файл видео Flash по имени Ryerson_High_Speed.flv доступен в общем каталоге запросов. Вы можете прикрепить видео поток к видео объекту, здесь назван videoArea (видео область), используя Video.attachVideo() и проигрывать его, используя NetStream.play (). Объект NetStream, in_ns, создан, через объект NetConnection, nc, из предшествующего примера:

```
in_ns = new NetStream (nc);
videoArea.attachVideo (in_ns);
in_ns.play ("public/Ryerson_High_Speed");
```

В этом случае, видео появляется в видео объекте, который был помещен на сцене во время авторизации и которому было дано имя запроса videoArea; аудио в пределах видео потока прослушивается автоматически. Обратите внимание, что .flv расширение не включено в поток URI, проходящего в метод play ().

Помещение потока в приложении Flash Movie также просто. Здесь, вы создаете новый объект NetStream, out_ns, и прикрепляете аудио и видео потоки от пользовательского микрофона и камеры перед передачей потока, используя NetStream.publish ():

```
out_ns = new NetStream (nc);
out_ns.attachAudio (Microphone.get());
out_ns.attachVideo (Camera.get());
out_ns.publish (userName);
```

Класс Camera может получить видео поток от вэб-камеры или другого видео источника. Класс Microphone может получить аудио поток от микрофона или другого источника.

Вы можете создавать многочисленные потоки (NetStream объекты) для единственного NetConnection, как показано на рисунке 3. Каждый из потоков может создавать или запускать другие потоки к серверу или от него, но данные в пределах потока передаются только в одном направлении. Если каждый пользователь послал единственный поток, самое легкое назвать поток уникальным именем его пользователя или уникальным ID номером.

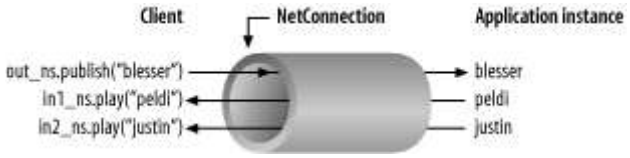

Рисунок 3. Использование потоков

На сервере, класс Stream может использоваться для создания плейлистов потоков и даже для проигрывания и публикации потоков через многочисленные серверы. Плейлист (список файлов для воспроизведения) определяет последовательность потоков, которые будут запущены один за другим. Сервер буферизует их, так что они проигрываются без задержки; когда один поток заканчивается, другой начинается.

Камера, Микрофон, и Видео

Классы Camera и Microphone, обеспечивают доступ к видео и аудио источникам (камеры и микрофоны) на системе клиента. Ко многим камерам и микрофонам можно получить доступ с помощью того же Flash Movie. Только один видео и аудио поток может быть запущен в пределах отдельного NetStream; однако, может быть много NetStream'ов в пределах одного NetConnection, как показано на рисунке 4.

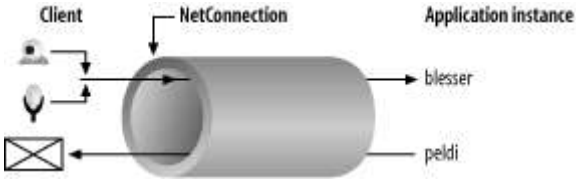

Рисунок 4. Множество потоков NetStream

Flash Player ответствен за кодирование всех аудио и видео данных. Видео кодирование от каждого видео источника в настоящее время ограничено кодированием скорости передачи данных в битах в единственном разрешении. Camera и Microphone классы могут использоваться, чтобы управлять количеством аудио и видео данных, посылаемых серверу, и могут быть динамически откорректированы, чтобы соответствовать ограничениям пропускной способности клиентов, связанных с приложением.

Класс Camera обеспечивает удобный способ изменять разрешающую способность, разряд фрейма, и качественные параметры настройки для каждого видео источника. Класс Microphone позволяет устанавливать частоту отсчетов, частоту амплитудно-импульсной модуляции, усиление, и уровень затухания, и др. Video объект используется, чтобы отобразить видео в пределах Flash movie и может быть динамически изменен и перемещен

Совместное использование Данных в Реальном времени

Приложения в реальном масштабе времени часто требуют, чтобы данные были разделены или переданы между множеством приложений (movie). SharedObject класс

знаком многим программистам Flash, как способ создать своего рода supercookie (в системах с удаленным доступом - пароль, порождаемый сервером при первом подключении и отсылаемый пользователю; при последующих подключениях пользователь должен предоставлять серверу этот пароль). Локальный общедоступный объект (LSO) может загружать ActionScript данные на клиенте между сеансами. Приложения Flash communication могут использовать удаленные общедоступные объекты (RSO), чтобы совместно использовать информацию в реальном времени между movie, выполняющимися на различных клиентах. Если movie изменяет свойство отдаленного общедоступного объекта, то же самое свойство изменяется в каждом другом movie, связанном с ним. Когда происходит общедоступные изменения объекта, каждый клиент уведомлен относительно изменений. Общедоступные объекты могут быть использованы для:

- Регистрации всех movie, связанных с приложением видео конференции по имени каждого доступного живого потока;
- Обновления позиции элементов в игре;
- Владения позицией, формой, и информацией о цвете каждого графического элемента в общедоступном whiteboard приложении;
- Владения данными для каждой формы элемента в общедоступной форме.

В объектно-ориентированном программировании (ООР), данные в пределах всех объектов в приложении определяют текущее состояние приложения. Общедоступные объекты обеспечивают удобный механизм, чтобы владеть состоянием приложения связи, распределенного через многочисленных клиентов и серверов

Удаленные общедоступные объекты могут быть временными или постоянными. Временные совместно используемые объекты, сохраняются только, в том случае если они используются.

Постоянные общедоступные объекты сохраняются на сервере, когда они не в использовании, позволяя клиентам возобновить работу с того места где они закончили, как только они повторно соединяются с сервером. Общедоступные объекты типа Proxied - механизм для создания объектов, доступных через запросы множества приложений. Один запрос приложения всегда имеет общедоступный объект, но другие могут создавать сетевое подключение к главному запросу и создавать proxies (по существу псевдоним) общедоступного объекта. Таким образом, клиенты соединенные с любым запросом, могут соединяться с тем же самым общедоступным объектом. Эта функция часто важна для создания приложений крупного масштаба.

Код для работы с общедоступными объектами на клиенте немного отличается от требуемого на сервере. Работа с общедоступными объектами также немного более сложна, чем работа с потоками или руководящими сетевыми подключениями. Вы должны сделать четыре основных действия когда работаете с общедоступными объектами:

- Вызовите отдаленный общедоступный объект, используя SharedObject.getRemote();
- Настройте общедоступный объект, так чтобы ваша программа могла ответить на изменения, примененные к общедоступному объекту каждым movie или сервером;
- Подключитесь к общедоступному объекту;
- После того, как подключитесь, обновите свойства общедоступного объекта по мере необходимости.

На клиенте, SharedObject.getRemote() метод возвращает удаленный общедоступный объект. Однако, общедоступный объект обычно должен быть установлен с помощью метода onSync() и затем подключен, используя объект NetConnection прежде, чем он может использоваться. Здесь мы получаем общедоступный объект названный пользователями, с которыми каждый movie соединится.

Затем, метод onSync() задается так, чтобы, в зависимости от цели демонстрации, отображать информацию об изменениях, примененных к общедоступному объекту, как только они происходят. Затем общедоступный объект связывается с сервером:

```
users_so = SharedObject.getRemote("users", nc.uri);
users_so.onSync = function (infoList) {
 for (var i in infoList) {
  var info = infoList[i];
  switch (info.code) {
   case "change":
    var id = info.name;
    trace("пользователь соединяется с id: " + id);
    trace("и именем: " + users_so.data[id]);
    break;
   case "delete":
    var id = info.name;
    trace("пользователь разъединяется с id: " + id);
    break;
  }
 }
};
users_so.connect(nc);
```

Метод onSync()- часто наиболее интересная часть общедоступного объекта. Когда локальная копия (копия файла, хранимая на локальном сетевом компьютере) общедоступного объекта синхронизирована с сервером и всякий раз, когда любые свойства в общедоступном объекта изменяются, метод onSync() вызывается автоматически. Метод onSync() «пропускает» через себя массив информационных объектов. Каждый объект содержит информацию об общедоступном объекте или о том, что случилось со слотом (свойство) в общедоступном объекте. Каждый информационный объект имеет свойство кода, которое описывает происходящее "clear", если все свойства удалены, "change", если свойство добавлено или модифицировано удаленно, или "delete", если свойство удалено. Имя слота (свойства), который был модифицирован или удален, всегда обнаруживается в названии свойства информационного объекта. Чтобы добавлять или обновлять (модифицировать) данные в общедоступном слоте объекта в пределах Flash, используйте свойство данных общедоступного объекта:

```
users_so.data ["guest_4"] = "Брайен";
```

Аналогично, Вы можете извлекать значение в слоте, используя свойство данных:

```
trace (" пользовательское название(имя): " + users_so.data ["guest_4"]);
```

Клиент и Прикладные Объекты

Всякий раз, когда клиент соединяется с прикладным запросом на сервере, объект Client создается на стороне сервера, чтобы представить удаленного клиента. Объект Client может использоваться серверными скриптами, чтобы посылать и получить сообщения, посылаемые к и от каждого индивидуального отдаленного клиента Flash.

Каждый запрос приложения также имеет единственный объект приложения, который обеспечивает удобный способ управления циклом жизни запроса. Приложение - одноэлементный запрос Application class на стороне сервера.

Client и application объекты, одновременно со способностью сделать запрос к серверу приложения сети, предусматривают основные возможности необходимые, для подтверждения подлинности клиента, как только они подключаются.

Удаленные Методы

В типичном приложении, индивидуальный клиент должен сделать запрос, чтобы сервер выполнил действие от своего лица. Например, необходимо запросить у сервера длину зарегистрированного потока перед его проигрыванием. Сервер также должен послать запрос клиенту для того, чтобы предпринять что-либо. Например, серверу нужно, чтобы клиент использовал уникальную строку, предоставленную сервером, чтобы дать

имя потокам, которые клиент создает. Запросы удаленного метода - путь, по которому клиент и сервер могут вызывать методы друг у друга.

Пример приложения может вызывать метод клиента, использующего Client.call() метод. Клиент может вызывать метод на пример приложения, используя NetConnection.call() метод. Например, скрипт сервера может вызывать метод индивидуального клиента, чтобы позволять клиенту узнать его уникальный ID номер:

```
client.call ("setID", null, id);
```

Если метод клиента, названный setID() вызван, используя Client.call() от сервера, метод должен быть, определен на объекте клиента NetConnection, в противном случае ничего не произойдет. Например, скрипт клиента мог бы выглядеть следующим образом:

```
nc = новый NetConnection ();
nc.setID = function (id) {
 myID = id;
};
```

Наоборот, клиент может вызывать метод сервера, используя объект NetConnection:

```
nc.call ("getStreamLength", streamInfoResponder, streamName);
```

Для результата, который будет возвращен отдаленным методом, второй параметр в функции NetConnection.call() должен быть объектом с onResult() методом.

FlashCom также поддерживает механизмы, которые посылают отдаленный запрос метода многочисленным клиентам в одно и тоже время. Клиенты соединенные с одним и тем же общедоступным объектом, или пропускающие один и тот же поток, могут получать отдаленный запрос метода одновременно. Например, популярный способ обновлять текстовую область чата состоит в том, чтобы использовать send() метод общедоступного объекта для отправки текстового сообщения каждому клиенту:

```
chat_so.send ("showMessage", " Добро пожаловать в чат. ");
```

В этом случае, showMessage() метод должен быть определен на общедоступном объекте, или ничего не произойдет:

```
chat_so.showMessage = функциональный (сообщение) {
 chatTextArea.text + = сообщение + '\n ';
 chatTextArea.vPosition = chatTextArea.maxVPosition;
};
```

Соединение с Серверами приложений, Базами данных, и серверными Каталогами

Flash и Flash Communication Server часто должны работать с другими существующими приложениями и ресурсами. Например, пользователям, необходимо, зарегистрироваться непосредственно на существующем каталоге обслуживания или в базе данных прежде, чем войти в чат или просмотреть видео потоки. Системы Базы данных могут использоваться, чтобы хранить большое количество информации, с которой FlashCom не может легко справляться, типа миллионов записей, которые представляют местоположение видео сообщений в видео почтовой системе. Каждая запись может содержать почтовый текст и местоположение зарегистрированного видео сообщения в пределах приложения FlashCom.

Flash Player и FlashCom могут взаимодействовать с серверами вэб-приложений и, через них, с базами данных и серверными каталогами. Клиент Flash может вызвать любой серверный скрипт доступный на сервере сети, посылать и получать XML данные, управлять доступом сервера сети, и использовать шлюз Flash Remoting, чтобы более быстро обратиться к прикладным серверам.

Flash Remoting - технология запроса/ответа, которая позволяет скриптам в клиенте Flash или FlashCom вызывать отдаленные методы на прикладном сервере. Он использует HTTP, чтобы посылать и получать данные в AMF. Удаленные методы могут извлекать информацию из базы данных, каталога сервера или сервера сети и возвращать

информацию к FlashCom. С точки зрения разработчика, легко работать с Flash Remoting потому что комплексные ActionScript данные, преобразовываются в последовательную форму и параллельную автоматически. Flash Remoting может использоваться с FlashCom или без него. Дело в том, что Flash Remoting достаточно гибка, это позволяет FlashCom связываться эффективно, несмотря на недостаток FlashCom в отсутствии прямой поддержки для XML или серверном доступе.

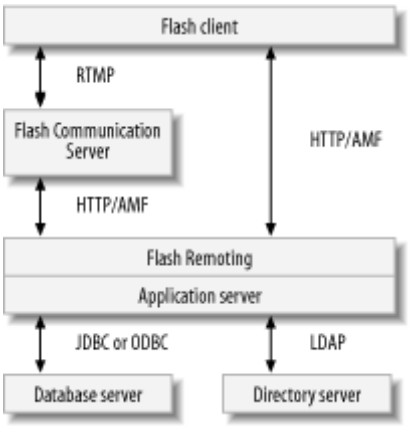

Рисунок 5. Опции связи для Flash и FlashCom

Рисунок 5 иллюстрирует некоторые опции связи для Flash и FlashCom. Flash Клиент может обращаться к приложению сети непосредственно, как может и FlashCom.

В некоторых случаях, клиенты могут соединяться и с прикладным сервером и FlashCom. В других случаях, FlashCom может соединяться с прикладным сервером и предоставлять информацию от или до ее клиентов. Когда много клиентов нуждаются в доступе к одним и тем же данным, FlashCom должен использоваться как посредник между прикладными серверами. Прикладное выполнение будет улучшено, если сократить число запросов от каждого приложения Flash. Когда каждый Movie должен искать информацию, уникальную для него, прямое подключение к прикладному серверу - это обычно лучший подход.

Аппаратно-программные средства межсетевой защиты и Безопасность

Некоторые общие средства межсетевой защиты и промежуточные сервера с ограниченными правами доступа делают невозможным установку постоянного TCP/RTMP подключения к FlashCom серверу. Некоторые общие FlashCom сервера могут открыть доступ к удаленным FlashCom серверам, но к другим не могут. Когда общий доступ к сети разрешается, но TCP/RTMP подключения не позволяются, FlashCom предусматривает особенность туннелирования, которая позволяет Flash и FlashCom посылать и получать RTMP по HTTP. Когда туннелирование используется, Flash Player работает с пользовательским браузером, чтобы выбирать сервер связи вместо установления прямого серверного подключения TCP. Когда RTMP - туннелируется, это известно как RTMPT. Из этого следует, что ни Flash Player ни FlashCom непосредственно не поддерживают SSL. Однако, шифрование - опция, которая используется при туннелировании, потому что браузеры поддерживают SSL, и полномочие SSL может использоваться сервером. Для получения дополнительной информации, см. статью в:
`http://www.macromedia.com/devnet/mx/flashcom/articles/firewalls_proxy.html`

Подготовка к работе

Вы можете загрузить Flash Communication Server бесплатно с сайта Макромедиа. более подробную информацию о загрузке и выборе линцензирования смотри на сайте:
http://www.macromedia.com/software/flashcom

Установка FlashCom:

Следуйте за ссылкой от предшествующего URL, чтобы загрузить бесплатную версию программы разработчика, или идите непосредственно на:
http: // www.macromedia.com/cfusion/tdrc/index.chm?product=flashcom

Как только Вы устанавливаете FlashCom, Вы должны загрузить самый последний модуль обновления с:
http://www.macromedia.com/support/flashcom/downloads_updaters.html

и установить его. Модули обновления обычно не добавляются к инсталляционным файлам самого последнего выпуска, так если модуль обновления доступен для самого современного выпуска, Вы должны загрузить и установить его.

Наконец, загрузите самые современные communication components Flash, и выполните инсталляцию, чтобы установить их в среду создания Flash. Для Windows или Macintosh обновления присутсмтвуют на сайте:
http://www.macromedia.com/support/flashcom/downloads_updaters.html

FlashCom выполняется на Windows 2003 Сервере, Windows 2000 Сервере, Сервере Windows NT SP6 или выше, и RedHat Linux. Полные системные требования и серверы поддержки перечислены в:
http://www.macromedia.com/software/flashcom/productinfo/systemreqs

Подробная документация о сервере, включая инсталляционные команды доступна от Macromedia в:
http://www.macromedia.com/support/flashcom/documentation.html

Для простого тестирования сервера, Вы можете просто выполнить инсталлятор, указав начальное имя администратора и пароль, а затем сделать прогон примеров, чтобы удостовериться, что сервер работает. Вы можете не вводить серийный номер. На Windows, инсталлятор назван FlashComInstaller.exe, и Вы можете просто запустить его. Под Linux, Вы должны разархивировать и затем инсталляцировать файл, cd, к installation_directory и напечатать:
./installFCS

По умолчанию, FlashCom Сервер воспринимает для подключений TCP порт 1935. В идеале, Вы должны позволить доступ к этому порту от других машин только под вашим управлением. Если машина не под действием межсетевой защиты, Вы должны по крайней мере удостовериться, что сервер принимает подключение, запрашивая только файлы с расширением .swf, инициирует передачу данных расположенных в вашем собственном домене. См.:
http://www.macromedia.com/devnet/mx/flashcom/articles/firewalls_proxy06.html

Вы можете проверить, как, сервер работает, войдя в подкаталог tutorial_sharedball и запуская две копии файла tutorial_sharedball.swf. Вы должны свободно перетаскивать шар всюду в одном окне и наблюдать, его движение в другом movie. Если это не работает, Вам, вероятно, придется вручную запускать сервер. В окне Windows, выберите Start Programs Macromedia Flash Communication Server Start Server.

Macromedia обеспечивают различные издания и лицензирующие схемы для FlashCom Сервера. Издание разработчика не лицензировано для дальнейшего распространения. FlashCom Сервер включает менеджера лицензии, который контролирует число одновременных клиентов, которым позволяют соединиться в одно время и разрешают потреблять серверу полную пропускную способность. Есть также ограничения на издания сервера чем, можно пользоваться, чтобы создавать виртуальные хосты. Полное описание продукта и лицензирования может быть найдено здесь:

http://www.macromedia.com/software/flashcom/productinfo/editions

Крис Хок написал полезную статью о, Вычисление вашей Пропускной способности и Программных Потребностей Лицензии для Macromedia Flash Communication Server MX. Вы можете найти это здесь:
http://www.macromedia.com/software/flashcom/productinfo/editions/fcs_whitepaper_bandwidth.pdf

Admin Service, Administration Console, и App Inspector

FlashCom инсталлятор также включает вторичное приложение, известное как Admin Service или Administration Controller. На Windows, Admin Service установлено FlashComAdmin.exe, и на Linux fcadmin файл. По умолчанию, Admin Service запускается всякий раз, когда FlashCom запущен и обеспечивает административные услуги и контроль приложения, управление, и услуги отладки. Вы можете соединяться непосредственно с Admin Service использованием одного из двух Flash movies, обеспеченных FlashCom. В FlashCom 1.5.2 версии, Вы найдете их в подкаталоге flashcom_help\html\admin инсталляционного каталога. Administration Console (admin.swf) movie может использоваться, для обновления информации о лицензии, начинать и останавливать прикладные запросы, и просматривать на сервере диагностическую информацию. App Inspector (app_inspector.swf) может использоваться, чтобы запускать и остановливать запросы приложений, контролировать прикладные ресурсы запроса, и выводить на дисплей сообщения trace ().

Чтобы использовать любой movie, запустите movie и войдите в Admin Service, используя пользовательское имя администратора и пароля, который Вы определили, когда установилвали сервер. Macromedia предоставляет информацию относительно использования Administration Console и Communication App Inspector в Managing Flash Communication Server и Developing Communication Applications, оба доступны из:
http://www.macromedia.com/support/flashcom/documentation.html

Communication Application Inspector особенно полезен при отладке приложений. Вы можете также создавать ваше собственное Flash Movie и приложения связи, которые соединяются с Admin Service.

Привет Видео (helloVideo)

Приложение предназначено, для демонстрации многих вещей, описанных в этой части, как-то: публикации и проигрывания потоков и обновления и реагирования на изменения в общедоступном объекте. Идея состоит в том, чтобы обеспечить быстрый просмотр формирования приложения связи. Хотя действующее видео приложение чата уже существует, это - хорошая демонстрация понятий и действий, в которых Вы будете нуждаться, при создании ваших собственных приложений.

Установка helloVideo на Сервере

Создание helloVideo приложения на сервере требует, чтобы Вы нашли каталог приложений и добавили подкаталог, названный helloVideo. После заданной по умолчанию инсталляции на моей системе, каталог приложений расположен в:
C:\Program Files\Macromedia\Flash Сервер Связи MX\applications

Как только Вы создаете helloVideo подкаталог, Вы создали приложение FlashCom. Теперь Вы должны обеспечить приложение его уникальным серверным режимом. Создайте пустой текстовый файл, названный main.asc, и сохраните его в каталог helloVideo. Вы можете использовать любой текстовый редактор открытого текста типа

того, который включен в Flash MX Professional 2004 или Dreamweaver MX 2004 Пример 1 показывает исходный текст, который Вы должны добавить к main.asc файлу.

Пример 1

```
idPool = ["guest_1", "guest_2", "guest_3", "guest_4"];
application.onAppStart = function () {
  users_so = SharedObject.get("users");
};
application.onConnect = function (client, name) {
  if (idPool.length <= 0) {
    application.rejectConnection(client, {msg:"Too many users."});
  }
  client.id = idPool.pop();
  application.acceptConnection(client);
  client.call("setID", null, client.id);
  users_so.setProperty(client.id, name);
};
application.onDisconnect = function (client) {
  idPool.push(client.id);
  users_so.setProperty(client.id, null);
};
```

Файл будет загружен, компилироваться, и выполняться сервером, когда первый клиент попытается соединяться с примером helloVideo. Затем будет вызван метод application.onAppStart(), после того, как файл будет запущен. Затем, когда movie попробует соединиться, будет вызван метод application.onConnect(), и когда movie разъединится, application.onDisconnect() будет вызван.

main.asc файл, представленный в Примере 1 предназначен, для трех вещей:

- Только четырем клиентам позволяется соединиться в любое время. Так, как приложение создает четыре уникальных пользовательских значения ID, передает по одному на каждого клиента, когда подключает клиентов, и востребует обратно ID, когда клиент отключается.
- Уведомляет каждого клиента его ID, вызывая отдаленный метод клиента после того, как подключение осуществлено.
- Модифицирует общедоступный объект, когда клиент прибывает или уходит так, чтобы все другие клиенты знали, кто находится на связи и имя каждого клиентского потока, который проигрывается.

Приложение не использует Flash Remoting, для подключения к опознавательной базе данных или каталогу сервера. Это - простая демонстрационная программа и не предназначена для защиты. В helloVideo приложении, любым четырем пользователям позволяют соединиться, и каждому дают уникальную пользовательскую строку ID. Глобальная переменная idPool -это массив, содержащий четыре доступных строки ID. Он создается, как только main.asc файл загружен сервером обычно, когда первый клиент пытается соединяться. application.onAppStart() метод вызывается немедленно после того, как main.asc файл загружен и выполнен. onAppStart() метод использует SharedObject.get() для создания временного общедоступного объекта, который содержит дополнительное имя, предусмотренное для каждого пользователя. Например, если бы четыре пользователя с именами "justin", "peldi", "robert", и "brian" были связаны, общедоступный объект имел бы имена слота и значения, иллюстрированные в Таблице 1.

Таблица 1.Слот имен и значений для users_so общедоступного объекта

Имя Слота	Значение Слота
"guest_1"	"justin"
"guest_2"	"peldi"
"guest_3"	"robert"
"guest_4"	"brian"

Следующий оператор получает общедоступный объект названный пользователями и определяет его в переменную users_so:

```
users_so = SharedObject.get ("users");
```

Согласно этому, мы можем использовать users_so, чтобы обратиться к методам и свойствам пользователей общедоступного объекта.

Всякий раз, когда клиент пытается соединяться, метод application.onConnect() вызывается через объект клиента, переданного во FlashCom, который представляет клиента, пробующего соединиться. Любую другую информацию о клиенте также прописывают в onConnect() как дополнительные параметры. В Примере 1, имя, которое пользователь вводит - второй параметр, name.

Когда вызывается onConnect(), мы имеем опцию отклонения или принятия подключения или оставления его (на ожидание). В этом примере, если нет, пользовательских строк ID, помещенное в idPool, приложение отклоняет подключение и отправляет сообщение назад клиенту, чтобы сообщить причину отказа в соединении:

```
  if (idPool.length <= 0) {
    application.rejectConnection(client, {msg:"Too many users."});
```

Если есть доступная строка ID, это удалено с конца массива idPool и назначено к id свойству объекта клиента (id - не встроенное свойство объекта Client; мы создали его, чтобы удовлетворить наши потребности):

```
client.id = idPool.pop ();
```

Если ID доступен, приложение примет запрос клиента, чтобы подключить и послать клиенту его пользовательский ID, вызывая setID(), который был представлен ранее под " Отдаленные Методы, " на клиенте:

```
  application.acceptConnection(client);
  client.call("setID", null, client.id);
```

Наконец, приложение позволяет всем другим клиентам узнать, что новый клиент соединился, так что они могут подписаться на видео и аудио поток:

```
  users_so.setProperty(client.id, name);
```

setProperty() метод сохраняет имя параметра в слоте названном по имени строки ID клиента.

Позже, когда клиент отключается, щелкая кнопкой Disconnect (Разъединить) или, переходит а браузере на другую страницу, метод application.onDisconnect() будет, обращаться к серверу и передавать объект клиента, представляющего клиента, который отключился. Когда клиент разъединяется, мы должны востребовать его строку ID для использования с другими клиентами, и мы должны удалить ее слот в пользователях общедоступный объект указав, что она не связана никакой длинной:

```
application.onDisconnect = function (client) {
  idPool.push(client.id);
  users_so.setProperty(client.id, null);
};
```

Приложение помещает ID назад в массив idPool и устанавливает его слот в общедоступном объекте к пустому указателю.

Создание helloVideo Клиента во Flash

Когда Flash movie соединяется с сервером, он получает собственную уникальную пользовательскую строку ID в ответ. Он издаст поток, названный по имени его пользовательского ID для контроля изменений в users общедоступном объекте обнаруживая уникальный ID каждого пользователя, кто соединяется. Он использует пользовательский IDs в users общедоступном объекте при запуске каждого отдаленного пользовательского потока.

Создание интерфейса пользователя

На рисунке 6 изображен интерфейс пользователя для клиента helloVideo.

Рисунок 6. Интерфейс HelloVideo

Интерфейс сделан, используя четыре movie clip'а и несколько компонентов. В приложении присутствует один movie clip (MC) для каждого пользователя. MC содержит вложенный Видео объект, Метку, текстовое поле компонент TextInput. TextInput компонент в каждом MC отображает имя каждого пользователя введенного в поле My Name. Обратите внимание на TextInput компонент внизу экрана, содержащего текст "Robert". Компонент TextInput используется, чтобы отобразить текущее состояние приложения подключения. Кнопка также указывает, что пользователь подключен, показывая метку Disconnect (Разъединить). Если пользователь не подключен, она переключается в Connect (Соединить).

Создание интерфейса:

Начините с пустого приложения Flash и установите его размеры 320 x 480, используя панель Properties (Свойства).

Создайте новый символ библиотеки, используя команду Insert New Symbol (Ctrl-F8).

В диалоговом окне Create New Symbol, введите символ, названный GuestVideo и установите тип Behavior в MovieClip. Укажем Export для опции ActionScript, (щелкните кнопкой Advanced, чтобы отобразить эту опцию, если ее Не видно). Настройте также поле Identifier в GuestVideo.

Когда символ создан в Библиотеке, Сцена отображает пустой символ и его точку регистрации. Мы хотим разместить вложенный Видео объект в пределах символа так, чтобы левый верхний угол видео был в точке регистрации символа.

Для того чтобы добавить видео в библиотеку, откройте Library panel и выберете new Video из панели настроек как показано на рисунке 7.

Перетащите Видео объект из Библиотеки на Сцену, и установите его в точке регистрации символа GuestVideo. Используйте поля X и Y в панели Properties как иллюстрировано на рисунке 8, чтобы установить точно в начало координат(0, 0), и дайте этому примеру имя video.

Перетащите одну Метку и один TextInput компонент из панели Components на Сцену. Разместите их как показано на рисунке 8, и дайте TextInput компоненту примера имя nameInput.

Установите параметр Text из Метки к тексту Name, используя панель Properties.

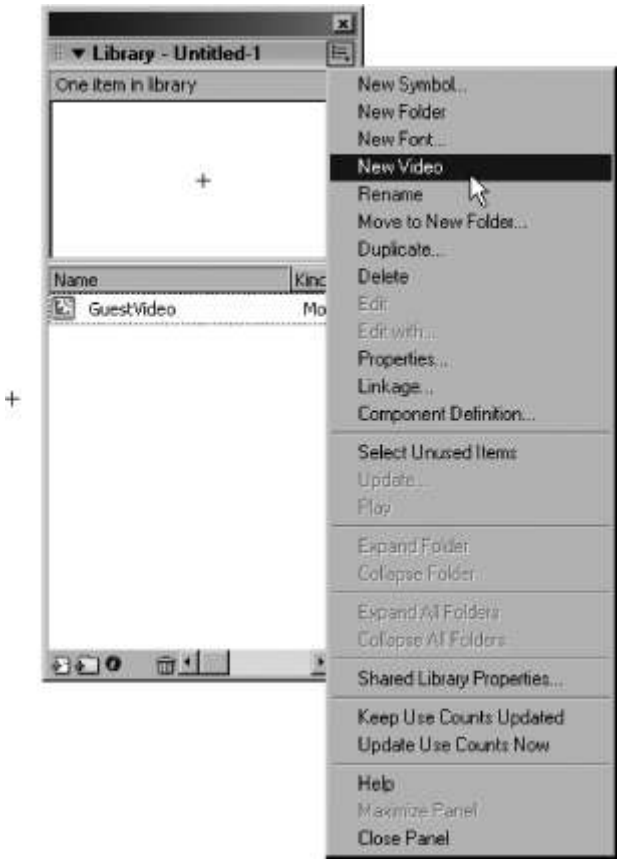

Рисунок 7. Вставка нового видео

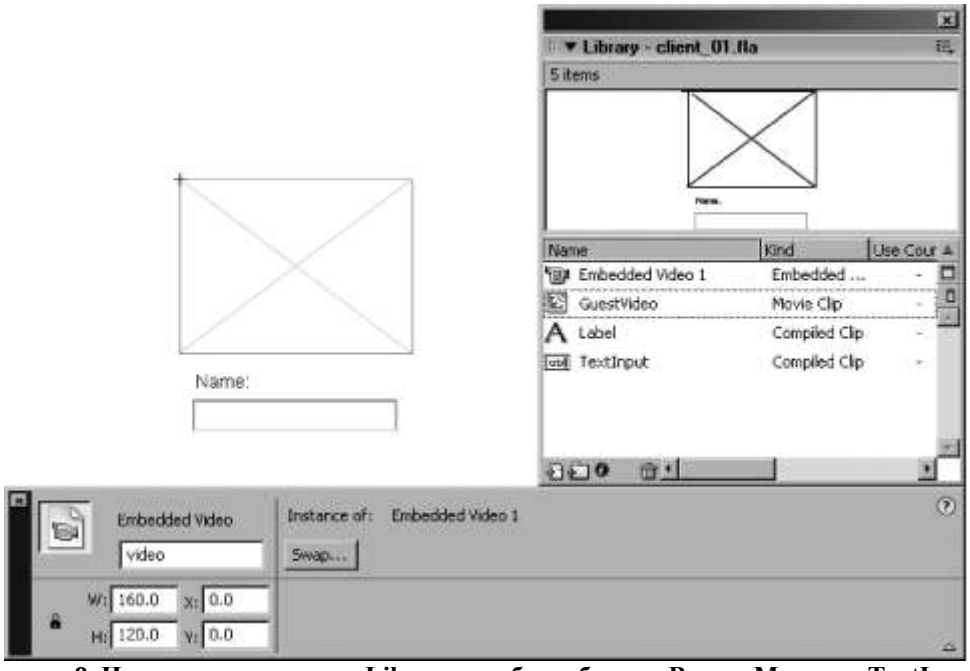

Рисунок 8. Использование меню Library, чтобы добавить Видео, Метку, и TextInput к GuestVideo символу

Теперь, когда Вы создали GuestVideo символ с вложенным Видео объектом, необходимо обеспечить ее четырем компонентам на Сцене (чтобы отобразить четырех возможных одновременных клиентов):

Чтобы вернуться к Сцене главного movie, щелкните на Сцене ссылкой 1 в полосе Редактирования панели Timeline.

Перетащите GuestVideo символ из Библиотеки на Сцену четыре раза, и упорядочьте эти четыре образца как показано на рисунке 6. Выберите каждый в свою очередь, и установите ему имя (instance name) guest_1, guest_2, guest_3, или guest_4, используя панель Flash's Properties.

Наконец, внизу, добавьте два компонента Метки, два компонента TextInput, и один компонент Кнопки. Их Позиции и размеры проиллюстрированы на рисунке 6. Назовите большой TextInput statusInput, TextField userNameInput, и Кнопке connectButton.

Теперь основы созданы, можно программировать.

Установки NetConnection и просмотр его состояния

Пример 2 показывает, как создается объект NetConnection ,и к нему динамически добавляются методы. Этот клиентский скрипт, подобно коду в следующих примерах, должен быть помещен в первый фрейм на отдельном слое скриптов во временной шкале movie.

После того, как nc переменная назначена, новый NetConnection, код добавляет два метода. setID() метод вызывается сервером, чтобы уведомить клиента о его уникальном пользовательском ID. Метод onStatus() вызывается всякий раз, когда происходят изменения в состоянии подключения. Взгляните на Пример 2, и попробуйте разобрать то, что эти два метода делают.

Пример 2. Установка NetConnection

```
myID = "";
nc = new NetConnection();
/* setID() is a remote method that will be called by the server
 * after the client connects. Once we get our unique ID from the
 * server we can use it to publish our stream.
 */
nc.setID = function (id) {
  myID = id;
  statusInput.text = "Online. Your client's ID is: " + myID;
  ns = new NetStream(nc);
  ns.attachAudio(Microphone.get());
  var cam = Camera.get();
  ns.attachVideo(cam);
  ns.publish(id);
  _root[id].video.attachVideo(cam);
  initUsers();
};

// onStatus() is called whenever the status of the network
// connection changes. It is how we know whether we are connected.
nc.onStatus = function (info) {
  connectButton.label = "Connect";
  connectButton.enabled = true;
  switch (info.code) {
    case "NetConnection.Connect.Success":
      connectButton.label = "Disconnect";
      statusInput.text = "Online";
      break;
    case "NetConnection.Connect.Failed":
      statusInput.text = "Cannot reach server. Possible network error.";
      break;
    case "NetConnection.Connect.Rejected":
      statusInput.text = info.application.msg;
```

```
      break;
    case "NetConnection.Connect.Closed":
      statusInput.text += " Connection closed.";
      break;
  }
};
```

Некоторое время после подключения к серверу прошло, setID() метод вызывается сервером, и его id параметр передается как одна из четырех строк: " guest_1 "," guest_2 "," guest_3 ", или " guest_4 ". setID() метод сохраняет имя в переменную myID и отображает его для пользователя в текстовом поле statusInput. Затем он использует его строку ID, чтобы издать поток, содержащий аудио и видео, создавая объектNetStream на nc сетевом подключении. Это подключает микрофон и камеру к потоку и издает его, используя id как имя потока. Наконец, это отображает поток, который его посылает в одном из GuestVideo movie clips на Сцене.

Так как каждый клип назван по имени пользовательского ID, сервер обеспечивает будет один клип на Сцене с тем же самым (именем) как пользователь, который только что был назначен. _root свойство, которое проводит ссылку к главному movie's временной шкалы, используется, чтобы получить ссылку к movie clip. _root [id] возвращает ссылку к одному из clip GuestVideo. Свойство _root [id] .video возвращает ссылку к вложенному Видео объекту в clip, и _root [id] .video.attachVideo(cam) отображает то, что камера видит в Видео объекте. setID() метод также вызывает initUsers() метод для установления общедоступного объекта пользователей.

onStatus() метод получает информационный объект, который содержит информацию о самом последнем связанном с подключением. Свойство code объекта-строка в разграниченном точкой формате, типа " NetConnection. Connect.Success ". В зависимости от строки в свойстве code, сообщение, указывающее сетевое состояние подключения отображено в statusInput текстовой области). Для соединения с сервером, пользователь должен щелкнуть кнопкой Connect (connectButton). В то время, кнопка заблокирована, таким образом пользователь не может сделать попытку второго подключения не дождавшись результата первой попытки. onStatus() метод поэтому повторно запускает кнопку и ставит ее метку в Connect (Соединенить), пока подключение не будет установлено.

Создание подключения

В Примере 3, connectButton установлен, чтобы передать события щелчка click() функции всякий раз, когда кнопка нажата. Кнопка отображает одно из трех меток Connect (Соединенить), Wait (Ожидание), или Disconnect (Разъединить) в зависимости от состояния соединения. Если кнопка отображает метку Connect, функция click() попытается сделать подключение, используя nc.connect(). Если стоит метка на кнопке - Disconnect, щелчек кнопкой закрывает текущее подключение, вызывая nc.close().

Пример 3. Создание подключения.

```
connectButton.addEventListener("click", this);
function click (ev) {
  var button = ev.target;
  var command = button.label;
  switch (command) {
    case "Connect":
      nc.connect("rtmp:/helloVideo", userNameInput.text);
      button.label = "Wait...";
      button.enabled = false;
      break;
    case "Disconnect":
      nc.close();
      button.label = "Connect";
      break;
```

```
  }
}
```

Показ удаленных пользователей

Давайте взглянем, как клиент узнает, какие потоки присоединены и как отобразить их видео вместе с именем каждого пользователя.

Когда сервер вызывает setID клиента метод, setID() вызывается методом initUsers(), показанным в Примере 4, чтобы установить общедоступный объект и подключить пользователей к этому объекту. Пример 4 демонстрирует три вещи: это получение общедоступного объекта, динамическое определение метода onSync() для него, и затем подключает его, используя nc сетевое подключение. Когда общедоступный объект сначала синхронизирован с сервером, любые данные в версии сервера общедоступного объекта скопированы в версии клиента общедоступного объекта, и затем onSync() будет вызываться, чтобы уведомить клиента об изменениях.

После этого, любые изменения, сделанные в версии сервера отражаются в модифицируемой версией клиента и onSync() будет вызвана снова. onSync() метод в Примере 4 делает всю работу по реагированию на изменения в общедоступном объекте. Когда метод вызван, он получает массив объектов. Каждый объект имеет свойство, которое указывает то, какому типу изменения подвергся объект. Есть многочисленные возможные свойства, но мы заинтересованы только двумя из них: "change" и "delete". Мы можем игнорировать остальное, потому что все модификации и стирание общедоступного объекта сделаны серверным кодом в main.asc. Код "change" указывает, что сервер добавил или обновил слот в общедоступном объекте.

Помимо встроенного свойства, содержимое в информационном объекте зависит от типа изменения, описанного общедоступным объектом. В этом примере, свойство названия информационного объекта содержит название (имя) слота, который изменился. Например, если пользователь вошел и ему был дан ID "guest_2", то информационный объект со значением кода "change" будет также иметь свойство имени "guest_2". Если пользователь разъединяется, сервер удалит слот общедоступного объекта, так что значение кода будет "delete", и свойство названия идентифицирует удаленный слот. onSync() код в Примере 4 происходит проход по элементам массива начала и конца массива информационных объектов, проходящих в onSync() и проверяет свойство кода каждого объекта. Если свойство кода - "change", метод initUsers() запускает пользовательский поток и показывает его имя в одном из GuestVideo movie clips. Если свойство кода - "delete", пример прекращает запускать поток, связанный с пользователем, который разъединился.

Пример 4. Установка пользователей Общедоступного Объекта

```
function initUsers (  ) {
  users_so = SharedObject.getRemote("users", nc.uri);
  users_so.onSync = function (infoList) {
    for (var i in infoList) {
      var info = infoList[i];
      switch (info.code) {
        case "change":
          var id = info.name;
          var mc = _root[id];
          mc.nameInput.text = users_so.data[id];
          if (myID != id) {
            var ns = new NetStream(nc);
            mc.video.attachVideo(ns);
            ns.play(id);
            mc.ns = ns;
          }
          break;
        case "delete":
          var id = info.name;
```

```
            var mc = _root[id];
            mc.ns.close();
            mc.nameInput.text = "";
            mc.video.clear();
            break;
        }
      }
  };
  users_so.connect(nc);
}
```

Желательно, более тщательно рассмотреть код, который начинает запускать отдаленный пользовательский поток и показывает его имя

```
case "change":
  var id = info.name;
  var mc = _root[id];
  mc.nameInput.text = users_so.data[id];
  if (myID != id) {
    var ns = new NetStream(nc);
    mc.video.attachVideo(ns);
    ns.play(id);
    mc.ns = ns;
  }
  break;
```

Свойство name - удаленный пользовательский ID. Например, если это - "guest_2", _root [id], приравнивает к GuestVideo movie clip то же самое имя. Как только узнается то, что clip представляет пользователя, мы можем показывать имя выбранного пользователя когда он вошел, устанавливая текст в поле nameInput TextInput компонента внутри movie clip:

```
mc.nameInput.text = users_so.data [id];
```

В отличие от кода на стороне сервера, который использовал users_so.setProperty() и users_so.getProperty() чтобы устанавливать и получать значения в общедоступном слоте объекта, в клиенте специальный объект данных доступен, чтобы читать и записывать значения слота. Например, чтобы получить или установить guest_2 слот общедоступного объекта, используйте выражение: users_so.data ["guest_2"].

Поскольку серверный код изменяет слот, каждый раз как пользователь соединяется, клиент получит уведомление, что слот был изменен, когда другие удаленные пользователи соединяются, а также когда он сам соединяется. Однако, необходимо запустить только потоки удаленных пользователей, потому что нет никакого смысла в растрате пропускной способности, чтобы запустить местный пользовательский собственный поток. К счастью, setID() метод вызывается прежде, чем общедоступный объект будет установлен и связан, так что мы уже знаем местный пользовательский ID. Если id отличается от значения в myID переменной, id представляет удаленного пользователя, и можно безопасно запустить его. Чтобы проиграть поток данных код создает новый объект NetStream и прилагает его как динамическое свойство movie clip:

```
var ns = new NetStream(nc);
mc.video.attachVideo(ns);
ns.play(id);
mc.ns = ns;
```

Если удаленный пользователь разъединяется, объект NetStream запускающий его поток может быть безопасно закрыт, а его видео очищено, и nameInput поле, принимается за пустую строку:

```
case "delete":
  var id = info.name;
  var mc = _root[id];
  mc.ns.close();
  mc.nameInput.text = "";
  mc.video.clear();
  break;
```

Заключение

Если Вы пролистали код и комментарий относительно приложения helloVideo, Вы увидели большинство классов, работающих вместе, чтобы создать очень простое приложение видео конференции. Вы уже видели многие из основных методов чтобы строить приложение связи.

Лабораторная работа №2: Компоненты соединения (Communication components)

Macromedia добавила возможность работать с компонентами в вышедший Flash MX и выше. Эти компоненты позволяют Flash-дизайнеру или разработчику быстро добавить элементы интерфейса или интерактивные объекты в Flash move. Стандартная установка Flash MX 2004 включает UI компоненты, которые являются основными расширениями графического пользовательского интерфейса, такие как полосы прокрутки, списки и командные кнопки. Macromedia продолжила выпуск дополнительных компонентов для Flash MX уделив особое внимание для таких серверных продуктов Macromedia MX, как Flash Remoting MX и Flash Communication Server MX. В этой части, Вы узнаете о компонентах соединения и о том, как эти основные компоненты могут использоваться, для создания базовых FlashCom приложений.

Прежде чем Вы приступите к работе пео этой части, удостоверьтесь, что Вы загрузили и установили последние компоненты соединения.

Если Вы используете Flash MX, получите компоненты соединения здесь:
`http://www.macromedia.com/software/flashcom/download/components/`

Если Вы используете Flash MX 2004 или Flash MX Pro 2004 и выше, загрузите обновленные компоненты здесь:
`http://www.macromedia.com/support/flashcom/downloads_updaters.html`

Вы можете полагаться не только на компоненты Macromedia. Вы можете загрузить сторонние компоненты Flash Communication Server (http://www.macromedia.com/cfusion/exchange/) и установить их с помощью Macromedia Extension Manager. Вы также можете также создать Ваши собственные компоненты.

Оригинальные компоненты соединения, выпущенные для Flash MX, отличаются от компонентов Flash MX UI новым внешним видом, улучшенными командными кнопками с закругленными синеватыми углами. Компоненты соединения для Flash MX 2004 все еще используют старые компоненты Flash MX UI, за исключением того, что эти версии используют первоначальные серые оболочки. Более новые версии UI компонентов Flash MX 2004 не используются ни одной версией компонентов соединения. Пока внешний вид и возможности старых компонентов не обновляются так, имейте в виду, что старые UI компоненты совместимы со всеми старыми выпусками Flash Player 6, в то время как новые UI компоненты совместимы только с последними выпусками Flash Player 6 и выше.

Обзор компонентов соединения

Существует 16 компонентов в комплекте компонентов соединения. Как и все компоненты Macromedia, каждый компонент содержит несколько встроенных методов ActionScript и свойств, что позволяет изменять внешний вид каждого компонента, чтобы он соответствовал вашем индивидуальным запросам.

Некоторые разработчики обсуждают трудности создания пользовательских оболочек. Ищите архив FlashCoder или Wiki на http://chattyfig.figleaf.com для обсуждения различных вопросов.

Чафик Казоун написал следующее руководство по skinning'у для компонентов Flash MX 2004:
http://www.macromedia.com/devnet/mx/flash/articles/skinning_2004_print.html

Вы можете использовать компоненты соединения для быстрой настройки FlashCom application, которое подключает одного и более пользователей к приложению и дает возможность, находясь в реальном времени, согласовывать действия участников. Многие компоненты могут публиковать аудио/видео потоки на FlashCom, пока другие сосредотачиваются на тексте и разделении данных.

После того как Вы установили компоненты соединения, Вы можете просмотреть эти компоненты во Flash:

Если Вы только что установили компоненты и у Вас запущен Flash – выйдите и перезапустите Flash.

Откройте панель Components, выбирая Window → Development Panels → Components.

На панели Components раскройте узел Communication Components, как показано на рисунке 9.

Рисунок 9. Компоненты соединения на панели Components

Компоненты соединения самостоятельно и неограниченно работают на Вашем FlashCom Server; каждый может подключиться к Flash movie и начать использовать FlashCom application, к которому привязан ролик. По этой причине, мы настоятельно рекомендуем, чтобы Вы ограничили Ваши разработки примерами этой части на испытательном сервере.

Итоговый перечень компонентов кратко приведен в таблице 2. После описания каждого из компонентов, мы покажем Вам, как создать некоторые простые FlashCom applications, используя их.

Таблица 2. Компоненты соединения

Название компонента	Описание	Примечания
AudioConference	Дает возможность двум и более пользователям, соединяться друг с другом	Вложенные компоненты CheckBox, ListBox, PushButton и ScrollBar

AVPresence	Позволяет пользователю отправлять аудио и/или видео поток другим пользователям	Нет вложенных компонентов; смотрите VideoConference и SetBandwidth
Chat	Создает основанный на тексте интерфейс передачи сообщений для Flash movie	Вложенные компоненты PushButton и ScrollBar; смотрите UserColor
ConnectionLight	Показывает цветной круг, указывающий на статус связи	Нет вложенных компонентов; смотрите SimpleConnect
Cursor	Отображает местоположение и логины пользователей	Нет вложенных компонентов; смотрите UserColor
PeopleList	Показывает имена всех пользователей, подключенных к FlashCom application	Вложенные компоненты ListBox и ScrollBar
PresentationSWF	Показывает синхронизированное слайд-шоу, под звуковым контролем различных пользователей	Вложенный компонент PushButton
PresentationText	Показывает синхронизированные текстовые слайды, под звуковым контролем различных пользователей	Вложенные компоненты ListBox, PushButton, and ScrollBar
RoomList	Дает возможность разработчику создать шлюз к другим приложениям, запущенным на Вашем FlashCom Server	Вложенные компоненты ListBox, PushButton, и ScrollBar
SetBandwidth	Позволяет пользователю управлять тем, сколько полос пропускания может использоваться объектами Камера и Микрофон	Вложенные компоненты ListBox, PushButton, и ScrollBar
SimpleConnect	Устанавливает первичное соединение Flash movie client с FlashCom application	Вложенные компоненты PushButton
UserColor	Позволяет каждому пользователю выбирать цвет, представляющий его идентификатор в компонентах Chat и Cursor	Вложенные компоненты ComboBox and ScrollBar; затрагивают Chat and Cursor
VideoConference	Создает Flash-клиента, способного отображать различные требования компонента AVPresence	Автоматически включает AVPresence
VideoPlayback	Обеспечивает быстрый пользовательский интерфейс для воспроизведения записанных потоков	Нет вложенных компонентов
VideoRecord	Записывает и публикует аудио и видео потоки из Flash movie на FlashCom application	Нет вложенных компонентов
Whiteboard	Показывает панель с рисунками и подписями инструментов	Нет вложенных компонентов

Требования со стороны сервера

Каждый компонент соединения требует одного или более файла Server-Side ActionScript (SSAS), которые используют расширение .asc. Эти файлы находятся в папке scriptlib, которая находится в директории установки FlashCom'a, и автоматически устанавливаются вместе с Flash Communication Server MX. К счастью, Вы можете быстро задействовать компоненты соединения (т.е., сделать их доступными для использования Вашими client-side applications), загружая файл components.asc в документ main.asc любого приложения FlashCom. Файл components.asc загружает другие файлы .asc в требуемое приложения, чтобы задействовать все компоненты соединения. В качестве альтернативы, Вы можете загрузить файл .asc для каждого компонента индивидуально, чтобы минимизировать потребности в памяти для Вашего приложения FlashCom.

Каждый компонент соединения требует взаимодействия со стороны клиента и со стороны сервера. В случае, когда Вы используете компоненты соединения в Flash movie на стороне клиента, компонент обращается на сервер, к компоненту с таким же именем. Например, если в Вашем Flash movie есть компонент Chat, компонент автоматически создаст класс с именем FCChatClass. Когда Chat потребует соединения с FlashCom application, server-side application создает класс FCChat, предназначенный для связи с клиентским FCChatClass.

Для достижения целей этой части и этих примеров, рассмотрим основные выполняемые функции компонента framework относительно компонентов соединения.

Загрузите компонент framework, который дает возможность клиентским приложениям использовать компоненты соединения:

Создайте директорию, например, myApplication, для Вашего приложения FlashCom в директории applications, где установлен Ваш FlashCom.

Используя текстовый редактор, типа того, который входит во Flash MX Pro 2004, создайте текстовый документ, содержащий следующий код:

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

Сохраните этот документ как main.asc, в директории, созданной Вами ранее.

Вы можете продолжить добавление SSAS кода в main.asc, чтобы обеспечить выполнение функций, помимо компонентов соединения. Позже в этой части, Вы создадите базовое приложение, использующее код, показанный в предыдущих шагах.

Если Вы не хотите загружать server-side code для связи всех компонентов соединения в Вашем FlashCom application (приложении для FlashCom), Вы можете загрузить только server-side code для специфических компонентов, которые Вы используете в Вашем Flash приложении. Например, чтобы загрузить файл для компонента Chat сервер использует:

```
load("framework.asc");
load("components/chat.asc");
```

в первой строке Вашего документа main.asc.

Общие методы компонентов соединения

Если Вы будете редактировать один из символов компонентов соединения во Flash MX 2004 и выше, Вы найдете несколько методов, являющимися общими для всех используемых компонентов.

В следующих упражнениях Вы поместите весь ActionScript в кадр 1 слоя Actions. Создайте новый слой (Layer) Actions для присоединения скриптов и поместите его сверху панели времени (Timeline).

Чтобы создать типовой компонент, поместите экземпляр компонента ConnectionLight на сцену Flash-документа, кликните правой кнопкой мыши (для Windows) или левой кнопкой мыши (для Mac) на экземпляре и выберите Edit. Выберите кадр 1 слоя Actions и откройте панель Actions (F9)

init()

Метод init() инициализирует каждый компонент экземпляра со свойствами name и prefix также как и с другими свойствами, характерными компонентам этого типа. Prefix – это строка, используемая для помощи при идентификации каждого компонента экземпляра на сервере компонента framework FlashCom-приложения. Каждому экземпляру назначен prefix, используемый следующее условие:

```
class name + "." + instance name + "."
```

Например, экземпляр компонента ConnectionLight, названный connLight_mc, использующий панель свойств, имел бы следующий prefix:

```
FCConnectionLightClass.connLight_mc.
```

Каждому экземпляру компонента так же дается свойство name, которое в точности такое же, как свойство name_ экземпляра movie clip этого компонента. Если экземпляр

компонента не был назван в панели свойств, то свойству name присваивается значение "_DEFAULT_".

connect()

Метод connect() каждого компонента соединения устанавливает связь с компонентами экземпляра FlashCom application. Метод требует только один аргумент: объект NetConnection, который уже успешно соединился с FlashCom application. Таким образом, компонент совмещает передачу прямых и обратных пакетов по предварительно установленной связи. Для экземпляра SimpleConnect, определенного внутри списка компонентов соединения, метод connect() вызывается автоматически.

Метод connect() используя prefix, установленный методом init(), обсуждаемым ранее, вызывает метод connect() со стороны сервера для того же самого компонента класса. Код на стороне сервера, вызванный клиентом с помощью метода NetConnection.call(), просматривает значение prefix и «подключенное» имя метода как строку, наряду с любыми другими параметрами, необходимыми для компонента соединения. Каждый компонент требует, чтобы его метод connect() был назван. Это неодходимо, чтобы инициализировать новый компонент в пределах FlashCom application.

Мы не используем термин соединение здесь, подразумевая фактическую связь экземпляра NetConnection или пользовательскую связь с FlashCom application. Даже несмотря на то, что у Вас может быть несколько экземпляров компонентов соединения Flash movie, все компоненту могут быть связанны с FlashCom application через один экземпляр NetConnection. Связь компонентов соединения легко устанавливается между одним и более разделенными удаленными объектами или потоками, связанными с выполняемыми функциями компонента.

close()

Каждое соединение компонента соединения с FlashCom application может быть завершено с помощью метода close(). Этот метод вызывает метод close() со стороны сервера для класса компонента, используя одинаковые методологии NetConnection.call(). Следует отметить, что метод close() не вызывает метод NetConnection.close() от Flash клиента; подключение к FlashCom application все еще поддерживается, даже если экземпляр компонента закрыл подключение.

onUnload()

Метод onUnload() каждого компонента соединения просто вызывает тот же самый метод close(), обсуждаемый ранее. OnUnload() обработчик вызывается тогда, когда экземпляр компонента удален со сцены Flash movie. Экземпляр может быть удален, если бегунок времени (playhead) перейдет в кадр, где экземпляра не существует или клиентский ActionScript использует метод MovieClip.removeMovieClip().

setUsername()

Многие компоненты соединения, такие как AVPresence и PeopleList, содержат метод setUsername(). Этот метод определяет имя пользователя как строку (string) и обычно отображает ее где-нибудь в пользовательском интерфейсе компонента. Например, если экземпляр компонента AVPresence, имеющий имя user_1_mc, вызывает метод setUsername() со строкой "Alan", имя окажется сверху слева от экземпляра AVPresence, когда пользователь начнет передачу аудио и видео. Следующая строка демонстрирует эти возможности:

```
user_1_mc.setUsername("Alan");
```

Сведение воедино компонентов соединения

Рассмотрим индивидуальные особенности каждого компонента (в алфавитном порядке). Вы можете быстро взять и использовать компонент SimpleConnect, который, как и множество других компонентов, полагается на заранее установленное соединение.

Здесь приведены некоторые указания, которые помогут Вам понять индивидуальные разделы о компонентах:

Подобно client-side ActionScript, класс компонента соединения FlashCom имеет имя: FCComponentNameClass; например, имя класса компонента AudioConference будет иметь вид FCAudioConferenceClass.

Большинство компонентов соединения имеют параметры, которые могут быть настроены в панели Properties (Свойства) или панели Component Inspector (Инспектор компонентов). Если есть параметры, которые настраиваются через UI параметра, они включаются их в описание каждого компонента. Все параметры компонента могут быть также настроены, используя свойства компонента ActionScript. Например, параметр LatencyThreshold компонента ConnectionLight настраивается с использованием свойства threshold данного компонента. Некоторые параметры компонентов, такие как свойство username компонента Chat, настраиваются только через ActionScript. Разработчик компонентов сам определяет, какие свойства компонентов выставить на панели Properties или на панели Component Inspector. Параметры компонентов соединения могут быть выставлены в диалоговом окне Component Definition (доступ по нажатию правой клавишей мыши на символе компонента в библиотеке). Для любого компонента, написанного на ActionScript 2.0 (AS 2.0), например, Flash MX 2004 UI Components, параметры могут быть так же выставлены путем определения свойств в его файле .as. Компоненты соединения Macromedia не написаны на AS 2.0; поэтому, настраиваемые параметры не могут быть выставлены UI через файлы .as. В дальнейшем, надеюсь, эта ситуация подправится.

Если ничего дополнительно не определено специально, то компонент не включает в себя какие-либо другие компоненты Flash UI (смотрите таблицу). Если компоненту требуются другие компоненты, они будут включены в Flash move автоматически.

AudioConference

Компонент AudioConference дает возможность общаться друг с другом двум и более пользователям, присоединившись к одному или более аудио потокам. Каждый пользователь, который желает использовать аудио поток, должен иметь микрофон, совместимый с Flash Player 6 или выше. Экземпляр компонента (рисунок 10) показывает всех подключенных пользователей. Это позволяет каждому пользователю по щелчку на кнопку Talk, начать передачу аудио потока. Пользователь также может выбрать режим Auto и компонент будет сам автоматически начинать и останавливать аудио поток, когда пользователь начинает и заканчивает говорить. Когда пользователь говорит, индикатор слева от имени пользователя становится зеленым. Когда пользователь заканчивает говорить, индикатор возвращается к серому цвету.

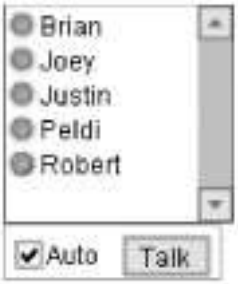

Рисунок 10. Компонент AudioConference

Вложенные компоненты Flash UI

Компонент AudioConference использует четыре компонента Flash UI: CheckBox, ListBox, PushButton, and ScrollBar.

Обзор действий компонента

Когда компонент AudioConference загружается в Flash movie, он использует существующий объект NetConnection (который подготовлен компонентом SimpleConnect), создает или соединяется с непродолжительным RSO, называемым FCAudioConference.audio.

Всякий раз, когда экземпляр компонента соединения использует RSO, любые изменения сделанные RSO, распространяются на всех пользователей подключенных к одному и тому же экземпляру приложения. Компоненты соединения не запрещают пользователям одного и того же экземпляра приложения использовать данные друг друга.

RSO следит за каждым подключенным пользователем. Когда пользователь нажимает на кнопку Talk, компонент присоединяет выход микрофона к экземпляру NetStream и устанавливает свойству RSO значение true. Название этого свойства основано на логине пользователя, который обеспечивается компонентом framework. Когда другие клиенты получают это обновленное значение с помощью обработчика SharedObject.onSync(), аудио поток другого пользователя запускается. Когда пользователь отпускает кнопку Talk, компонент устанавливает свойству RSO значение false и выход микрофона пользователя отключается от потока.

AVPresence

Компонент AVPresence, безусловно, один из самых популярных компонентов соединения, позволяющих пользователю подключаться к FlashCom application и отправлять аудио и/или видео потоки другим пользователям. Экземпляр AVPresence показывает скрытую область для выхода камеры пользователя, как показано на рисунке 11. Пользователь может выборочно приглушить аудио и/или видео поток. Компонент показывает уровень активности микрофона справа.

Рисунок 11. Компонент AVPresence

Один из специфических атрибутов компонента AVPresence это то, что он может быть использован для отправки любым пользователем аудио/видео потока или получения от другого пользователя аудио/видео потока. Экземпляр AVPresence устанавливает уникальный идентификатор внутри FlashCom приложения. Если Вы расположите экземпляр компонента с именем user_1_mc внутри Flash movie (файл .swf), каждый кто загрузит Flash move и подключится к FlashCom application может задействовать экземпляр user_1_mc, чтобы начать передачу аудио/видео. Как только пользователь начал передавать поток с помощью экземпляра user_1_mc, любые другие пользователи, которые загрузили тот же Flash movie, будут видеть этот поток в своих экземплярах user_1_mc. Если Вы хотите обеспечить многоканальный синхронный доступ пользователей к выполняемым

функциям компонента AVPresence, Вам нужно будет создать уникальный экземпляр компонента AVPresence для каждого пользователя.

Как только пользователь прекращает управление экземпляром AVPresence, другой пользователь может использовать тот же самый экземпляр, чтобы начать передачу своего аудио/видео потока.

Чтобы пользователь получил доступ к компоненту AVPresence, должны произойти следующие события:

Компонент AVPresence должен быть подсоединен к активному экземпляру NetConnection приложения FlashCom. Это соединения может быть создано с помощью компонента SimpleConnect или с помощью пользовательского ActionScript.

Пользователь должен выбрать логин (login, имя пользователя) в компоненте SimpleConnect или AVPresence.setUsername() должен быть вызван со значением строки. Если значение логина не установлено, экземпляр AVPresence выведет текст: "Login to Send Audio/Video."

Как только имя пользователя будет установлено, пользователь может щелкнуть ссылку Send Audio/Video внутри экземпляра AVPresence, чтобы начать передачу аудио/видео потока для FlashCom application. Любой другой пользователь, просматривающий этот экземпляр AVPresence, будет получать потоки аудио и видео.

Чтобы остановить передачу аудио/видео потока, пользователь должен навести курсор мыши на область видео в экземпляре и нажать кнопку с изображением красного крестика в кружочке в нижнем левом углу компонента, как показано на рисунке. как только поток перестанет передаваться, любой другой пользователь сможет начать использовать этот же экземпляр AVPresence.

Клиентские параметры компонента

Экземпляр компонента AVPresence имеет шесть уникальных настроек на панели Properties или во вкладке Parameters на Component Inspector. Эти настройки управляют качеством и скоростью передачи данных, используемых компонентом:

Sync Speed

Контролирует частоту (в кадрах в секунду - fps), с которой обновленные данные поступают от к клиента на сервер. Настройка управляет частотой обновления RSO, используемого компонентом AVPresence. Более высокие значения увеличивают пропускную способность данных и обновляют дисплей уровня микрофона более быстро. По умолчанию значение равно 3 кадрам/сек. Эквивалент ActionScript для этого свойства - updateFps.

Video Width

Устанавливает, в пикселях, ширину области данных, получаемых с камеры, используемой компонентом AVPresence. Не путайте эту ширину с масштабом; Video Width определяет, сколько будет отрезано от области видео локальной камеры. Например, если область данных камеры - 640 x 480 пикселей, установка Video Width 320 подрезала бы 160 пикселей с левой и правой стороны области сбора данных. Значение по умолчанию для этой установки - 120 пикселей. Эквивалент ActionScript для этого свойства - vidWidth.

Video Height

Устанавливает, в пикселях, значение высоты области сбора данных с камеры. Как и Video Width, устанавливает границу обрезания области сбора данных. Значение по умолчанию - 120 пикселей. Эквивалент ActionScript для этого свойства - vidHeight.

Video Bandwidth

Ограничивает количество данных, в битах в секунду, которое может использоваться видео частью объекта NetStream, используемого компонентом AVPresence. Значение 0 позволяет видео использовать максимальную пропускную способность, доступную при подключении Flash movie к FlashCom Server. Значение по умолчанию - 12 000 битов в секунду, эквивалентных 11.7 Кб/сек. Эквивалент ActionScript этого свойства - vidBandwidth.

Video Quality

Устанавливает качество (от 0 до 100) видео объекта NetStream, используемого компонентом AVPresence. Более высокие значения приводят к лучшему качеству, но одновременно увеличивают требования по пропускной способности. Если Вы превысите допустимую пропускную способность, то камере придется обрезать видео область, поскольку они отправляются объекту NetStream. Значение по умолчанию – 0. Оно позволяет потоку использовать максимальное качество для пропускной способности, предоставленной потоку. Эквивалент ActionScript для этого свойства - vidQuality.

Video FPS

Это значение управляет скоростью передачи кадров видео части объекта NetStream, используемого компонентом AVPresence. Более высокая скорость передачи кадров приводит к более плавному движению, полученному с камеры пользователя, но увеличит требования по пропускной способности для потока. Значение по умолчанию - 10 кадров/сек. Эквивалент ActionScript для этого свойства - vidFps.

Настройки Video Width, Video Height, Video Bandwidth, Video Quality и Video FPS на панели Properties для экземпляра AVPresence игнорируются, если компонент SetBandwidth также используется во Flash movie. В этом случае, параметры настройки экземпляра SetBandwidth отменяют настройки экземпляра AVPresence.

Обзор действий компонента

ActionScript компонента AVPresence, создает несколько объектов, в пределах метода connect(). Эти объекты позволяют многочисленным экземплярам компонента AVPresence общаться друг с другом, если они подключены к одному и тому же FlashCom application.

Метод connect() создает экземпляры NetStream с именем ns, которые используются чтобы опубликовать или подписаться на аудио/видео поток.

Как и большинство компонентов соединения, экземпляр AVPresence создает RSO, название которого сформировано из значения prefix компонента. Для экземпляра AVPresence, значение prefix имеет вид:

```
"FCAVPresence." + instance name + "."
```

Это значение соединяется с прототипом значения soName и "av". Например, если Вы поместили экземпляр компонента AVPresence на сцену и назвали его speaker_mc, имя созданного RSO будет FCAVPresence.speaker_mc.av.

RSO требуются два ключевых свойства: broadcast и speakerID. Свойство broadcast - Булевская (логическая) переменная (принимает значения ложь или истина). Если экземпляр AVPresence передает поток, значение broadcast устанавливается TRue. Если поток открыт (то есть никто ничего не передает) значение broadcast устанавливается false. Свойство speakerID следит за идентифицирующей строкой, указывающей, который пользователь публикуется в потоке. Строка логина для каждого пользователя сохранена в экземпляре AVPresence и называется userID. Строка определяется с помощью ActionScript кода, созданного для компонента AVPresence.

Chat

Компонент Chat создает основанный на текстовых полях интерфейс передачи сообщений для Flash movie. Интерфейс компонента, как показано на рисунке, представляет собой текстовую область, полосу прокрутки, пользовательскую входную текстовую область и кнопку Send. Вы можете использовать компонент Chat с компонентом PeopleList для создания базового приложения чат. Компонент UserColor может также использоваться вместе с компонентом Chat, чтобы назначать цвет для текста каждого пользователя в текстовой области.

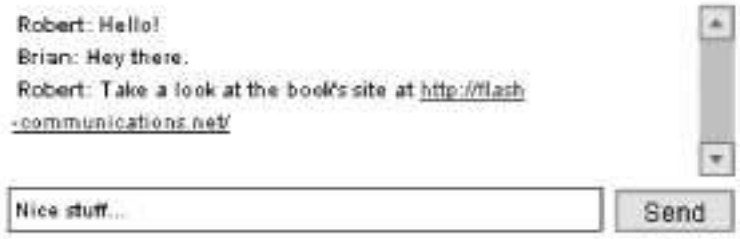

Рисунок 12. Компонент Chat

Вложенные компоненты Flash UI

Компонент Chat использует два Flash UI компонента: PushButton и ScrollBar. Вы можете изменить внешний вид этих компонентов, изменяя соответствующие оболочки символов в папке Component Skins в панели Library Flash документа.

Обзор действий компонента

Когда Вы добавляете экземпляр компонента Chat в Flash movie, экземпляр должен быть подключен к Вашему FlashCom application. Как и для всех компонентов соединения, это соединение может быть создано с помощью экземпляра SimpleConnect или Вы можете создать свой объект NetConnection и вызвать непосредственно метод Chat.connect().

Если Вы решили создать собственное подключение, Вы должны установить глобальный объект клиента внутри Серверной части (Server-Side) ActionScript Вашего FlashCom application. Если Вы используете компонент SimpleConnect, его клиентский объект уже создан для Вашего приложения. Код ActionScript на стороне сервера, содержащийся в документе chat.asc, восстанавливает имя каждого пользователя с помощью метода Component.getClientGlobalStorage(). Имя каждого пользователя хранится в свойстве, называющимся username. Значение username прикрепляется к каждому сообщению, которое передается всем остальным пользователям.

Два RSO создаются Server-Side ActionScript из компонента Chat: message и history. Полные названия этих RSO создаются используя точно такую же формулу, которая была приведена при обсуждении компонента AVPresence. Значение prefix каждого отдельного объекта начинается с "FCChat.", далее следует клиентское название экземпляра компонента.

Каждый раз, когда пользователь вводит сообщение в поле ввода текста компонента Chat и нажимает кнопку Send, метод call() клиентского экземпляра Chat отправляет текстовое сообщение методу FCChat.sendMessage() на стороне сервера. Кнопка Send компонента Chat уже настроена так, чтобы «прислушиваться» к клавише Enter, когда поле ввода текста находится в фокусе. Если пользователь, вводя сообщение в текстовое поле, нажимает клавишу Enter (или Return), автоматически отправляется сообщение на FlashCom application.

Как только метод sendMessage() получает текстовое сообщение, код на стороне сервера обрабатывает сообщение, вставляет в него значение имени пользователя отправителя и преобразует текст, начинающийся с "http:" или "www." в понятные для

HTML «слова». Затем текст сообщения отправляется методу message() отдельному объекту message. Этот отдельный объект доставляет текст сообщения каждому подключенному пользователю.

Удаленный объект history также обрабатывается компонентом FCChat на стороне сервера. Этот объект имеет одно свойство, также называющееся history, которое хранит массив, в котором каждый индекс – это сообщение, отправленное клиентом. Всякое новое сообщение, отправленное экземпляром Chat, помещается в массив history.

Конфигурируемые атрибуты стороны сервера

В отличие от других компонентов соединения, компонент Chat имеет несколько особенностей, которыми можно управлять, используя настройки документа chat.asc со стороны сервера. Вы можете найти этот документ в папке scriptlib/components. Следующий код нужно вставить между 33 и 35 строками документа scriptlib/components/chat.asc:

```
FCChat.prototype.histlen  = 250;  // Maximum history length.
FCChat.prototype.persist  = true;  // Whether to save history.
FCChat.prototype.allowClear = true;  // Allow clients to clear history.
```

Свойство histlen контролирует максимум length массива history, которое показывает максимальное число хранящихся сообщений (по умолчанию 250). Увеличение этого числа позволяет сохранять более длинную историю чата.

Свойство persist решает, сохранять ли history объекта. Если persist установлено TRue (по умолчанию), то история чата сохраняется от одной сессии до другой. Даже после того, как приложение, использующее компонент Chat, будет выгружено, история сохраняется. Если свойство persist устанавливается false, то каждая сессия чата начинается с пробела, прошлые сообщения загружаются в экземпляр Chat когда он запускается.

Свойство allowClear определяет, может ли метод FCChat.clearHistory() со стороны сервера быть вызван клиентским ActionScript. По умолчанию установлено значение true, позволяющее клиентскому ActionScript очищать историю чата. Чтобы препятствовать клиентскому коду очищать историю allowClear устанавливается значение false.

ConnectionLight

Компонент ConnectionLight позволяет визуально показывать состояние подключения к FlashCom application.

Состояние показывается в виде кружка, который может окрашиваться в один из четырех цветов:

Серый (grey)

Указывает, что должна быть предпринята попытка подключения или закрытие предыдущего подключения.

Зеленый (green)

Показывает стабильное подключение (через экземпляр NetConnection) к FlashCom application.

Красный (red)

Показывает, что попытка подключения к FlashCom application не удалась или признано негодным. Этот цвет может также показывать, что текущее подключение было неожиданно потеряно или специально закрыто.

Желтый (yellow)

После установки удачного соединения к FlashCom application, клиентский ролик может задерживаться с передачей пакетов. Если период задержки превышает указанный порог, индикатор стал желтым.

Компонент ConnectionLight повторно проверяет статус соединения (подключено или не подключено) каждые пол секунды (500 миллисекунд).

Если пользователь щелкнет на экземпляр ConnectionLight (цветной кружок), Flash выведет информационный overlay как показано на рисунке 13.

Рисунок 13. Информация, выводимая компонентом ConnectionLight

Overlay содержит больше информации о состоянии соединения:

Latency

Среднее время ожидания, в миллисекундах, при передаче пакетов от клиентского ролика к FlashCom Server. Более высокие значения времени ожидания могут указывать на медленное соединение (обычно с клиентской стороны), временную перегрузку сети или перегрузку FlashCom Server.

Up

Скорость передачи данных в битах, отправляемых клиентским роликом на FlashCom Server. Используется для контроля пропускной способности, используемой объектами подключения и при публикации аудио и видео потока через экземпляр NetStream.

Down

Скорость передачи данных в битах, полученных клиентом от FlashCom сервера. Операции синхронизации с отдельными объектами данных и получаемые потоки вносят свой вклад в полное значение.

Значения Up и Down автоматически отображаются в бит/сек, Кбит/сек или Мбит/сек и основываются на скорости подключения.

Параметры компонента со стороны клиента

Компонент ConnectionLight имеет два параметра настройки, которые могут быть сконфигурированы на уровне экземпляра на панели Properties или панели Component Parameters:

Measurement Interval

Этот параметр соответствует свойству interval и определяет время, в секундах, которое экземпляр ожидает, перед тем как запросить FlashCom сервер для обновления статистики подключения. По умолчанию установлено значение 2 секунды. Если Вы увеличите это значение, например, установите 5 или 10 секунд, то пропускная способность для трафика данных увеличится. А если Вы уменьшите это значение до 1, то пропускная способность станет ниже. Если Вы не хотите увеличить нагрузку на пропускную способность Вашего канала, то используйте более высокие значения.

Latency Threshold

Если время прохождения пакета между локальным Flash movie и FlashCom сервером превышает определенное значение, иконка экземпляра ConnectionLight из зеленой превращается в желтую. Этот параметр соответствует свойству tHReshold, по умолчанию установлено значение 0.1 секунда (100 миллисекунд). Это значение сообщает другим участникам, подписавшимся на поток, будет ли аудио/видео поток иметь значительные задержки и будут ли иметь задержки текстовых сообщений в чате.

В любое время, если локальное приложение Flash movie подключено к FlashCom приложению, пакеты данных посылаются от клиента к FlashCom Server и обратно, чтобы знать, что клиент все еще подключен. Также FlashCom Server всегда знает, сколько времени ожидания между ним и любым подключенным клиентом.

Как работает компонент

Чтобы функционировать должным образом, экземпляр компонента ConnectionLight должен быть связан с объектом NetConnection, с помощью Flash movie, использующего ConnectionLight.connect(). Используется экземпляр SimpleConnect и определяется экземпляр ConnectionLight в списке компонентов соединения, вызываемых ConnectionLight.connect() автоматически. Иначе, Вам нужно вызвать метод connect() экземпляра ConnectionLight вручную из Вашего кода ActionScript.

В отличие от других компонентов соединения, компонент ConnectionLight не использует никаких отдаленных отдельных объектов или объектов NetStream. Он использует только метод call(), чтобы соединить методы между клиентским и серверным частями ActionScript. Клиентский метод ConnectionLight.connect() отправляет метод call() методу FCConnectionLight.connect() на стороне сервера. Метод connect() на стороне сервера используя интервал измерения, вычисляет с каким интервалом передается информация между FlashCom приложением и клиентским Flash move. Основываясь на интервале, с которым передается информация, Client.getStats() периодически вычисляет полосу пропускания для статистических данных и посылает эти данные Flash-клиенту через метод Client.call() на стороне сервера.

Возможности ConnectionLight без SimpleConnect

Когда компонент ConnectionLight используется без помощи компонента SimpleConnect, он вызывает ConnectionLight.connect() Вашего ActionScript. Следующий код вызывает метод connect() экземпляра ConnectionLight, который называется connLight_mc, когда первое сообщение onStatus() получено от объекта NetConnection с именем app_nc:

```
var app_nc:NetConnection = new NetConnection();
app_nc.onStatus = function (info) {
 if (info.code == "NetConnection.Connect.Success") {
  trace("--successful connection");
 }
 if (!initConnLight) {
  initConnLight = true;
  connLight_mc.connect(this);
 }
};
app_nc.connect("rtmp:/my_app");
```

Cursor

Компонент Cursor обеспечивает визуальное представление позиции мыши каждого подключенного пользователя Flash movie. Компонент отображает имя каждого пользователя, входящего в систему, под изображением курсора, указывающего позицию пользователя, как показано на рисунке 14. Этот компонент также выделяет пользователя

цветом, если используется вместе с компонентом UserColor, который будет обсуждаться позже.

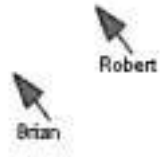

Рисунок 14. Компонент Cursor показывает позицию курсора каждого пользователя

Как работает компонент

Когда экземпляр Cursor инициализируется, он создает два RSO: cursors и mov, добавляя клиентский курсор к FlashCom application и отслеживая его движение. Когда новый клиент загружает Flash movie и подсоединяется к тому же FlashCom application, cursors объекта с именами пользователя и цветами обновляются у пользователей, подключившихся ранее. FlashCom передает эти обновления всем, давая возможность каждому клиенту добавить новый курсор для каждого участника. Аналогично, если участник перемещает свой курсор мыши, свойства объекта обновляются и каждый подписчик получает новые координаты курсора мыши данного участника.

PeopleList

Компонент PeopleList отображает имена всех пользователей, подключенных к данному FlashCom серверу, как показано на рисунке 15. Когда новый пользователь присоединяется к приложению, его имя добавляется к экземпляру PeopleList. Когда пользователь выходит, его имя удаляется из списка. Если этот компонент используется вместе с компонентом SimpleConnect, то имя пользователя отображается также как текст, введенный пользователем в текстовое поле компонента SimpleConnect при регистрации.

Ели Вы хотите разрешить какие-либо действия для отдельных пользователей, Вы должны добавить обработчик изменений в экземпляр ListBox и назвать его people_lb внутри компонента PeopleList.

Рисунок 15. Список участников компонента PeopleList

Вложенные компоненты Flash UI

Компонент PeopleList использует два вложенных Flash UI компонента: ListBox и ScrollBar.ScrollBar используется внутри компонента ListBox, как и во Flash MX UI. Вы можете изменить внешний вид этих компонентов, подобрав им соответствующий внешний вид в папке Component Skins, находящейся на панели Library Flash-документа.

Обзор действий компонента

Компонент PeopleList, как и компонент Chat, придерживается процесса инициирования, выбранного стороной сервера. Как и компонент Chat, компонент PeopleList использует метод Component.getClientGlobalStorage() на стороне сервера. Когда клиентский экземпляр PeopleList загружается в Flash movie, создается свойство RSO users. Users хранит значение логина пользователя, созданного серверным ActionScript в scriptlib/components/people.asc, наряду с соответствующими именами пользователя. Когда происходит обновление этого объекта, экземпляр people_lb, внутри компонента PeopleList, удаляет существующий набор отображающихся имен пользователя и добавляет новый

список имен пользователя. По умолчанию значения label и data каждого экземпляра people_lb являются именем пользователя.

Чтобы хранить значение логина каждого подключенного пользователя, как значение data каждого пункта экземпляра people_lb, Вы можете изменить 46 строку Вашего кода ActionScript, который находится на слое actions компонента PeopleList:

```
this.owner.people_lb.addItem(this.data[i], i);
```

Новый код использует строку логина пользователя i, как данные для нового пункта ListBox.

Что такое пассивный режим?

Если Вы посмотрите клиентский и серверный ActionScript код компонента PeopleList, Вы обратите внимание на ссылки "fc_lurker". Компонент PeopleList поддерживает пассивный режим, позволяя пользователю скрыть свое присутствие в компоненте PeopleList, если имя пользователя не возвращается объектом Component.getClientGlobalStorage(). В этом случае, компонент PeopleList назначает значение " fc_lurker" для users RSO. Когда клиентсикй метод onSync() обновляет users RSO, имя пользователя не будет добавлено в экземпляр people_lb, если установлено значение " fc_lurker".

PresentationSWF

Компонент PresentationSWF позволяет Вам создать среду демонстрации слайдов, которые могут просматривать многие пользователи. Компонент синхронизирует воспроизведение Flash move (.swf файл) версии со звуком и версии с картинкой, как показано на рисунке 16. Во Flash move, содержащий слайд-шоу, будут загружены исходные версии Flash move звука и изображения с помощью компонента PresentationSWF.

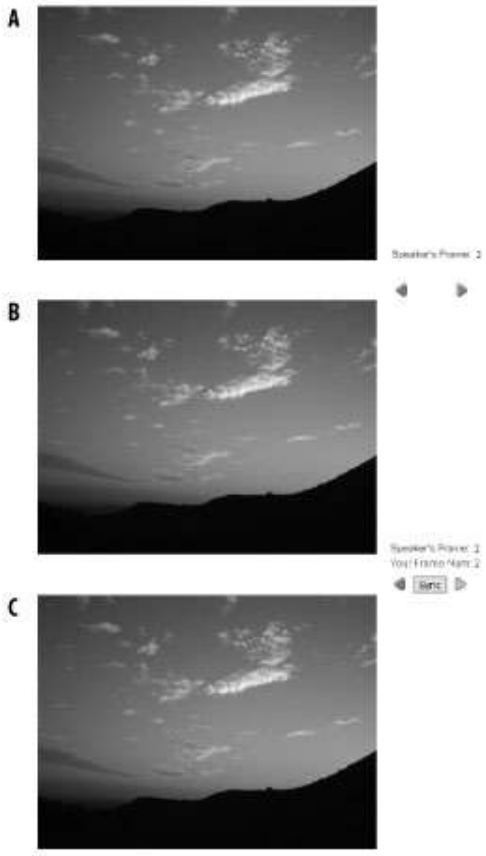

Рисунок 16. Компонент PresentationSWF в его двух режимах: (a) звук и (b) просмотр; оба режима загружаются в один и тот же ролик (c)

Вложенные компоненты Flash UI

Компонент PresentationSWF использует компонент PushButton Flash UI. Компонент PushButton используется кнопкой Sync внутри пользовательского интерфейса компонента PresentationSWF. Вы можете изменить внешний вид компонента PushButton, изменяя соответствующие символы оболочки в папке Component Skins на панели Library Flash-документа.

Параметры компонента со стороны клиента

Экземпляр компонента PresentationSWF имеет две настройки на панели Properties или панели Component Parameters:

PresentationSWF

Имя файла, обобщенного Flash move, которое будет загружено с помощью компонента PresentationSWF. Этот параметр, соответствующий свойству swfFile должен быть идентичным в звуковых и видео версиях компонента. Вы можете использовать относительный и абсолютный путь, для указания местоположения обобщенного ролика. Значение по умолчанию - simple_preso.swf (типовой ролик FlashCom).

Viewer Buttons Enabled

Это значение, соответствующее параметру viewerButtons, определяет, могут ли средства просмотра управлять уже просмотренными слайдами с помощью кнопок Forward, Back и Sync. По умолчанию установлено значение true, которое позволяет выводить эти кнопки на экран. Средство просмотра может перейти к текущему звуковому значению, используя кнопку Sync, но не может пропустить его. Если установлено значение false, средство просмотра не может управлять слайд-шоу.

Обзор действий компонента

Типовое использование этого компонента требует:

Content Movie

Ролик, содержащий слайд-шоу. Его главная дорожка времени должна содержать по одному слайду в кадре и команду stop() в первом кадре. Например, презентация из 10 слайдов должна содержать 10 кадров на дорожке времени (timeline).

Speaker Movie

Пользователь, управляющий слайд-шоу, должен установить _global.speakerMode значение true в своей версии исходного Flash move, содержащей компонент PresentationSWF. Может быть только один контролирующий (или говорящий) в приложении, использующем компонент PresentationSWF.

Viewer Movie

Каждый, кто смотрит слайд-шоу, должен использовать Flash move с установленным значением false для _global.speakerMode.

Вам не нужны две раздельные части Flash move (звук и картинка). Вы можете создать один Flash move с интерфейсом входа в систему, который позволяет определенному пользователю осуществлять управление презентацией, пока другие пользователи находятся в режиме просмотра.

Независимо от того, используете ли Вы компонент SimpleConnect или создаете собственный объект NetConnection, для подключения к FlashCom, Вам нужно установить значение глобальной переменной speakerMode значение true или false в Flash move,

содержащим компонент PresentationSWF. В версии со звуком следующий код перед или вместе с загрузкой экземпляра PresentationSWF на сцену:

```
_global.speakerMode = true;
```

Для версии с видео, переменной должно быть установлено значение false:

```
_global.speakerMode = false;
```

Экземпляр PresentationSWF ищет эти значения, чтобы определить, кто управляет отображением пользовательского интерфейса компонента.

Когда компонент PresentationSWF инициализируется, он создает RSO с именем presentation. Когда звук двигается назад или вперед по слайд-шоу, слот данных frameNum обновляется внутри RSO presentation. Затем, на стороне клиента метод onSync() экземпляра объекта с именем so, дает команду каждому подключенному Flash move реагировать соответственно.

PresentationText

Компонент PresentationText очень похож на компонент PresentationSWF, обсуждаемый в предыдущем разделе. С компонентом PresentationText Вы можете создать FlashCom приложение, в котором текстовые слайды управляются пользователем и синхронизируются с остальными подключенными картинками. Однако, в отличие от компонента PresentationSWF, компонент PresentationText не требует, чтобы оба Flash move предоставляли контекст слайд-шоу, не предоставляя при этом параметр Viewer Buttons Enabled. Кнопки Forward и Back видны только, если пользователь является предъявителем, а не участником. Компонент PresentationText фактически создатель и редактор текстовых слайдов. Как компонент PresentationSWF, звуковая версия может создать и редактировать текст, в то время как версия средств просмотра может только смотреть и управлять уже просмотренными слайдами презентации. Звуковая версия компонента также обеспечивает список слайдов в пределах навигационного окна списка, как показано на рисунке 17.

Рисунок 17. Компонент PresentationText в его двух режимах: (a) звук и (b) просмотр

Вложенные компоненты Flash UI

Компонент PresentationText использует три вложенных Flash UI компонента: ListBox, PushButton и ScrollBar. Вы можете изменить внешний вид компонента PushButton, изменяя соответствующие оболочки символов в папке Component Skins на панели Library Flash-документа. Если Вы хотите изменить внешний вид стрелок forward и back, Вы должны изменить графические символы внутри movie clip (папка Communication Components → Core Assets - Developer Only → PresentationText Assets → fc_presentationtext_arrow_elements).

Параметры компонента со стороны клиента

Экземпляр компонента PresentationText имеет только одну настройку на панели Properties или панели Component Parameters.

Speaker Mode

Принимая во внимание, что компонент PresentationSWF использует глобальную переменную speakerMode, компонент PresentationText обеспечивает переключение звук/видео на уровне экземпляра. Для звуковой версии компонента PresentationText, значение параметра установлено false. Этот параметр соответствует свойству speakerMode компонента PresentationText, а не глобальной переменной с таким же именем.

Как работает компонент

Компонент PresentationText имеет ту же самую структуру работы, как и компонент PresentationSWF, они оба используют RSO, чтобы синхронизировать информацию о слайдах среди подключенных пользователей. Однако, presentation RSO, созданный компонентом PresentationText, является набором постоянных данных, используемых слайдами, и может существовать постоянно. Эта особенность позволяет создать заготовку текстовых слайдов презентации заранее. Например, преподаватель мог создать заголовки и текст для каждого слайда заранее; информация каждого текстового слайда была бы сохранена в пределах постоянного presentation RSO. Когда преподаватель готов провести свой сетевой курс, поддержка звука компонента PresentationText автоматически загружает существующую информацию слайдов и отображает их на средствах просмотра.

RoomList

Компонент RoomList - возможно самый распространенный компонент соединения. Из всех компонентов соединения он требует самого большого объема «ручного кодирования». Вы не можете просто так перетащить этот компонент на сцену, даже с использованием компонента SimpleConnect. Компонент RoomList дает возможность Вам создать шлюз к другим экземплярам приложений, выполняющимся на вашем FlashCom Server. Когда Вы создаете FlashCom приложение, Flash move может соединиться с экземпляром по умолчанию или с определенным именем экземпляра приложения. Для приложения чат Вы можете приравнять экземпляр приложения к разделу чата. Компонент RoomList дает возможность показывать Flash move каждому из этих экземпляров. Ролик, содержащий компонент RoomList, однако, должен соединиться с приложением, полностью отдельным от приложения чат. Как показано на рисунке 18, пользовательский интерфейс компонента RoomList показывает список запущенных FlashCom приложений и позволяет Flash клиенту присоединять, создавать, или удалять экземпляры этих приложений.

Рисунок 18. Компонент RoomList организовывает вход

В отличие от других компонентов соединения, компонент RoomList используется отдельным экземпляром FlashCom application, а не тем, с которым в конечном счете соединятся клиенты.

Следующие шаги выделяют основные события приложения, использующего компонент RoomList:

- Пользователь загружает HTML страницу, содержащую Flash movie (SWF файл #1) и экземпляром компонента RoomList. Компонент RoomList подключается к FlashCom application (приложение A), которое контролирует один или более экземпляр другого FlashCom application (приложение B).
- Пользователь регистрируется в приложении A от SWF файла #1, используя компонент SimpleConnect.
- В SWF файле #1, пользователь выбирает один из экземпляров приложений компонента RoomList и щелкает кнопку Create Room или Join Room.

Компонент RoomList открывает новое окно браузера, содержащее другую страницу HTML с другим Flash move (SWF файл #2). Код HTML этого документа передает параметры от оригинального Flash move (SWF файл #1); SWF файл #2 соединяется с новым FlashCom application (приложение B), которое получает параметры. Серверный скрипт должен быть написан для приложения B, чтобы обработать параметры и зарегистрировать приложение A, чтобы пользователь теперь мог соединиться с приложением B.

Смотрите документацию Developing Communication Applications, которая сопровождает Flash Communication Server MX 1.5 (доступна в двух форматах: LiveDocs и PDF на http://www.macromedia.com/support/documentation/en/flashcom/index.html) для получения большей информации об обработчике событий на стороне сервера, который необходим для двух FlashCom applications, использующих компонент RoomList.

Вложенные компоненты Flash UI

Компонент PresentationText использует три вложенных Flash UI компонента: ListBox, PushButton и ScrollBar. Вы можете изменить внешний вид этих компонентов, изменяя соответствующие оболочки символов в папке Component Skins на панели Library Flash-документа.

Параметры компонента со стороны клиента

Экземпляр компонента PresentationText имеет только одну настройку на панели Properties или панели Component Parameters.

Room Application Path

Используя предыдущий пример как руководство к действию, этот параметр определяет URL документа HTML, содержащего второе Flash move (SWF файл #2), который соединяется с FlashCom приложением B. Этот документ HTML получает два параметра от Flash move, возглавляемого компонентом RoomList: имя пользователя и путь к экземпляру приложения, соединяющегося с приложением A. HTML документ должен использовать либо скрипт на стороне клиента, такой как JavaScript или VBScript, либо скрипт на стороне сервера, такой как ColdFusion, чтобы обработать переданные параметры в URL строку и передать их в теги <OBJECT> и <EMBED> второго Flash move, подсоединенного к приложению B.

Обзор действий компонента

Как упомянуто ранее, компонент RoomList разработан, чтобы работать в сценарии двух FlashCom приложений, с отдельными HTML и SWF файлами для каждого приложения.

Текст, приведенный ниже, предназначен, чтобы быть техническим кратким обзором требований и шагов, необходимых при использовании компонента RoomList.

Смотрите Developing Communication Applications для пробного осуществления использования компонента RoomList:

Папка приложения (приложения A) создается FlashCom сервером для Flash move отображающего компонент RoomList. Это приложение отслеживает все экземпляры приложений, создающиеся пользователями, и соединяющимися со вторым приложением (приложение B), управляемым компонентом RoomList.

Другая папка приложения (приложение B) создается на FlashCom сервере. Это приложение соединится с приложением A через метод Application.call() стороны сервера, чтобы обновить информацию приложения A, которая отображена в компоненте RoomList.

Оба приложения загружают код components.asc стороны сервера при запуске, для доступа и хранения клиента и переменных экземпляра компонента.

Flash документ (файл .fla), который мы используем, чтобы создать SWF #1, создан с использованием экземпляра компонента SimpleConnect и экземпляра компонента RoomList. Параметр Application Directory компонента SimpleConnect использует rtmp:// URI приложения A, обсуждаемого в ранее. Название экземпляра компонента RoomList определено в параметре списка значения Communication Components компонента SimpleConnect (который будет обсуждаться позже). Параметр Room Application Path компонента RoomList устанавливает URL документа HTML управляемого Flash movie, соединяющимся с приложением B. Отметьте, что этот URL - http: // URL, а не rtmp: // URI. Эти Flash и HTML документы опубликованы и лежат на Вашем web сервере.

Flash документ (.fla файл), который будет использоваться, чтобы создать SWF #2, сделан для второго приложения (приложение B). Этот документ содержит управляемое приложение, проверенное компонентом RoomList и приложением A. Flash move (.swf) для этого документа вложен в документ HTML, определенный в параметре Room Application Path компонента RoomList. Например, Вы можете создать Flash документ, который использует компоненты SimpleConnect, Chat и PeopleList для основной комнаты для дискуссий. Этот документ должен обработать параметры, которые передает компонент RoomList в URL документа HTML, управляемого Flash move (SWF #2). Если Вы будете использовать компонент SimpleConnect в SWF #2, который соединяется с приложением B, то параметры будут обработаны автоматически. Иначе, Вы будете должны использовать клиентский ActionScript, чтобы обработать две переменные _root, username и appInstance, которые передает компонент RoomList (username - значение строки, введенное пользователем в компонент SimpleConnect SWF #1, appInstance – название "участка памяти" экземпляра, определенного в диалоговом окне Create Room или существующее названии случая, отображенное в компоненте RoomList).

Наконец, метод application.onConnect() серверного кода ActionScript для приложения B, должен передать значения username и appInstance коду ActionScript на стороне сервера приложения A. Там RSO, используемое компонентом RoomList на стороне сервера, хранит, обновляет и транслирует новую информацию. В результате, каждый пользователь, подключенный к SWF #1, будет видеть обновленное количество пользователей и новый экземпляр приложения, отображающийся в пользовательском интерфейсе компонента RoomList.

SetBandwidth

Компонент SetBandwidth позволяет пользователю контролировать, сколько пропускной способности канала может быть использовано объектами Camera и Microphone, которые используются другими компонентами соединения, такими как компоненты AVPresence и AudioConference. Компонент SetBandwidth ограничивает количестово байтов, которое может передать клиентский Flash move на FlashCom application. Пользовательский интерфейс компонента содержит список из которого можно выбрать четыре типа подключения: Modem, DSL, LAN и Customas как показано на рисунке. От выбора типа подключения зависит предварительно установленная пропускная

способность (типичные скорости подключения), которую можно установить на панели Properties или в клиентском ActionScript.

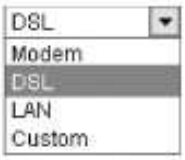

Рисунок 19. Меню компонента SetBandwidth

Если Вы выберите Customas, то откроется отдельное окно, как показано на рисунке ниже, в котором пользователь может самостоятельно выбрать максимальное и минимальное значения пропускной способности.

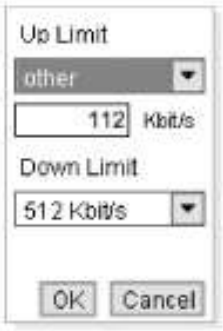

Рисунок 20. Опции Custom для компонента SetBandwidth

Всякий раз, когда пользователь выбирает элемент в компоненте SetBandwidth, качество и атрибуты объектов Camera и Microphone, управляемых другими компонентами соединения изменятся.

Вложенные компоненты Flash UI

Компонент PresentationText использует три вложенных Flash UI компонента: ComboBox, PushButton и ScrollBar. Вы можете изменить внешний вид этих компонентов, изменяя соответствующие оболочки символов в папке Component Skins на панели Library Flash-документа.

Параметры компонента со стороны клиента

Экземпляр компонента SetBandwidth имеет шесть параметров настройки в панели Properties или панели Component Parameters, как перечислено в Таблице 3.

Эти параметры устанавливают приоритетный предел пропускной способности для входящего и выходящего трафика, когда пользователь выбирает указанный тип подключения (Modem, DSL или LAN) в компоненте SetBandwidth. Тип подключения также затрагивает capture атрибуты объекта Camera (используется компонентом AVPresence) и sampling rate объекта Microphone (используется компонентами AVPresence или AudioConference).

Необходимо отметить, что:

Практическая пропускная способность обычных модемов 56 Кб/с, только 33 Кб/с. Опции DSL отражают верхний предел домашней ADSL (асимметричная DSL) подключения. По умолчанию значение Down для DSL больше чем значение Up; поэтому, качество поступающих потоков для Flash move может быть почти в два раза больше, чем качество исходящего потока. Значение по умолчанию для LAN - 10 000 Кб/с, оно соответствует типичным 10Мб сетям. Вы можете изменить значения, увеличивая или уменьшая пропускную способность. Например, если Вы предпочли бы, чтобы нормы для

DSL были выше, Вы могли бы использовать значение 512 Кб/с и для значения Up и для значения Down.

Таблица 3. SetBandwidth значения по умолчанию для каждого параметра

Название параметра	Значение по умолчанию	Установки камеры	Частота микрофона	Свойство
Modem Up	33 Кб/с	120 x 90 область сбора данных, 6 fps, 60 качество, интервал ключевых кадров 12	5 кГц	modemUp
Modem Down	33 Кб/с	Так же как Modem Up	5 кГц	modemDown
DSL Up	128 Кб/с	160 x 120 область сбора данных, 12 fps, 80 качество, интервал ключевых кадров 7	11 кГц	dslUp
DSL Down	384 Кб/с	Так же как DSL Up	11 кГц	dslDown
LAN Up	10,000 Кб/с	320 x 240 область сбора данных, 15 fps, 90 качество, интервал ключевых кадров 5	22 кГц	lanUp
LAN Down	10,000 Кб/с	Так же как LAN Up	22 кГц	lanDown

Пределы, установленные компонентом SetBandwidth, вторичны по сравнению с пределам практической скорости доступа в Интернет. Например, если пользователь выберет LAN, но будет передавать данные по подключению DSL, то исходящий поток будет передан только в том качестве, которое продиктовано фактическим подключением. Хотя пользователь это не сильно заметит. Однако, если пользователь выберет LAN несмотря на то, что он использует подключение по модему, то пропускная способность не будет поддерживать увеличенную область оцифровки видеоизображений и скорость передачи кадров. В том случае, FlashCom обрежет видео кадры, заставляя изображение заметно запинаться. Звуковой поток будет использовать частоту 22 кГц, но это слишком большое значение для модемной связи. Поэтому, аудио будет также заикаться. В результате, аудио и видео потоки будут довольно непонятными.

Обзор действий компонента

Компонент SetBandwidth выполняет две функции в Flash move. Он создает экземпляр Notifier в объекте gFlashCom, который изменяет пропускную способность других компонентов. Объект gFlashCom - просто объект, созданный в пространстве имен _global с целью совместно использовать параметры, такие как пропускная способность между компонентами соединения на стороне клиента. Например, когда Вы добавляете компонент AVPresence или компонент AudioConference, каждый из этих экземпляров добавляется к экземпляру Notifier. Во время выполнения, когда пользователь выбирает опцию в компоненте SetBandwidth, все эти компоненты получат обновления, которые требуют изменения атрибутов объекта Камеры и/или Микрофона.

Компонент SetBandwidth также отправляет эти измененные настройки на FlashCom. Когда компонент FCSetBandwidth на стороне сервера получает обновления, он вызывает Client.setBandwidthLimit(), чтобы ограничить пропускную способность для потоков, управляемых сервером. Заметьте, что этот метод управляет и пропускной способностью от сервера к клиенту и пропускной способностью от клиента на сервер. Изменения сервера отменяют любые параметры настройки, установленные кодом ActionScript на стороне клиента.

SimpleConnect

Компонент SimpleConnect устанавливает основное подключение Flash move к FlashCom application. Другие компоненты коммуникации часто используют подключение, установленное компонентом SimpleConnect, чтобы обратиться к FlashCom application. Название этого компонента говорит о том, что его очень просто использовать для создания соединения. Во время выполнения, этот компонент отображает поле имени пользователя и кнопку Login, как показано на рисунке 21.

Рисунок 21. Интерфейс компонента SimpleConnect

Пользователь вводит свое имя в текстовое поле и нажимает кнопку Login. Когда компонент удачно установит соединение с FlashCom application, он сообщает это объекту NetConnection других компонентов соединения (или Ваших собственных объектов).

Вложенные компоненты Flash UI

Компонент SimpleConnect использует вложенный Flash UI компонент PushButton. Вы можете изменить внешний вид этого компонента, изменяя соответствующие оболочки символов в папке Component Skins на панели Library Flash-документа.

Параметры компонента со стороны клиента

Экземпляр компонента SimpleConnect имеет две настройки на панели Properties или панели Component Parameters:

Application Directory

Путь к FlashCom Server и приложению, к которому подключается компонент. Это значение является RTMP URI, начинающегося со строки "rtmp:". Вы можете использовать относительный RTMP URI, который опускает имя домена сервера, если FlashCom Server находится на том же самом IP адресе или домене как и ваш сервер сети. Например, rtmp:/chat говорит компоненту SimpleConnect соединяться с приложением чата на FlashCom Server. Для готовых приложений, тем не менее, Вы должны использовать полный RTMP URI, такой как rtmp://fcs.mydomain.com/app, где fcs.mydomain.com - это имя сервера, где запущен FlashCom, а app - это имя папки приложения. Этот параметр соответствует свойству appDirectory.

Communication Components

Список других экземпляров компонентов соединения. Если Вы дважды щелкните на этом параметре в панели Properties, Вы сможете ввести множество имен экземпляров компонентов соединения, которые сосуществуют на сцене вместе с компонентом SimpleConnect. Когда экземпляр SimpleConnect соединяется с FlashCom application, он использует это подключение совместно с объектами, определенными в списке Communication Components. Вы можете непосредственно управлять этим списком, используя свойство fcComponents (является массивом).

Вы можете создать ваши собственные объекты, которые могут быть распознаны компонентом SimpleConnect. Просто создайте метод connect() для вашего объекта, и определите название объекта в списке Communication Components. Ваш объект должен быть обработан на временной дорожке экземпляра SimpleConnect.

Как работает компонент

Компонент SimpleConnect подсоединяет Flash move к FlashCom application, вызывая метод NetConnection.connect() на стороне сервера, как только экземпляр загружается в ролик. Компонент SimpleConnect фактически пробует соединиться и со стандартным RTMP и с туннелированным подключением RTMPT. Какой бы тип подключения ни отвечал, сначала становится активными URI соединения. Затем, подключение разделяется между объектами, определенными в параметре Communication Components (свойство fcComponents) экземпляра SimpleConnect.

Когда пользователь вводит имя и щелкает кнопку Login (или нажимает клавишу Enter), компонент SimpleConnect выпускает метод NetConnection.call() к методу FCSimpleConnect.changeName() на стороне сервера. Этот метод отслеживает имя пользователя, введенное в текстовое поле компонента SimpleConnect, и делит эти данные между другими компонентами соединения.

Поскольку компонент SimpleConnect не поддерживает такие особенности, как ввод пароля в дополнение к имени пользователя, Вы можете захотеть создать подключение вручную как часть входа в систему. Ручной подход требует, чтобы Вы зарегистрировали другие компоненты соединения.

UserColor

Компонент UserColor - элемент пользовательского интерфейса, который позволяет каждому пользователю выбирать цвет, чтобы выделить себя в компонентах Chat и Cursor. Интерфейс компонента UserColor представляет собой combo box с 10 предлагаемыми цветами, как показано на рисунке 22. Когда пользоваель выбирает цвет в компоненте UserColor, его текст в окне компонента Chat окрашивается в этот цвет. Если в Flash move используется компонент Cursor, пользовательский курсор будет так же отображаться этим же выбранным цветом.

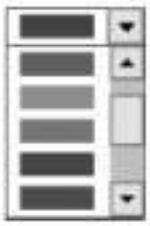

Рисунок 22. Компонент UserColor

Вложенные компоненты Flash UI

Компонент UserColor использует вложенные Flash UI компоненты ComboBox и ScrollBar. Вы можете изменить внешний вид этих компонентов, изменяя соответствующие оболочки символов в папке Component Skins на панели Library Flash-документа.

Как работает компонент

Компонент UserColor, как и компонент SetBandwidth, использует объект gFlashCom.Notifier в клиентском ActionScript, чтобы зарегистрировать изменения цвета, сделанные пользователем, в других компонентах соединения. Когда пользователь загружает Flash move, который содержит компонент UserColor, этот компонент соединяется с FlashCom application, с помощью компонента SimpleConnect или пользовательского соединения. Для нормальной работы компоненту UserColor требуется, чтобы был вызван метод setUsername() и ему было передано значение строки с именем пользователя. Если Вы используете компонент SimpleConnect, это случается автоматически. Компонент UserColor также хранит значение цвета, выбранного

пользователем, в свойстве usercolor в памяти объекта в пределах ActionScript на стороне сервера в течение сеанса приложения.

VideoConference

Компонент VideoConference позволяет Вам создать Flash client, способного к показу нескольких экземпляров компонента AVPresence. Поскольку каждый пользователь соединяется с FlashCom application, новый экземпляр AVPresence добавленный на сцену, отображает ее аудио/видео поток как показано на рисунке 23.

Рисунок 23. Исполнение компонента VideoConference

Этот компонент может быть очень требователен к пропускной способности и для клиента и для сервера. Поскольку каждый пользователь добавляемый к AV чату, мало того, что выдает свой поток должен связываться с дополнительными потоками, в зависимости от числа пользователей. Каждый новый пользователь имеет исходящий поток, плюс один поступающий поток. Аналогично, поток нового пользователя нужно послать существующим пользователям (в дополнение к потокам, которые они уже имеют). Например, если Вы имеете трех пользователей, связанных с приложением, используя компонент VideoConference, сервер управляет девятью потоками; сервер и пропускная способность каждого пользователя должны быть способными к обрабатыванию трех потоков для каждого пользователя: его собственный поток (выход) и два от других участников (вход). Вы можете быстро определить, сколько потоков сервер должен обработать – это значение равно квадрату числа участников. Например, пять участников будут требовать, чтобы сервер обработал 25 потоков (пять для каждого участника).

Вложенные компоненты Flash UI

Компонент VideoConference не имеет ни одного вложенного Flash UI компонента. Однако включает в себя символы компонента AVPresence. Оболочки компонента VideoConference находятся в паках Communication Components → Core Assets - Developer Only → VideoConference Assets библиотеки Вашего документа. Вы можете так же изменить внешний вид компонента AVPresence изменяя графические символы в папке Assets → fc_av_icons.

Параметры компонента со стороны клиента

Компонент VideoConference имеет четыре параметра, которые могут быть настроены на панели Properties или панели Component Parameters:

Show Boundary

Это параметр может принимать только булевы значения. Он соответствует свойству showBoundary и определяет, отображена ли граница экземпляра VideoConference. Значение по умолчанию - false.

Show Background

Это параметр может принимать только булевы значения. Он соответствует свойству showBackground, управляет цветом заливки фона в компоненте VideoConference. Фоновый графический символ может быть изменен в символе VideoConfBackground папки Assets VideoConference на панели Library. По умолчанию, это значение - false.

Clip Mask

Это параметр может принимать только булевы значения. Он соответствует свойству clipMask и управляет возможностью перетаскивания экземпляра AVPresence за границу области объекта VideoConference. Если этот параметр true (значение по умолчанию), экземпляры AVPresence появляются только в пределах граничной области экземпляра VideoConference. Иначе, экземпляры AVPresence появляются также вне границы области экземпляра VideoConference.

Drag Sharing

Этот параметр соответствует свойству dragSharing. Он контролирует перемещение экземпляра AVPresence на новое место сцены. Если установлено значение true (по умолчанию), то, когда один пользователь перемещает экземпляр, все пользователи, видит его в новой позиции. Если установлено значение false, каждый пользователь может изменять позиции экземпляра, не затрагивая позиции тех же самых экземпляров в роликах других пользователей. Этот параметр устанавливается пользователем; если есть три пользователя (A, B, и C) и пользователи A и B установят Drag Sharing равное true,а пользователь C установит значение false, то пользователи A и B будут видеть изменения друг друга. Тем временем, пользователь C может свободно перетащить экземпляры AV в пределах своего приложения без тех изменений, передаваемых от пользователей A и B.

Обзор действий компонента

Когда метод connect() экземпляра VideoConference вызывается экземпляром SimpleConnect или пользовательским объектом NetConnection, экземпляр VideoConference вызывает метод NetConnection.call() с серверного метода FCVideoConference.connect(), как определено в коде videoconference.asc (расположен в папке scriptlib/components). Метод connect() на стороне сервера создает уникальный ID для Flash client и хранит ID в RSO users. Метод connect() на стороне клиента также вызывает метод VideoConference.setID(). Этот метод приводит все осатльное в движение, где компонент VideoConference прикрепляет экземпляр Video Window (чьим ID является FCVideoWindowSymbol) к сцене. Этот символ в свою очередь прикрепляет экземпляр компонента AVPresence к его временной оси. В любое время, когда пользователь соединяется с FlashCom application, ID пользователя, добавляется к users RSO, вызывая обработчик событий onSync(), обновляющий сцену с новым экземпляром Video Window для нового пользователя.

VideoPlayback

Компонент VideoPlayback обеспечивает пользовательский интерфейс, для воспроизведения зарегистрированных потоков, которые постоянно находятся в FlashCom application. Вы можете воспроизвести потоки, записанные FlashCom с пользовательской web камеры, или Вы можете запустить .flv файлы, которые Вы создали с помощью инструмента Macromedia's FLV Exporter или сторонней утилиты, такой как Sorenson

Squeeze or Wildform Flix. Компонент VideoPlayback включает в себя строку управления, как показано на рисунке 24. Вы можете запускать, останавливать и искать потоки, а также управлять их громкостью. Края видео замаскированы округленными углами.

Рисунок 24. Исполнение компонента VideoPlayback

Вложенные компоненты Flash UI

Компонент VideoPlayback не имеет ни одного вложенного Flash UI компонента. Если Вы желаете изменить внешний вид этого компонента, Вы можете изменить графические символы, находящиеся в папке Communication Components → Core Assets - Developer Only → VideoPlayback Assets библиотеки Вашего Flash документа.

Параметры компонента со стороны клиента

Компонент VideoPlayback имеет два параметра, которые могут быть настроены на панели Properties или панели Component Parameters:

Default Stream Name

Этот параметр, соответствуя свойству streamName, определяет название потока, чтобы проигрывать его с FlashCom application. Вам не нужно включать .flv расширение в название потока. Например, если Вы хотите запустить файл, названный recording.flv, который сохранен в папке приложения streams/_definst_, определите recording этого поля в панели Properties. Если Вы создали виртуальный каталог, чтобы хранить потоки на вашем FlashCom Server, Вы можете определить название псевдонима перед названием потока. Например, если Вы создали псевдоним, названный common, Вы можете запустить файл, названный recording.flv расположенный там, определяя common/recording.

Buffer Time

Этот параметр, соответствуя bufferTime свойству, управляет тем, сколько из потока будет буферизировано, мгновенно, прежде чем воспроизведение начинается. Значение по умолчанию - 2 секунды. Большее значение увеличит время ожидания пользователей, прежде чем воспроизведение начнется, но целиком работа воспроизведения может быть улучшена, потому что есть меньше шансов, что видео, сделает паузу из-за перегрузки сети.

Обзор действий компонента

Когда экземпляр VideoPlayback инициализируется в Flash move и его метод connect() вызывается экземпляром SimpleConnect или пользовательским объектом

NetConnection, который Вы создали, экземпляр VideoPlayback создает новый экземпляр FCPlayStreamClass, класс, который определен в символе FCPlayStream_Class компонента. Этот класс наследует от класса NetStream и увеличивает его функциональные возможности, создавая методы и свойства, которые позволяют потокам работать с интерфейсом строки управления компонента VideoPlayback.

VideoRecord

Компонент VideoRecord может публиковать и записывать аудио/видео поток от Flash move до FlashCom application. Пользовательский интерфейс компонента VideoRecord, который показан на рисунке 25, подобен интерфейсу компонента VideoPlayback. Когда компонент экземпляра загружается, выход с Вашей камеры автоматически отображается в видео окне. Крайняя левая кнопка в интерфейсе управления – переключатель записи начало/остановка. Кнопка с правой стороны - переключатель игра/останова, который позволяет Вам просматривать ваш поток. С правой стороны от него - дисплей состояния буферизации. В то время как пользователь делает запись потока, область буферизации пуста. Когда пользователь просматривает запись, область буферизации отображает область продвижения, указывающую, сколько из буфера потока было заполнено. Индикатор состояния отчет/воспроизведение показывает справа от дисплея буферизации. Когда идет запись, индикатор состояния – мигающая красная точка. Когда идет просмотр, индикатор состояния - мигающая синяя точка.

Рисунок 25. Исполнение компонента VideoRecord

Вы можете использовать компонент VideoPlayback вместе с компонентом VideoRecord. Компонент VideoPlayback предлагает больше управления воспроизведением потока, чем кнопка play/stop в компоненте VideoRecord. После того, как Вы сделали запись потока с помощью компонента VideoRecord, Вы можете использовать экземпляр компонента VideoPlayback, чтобы запустить поток.

Вложенные компоненты Flash UI

Компонент VideoPlayback не использует вложенных компонентов Flash UI. Вы можете найти оболочка для символов компонента в папке Communication Components → Core Assets - Developer Only → VideoRecord Assets библиотеки Вашего документа.

Параметры компонента со стороны клиента

Компонент VideoRecord имеет шесть настроек на панели Properties или панели Component Parameters:

Default Stream Name

Этот параметр, соответствуя свойству streamName, определяет название потока, который зарегистрирован в FlashCom application. Вы можете использовать те же самые соглашения, описанные ранее для параметра Default Stream Name компонента VideoPlayback.

Default Settings

Этот параметр контролирует настройки степени качества, которые установлены у объектов Camera и Microphone во время записи. По умолчанию установлено значение setHigh, которое ссылается на объект, определенный параметром High Quality. Вы можете создать свой собственный объект качества и использовать его как значение defaultSetting параметра экземпляра VideoRecord. В качестве альтернативы, Вы можете передать этот объект VideoRecord.adjustSettings().

Buffer Time

Это параметр соответствует свойству bufferTime и устанавливает количество времени, в секундах, которое записываемый поток и поток воспроизведения используют для буферизации. Значение по умолчанию – 10 секунд.

High Quality Settings, Medium Quality Setting, и Low Quality Settings

Этот параметр используется, чтобы приспособить качество выходных данных с объектов Camera и Microphone к записываемому потоку. Эти параметры управляют значениями настройки по умолчанию для низкой, средней и высокой скорости передачи данных Интернет соединения. (Используемые параметры настройки зависят от значения параметра Default Settings или свойства defaultSetting.) Каждая настройка использует объект со следующими свойствами (для объекта Camera): width (в пикселях), height (в пикселях), колдичество кадров в секунду (fps), bandwidth (в Кб/с) и key (для keyframe интервала – интервала ключевых кадров). Объект также имеет свойство rate, которое управляет частотой объекта Microphone. Вы можете аннулировать эти параметры настройки в клиентском ActionScript, указав новые значения свойств setLow, setMed и setHigh компонента VideoRecord.

Компонент SetBandwidth не применяет никаких изменений к компоненту VideoRecord. Вы можете, однако, использовать экземпляр ComboBox, чтобы позволить пользователю выбирать, какая качественная настройка должна быть применена к экземпляру VideoRecord.

Обзор действий компонента

Когда метод connect() экземпляра VideoRecord вызван и он передал ссылку объекту NetConnection, компонент создает два потока: выходной поток (out_ns) для записи и входной поток (in_ns) для воспроизведения. В отличие от компонента VideoPlayback, компонент VideoRecord не использует FCPlayStreamClass, чтобы создать пользовательский класс NetStream. Компонент использует существующие методы класса NetStream. Компонент также вызывает метод подключения cameraOn(), который захватывает выход объектов Camera и Microphone. Метод CameraOn() вызывает метод adjustSettings(), чтобы управлять качеством вывода, установленным для объектов Camera and Microphone. Вы можете вызвать VideoRecord.adjustSettings() Ваших собственных

элементов пользовательского интерфейса, типа поля со списком, изменять качество записи.

Whiteboard

Компонент Whiteboard, как видно из названия, показывает панель пользовательского интерфейса с рисунками и текстовыми подписями инструментов рисования, как показано на рисунке 26. Меню имеет следующие опции (сверху вниз):

Move/Transform tool

Осуществляет выделение и перемещение на панели инструментов. Вы можете удалить иконку, щелкнув на ней инструментом Move/Transform с нажатой клавишей Delete. Вы можете использовать этот инструмент, чтобы изменить размеры линий стрелки, но Вы не можете изменить размеры текста или текстовых полей с этим инструментом.

Text tool

Создает стандартный текст, шрифтом a _serif.

Text Box tool

Перетаскивание контура поля, используя цвет, выбранный в меню цвета. Текст в этом поле использует шрифт _sans.

End Arrow Line tool

Чертит линию оканчивающуюся стрелкой (конец определяется местоположением второго щелчка).

Start Arrow Line tool

Работает так же как End Arrow Line tool, за исключением того, что стрелка появляется в начале линии (на месте первого щелчка).

Dual Arrow Line tool

Создает линию со стрелками на обоих концах.

Color menu

Восемь цветов, которые Вы можете использовать как цвета заливки для инструмента Текста или Текстового поля или как цвета линии для инструментальных средств линии.

Expand/Collapse button

Разворачивает и сворачивает пользовательский интерфейс панели инструментов.

Рисунок 26. Интерфейс пользовательского меню компонента Whiteboard

Если Вы ищете полный компонент whiteboard, закажите Fig Leaf Software's WYSIdraw на http://products.figleaf.com. Этот компонент whiteboard дает больше средств рисования и может работать с изображениями JPEG и другими художественными работами в совместной среде. Для предприятия, цена этого компонента начинается с $995.

Вложенные компоненты Flash UI

Компонент whiteboard не использует вложенных компонентов Flash UI. Вы можете корректировать внешний вид компонента, регулируя оболочки его символов в папке Communication Components → Core Assets - Developer Only → VideoRecord Assets библиотеки Вашего документа.

Как работает компонент

Когда метод connect() компонента whiteboard вызывается экземпляром SimpleConnect или пользовательским объектом NetConnection, компонент создает RSO whiteboard внутри FlashCom application. Когда пользователь перетаскивает одно из инструментальных средств компонента, параметры пользовательского объекта посылаются общедоступному объекту, и, в свою очередь, обновление передано всем пользователям. Обработчик OnSync() общедоступного объекта, удаляет форму с помощью MovieClip.removeMovieClip() или вызывает Whiteboard.drawShape(), чтобы создать новый текст, текстовое поле или объект линию на Сцене.

Создание приложения, отслеживающего соединение

Сейчас, когда Вы уже имеете общее представление о компонентах соединения, давайте посмотрим, как использовать их для создания приложения. Мы создадим Flash movie, который подключается к FlashCom application, используя компоненты SimpleConnect и ConnectionLight. Компонент SimpleConnect определит URL FlashCom application, а экземпляр ConnectionLight будет показывать статистику соединения.

Перед началом выполнения этого упрежнения, убедитесь, что у Вас установлены последние компоненты соединения для Flash MX или Flash MX 2004, как сказано ранее.

Создание папки FlashCom Application и серверного приложения

Во-первых, установите приложение basic_app, следующим образом:

На компьютере, где запущен FlashCom Server, откройте папку applications. Здесь создайте новую папку и назовите ее basic_app.

Во Flash Pro или любом другом текстовом редакторе, создайте новый документ ActionScript Communication. Добавьте следующий код в этот документ:

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

Эта строчка кода дает инструкцию FlashCom application загрузить документ components.asc из папки scriptlib, который в свою очередь загружает .asc файлы для каждого компонента соединения.

Сохраните текстовый документ как main.asc в папке basic_app, созданной Вами в шаге первом.

Ваше FlashCom application сейчас готово к использованию компонентов соединения.

Создание клиентского Flash приложения

Сейчас, когда серверный код .asc готов, Вы должны создать клиентское приложение (.swf)

Создайте новый Flash-документ (File→New). Сохраните его как component_connection.fla.

Переименуйте Layer 1 в connect_mc. Этот слой будет использоваться для хранения экземпляра компонента SimpleConnect.

Откройте панель Components (Ctrl-F7 или Cmd-F7), и откройте набор компонентов соединения. Перетащите экземпляр компонента SimpleConnect в первый кадр слоя connect_mc. Разместите экземпляр около верхнего левого угла сцены.

Выберите новый экземпляр и, используя панель Properties, задайте ему имя connect_mc. В таблице параметров панели Properties введите в поле Application Directory путь к basic_app FlashCom application. Например, если Вы используете такую же версию FlashCom, как и Ваш web-серевер, введите rtmp:/basic_app. Иначе, определите полный RTMP путь к Вашему FlashCom Server и приложению, например, rtmp://fcs.mydomain.com/basic_app.

Создайте новый слой на панели Timeline и назовите его light_mc. В первый кадр этого слоя перетащите экземпляр компонента ConnectionLight из панели Components. Разместите этот экземпляр справа от экземпляра connect_mc. В панели Properties назовите этот экземпляр light_mc и оставьте значения по умолчанию для параметров компонента.

Выберите экземпляр connect_mc и на панели Properties щелкните поле Communication Components. В списке зничений щелкните на кнопку «+» и введите light_mc в поле, отмеченное нулем, как показано на рисунке 27. Щелкните OK и закройте диалоговое окно. Определяя имя экземпляра light_mc, экземпляр SimpleConnect регистрирует его, подключая к приложению basic_app.

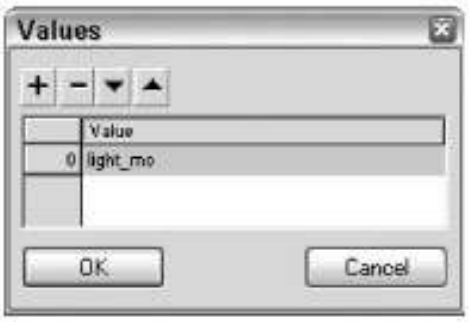

Рисунок 27. Список значений для параметра Communication Components компонента SimpleConnect

Сохраните Ваш документ и протестируйте ролик (Control→Test Movie). Как только Flash movie соединяется с basic_app application, экземпляр ConnectionLight станет зеленым. Если соединение не удалось, индикатор будет красным. Если достигнут порог времени ожидания подключения, индикатор будет желтым.

Создание простого чата

Теперь, когда Вы уже знаете, как подключать FlashCom application, используя компоненты SimpleConnect и ConnectionLight, Вы можете продолжить добавление возможностей Вашему приложению. В следующем шаге, Вы научитесь добавлять компоненты PeopleList, Chat, и UserColor в приложение basic_app:

Откройте component_connection.fla, созданный в прошлом разделе. Пересохраните этот документ как component_chat.fla.

Создайте новый слой и назовите его list_mc. В кадр 1 этого слоя перетащите экземпляр компонента PeopleList из панели компонентов. Используя панель свойств, назовите этот экземпляр list_mc. Разместите этот компонент около нижней левой кромки сцены так, как показано на рисунке 28.

Рисунок 28. Экземпляр list_mc

Создайте еще один слой и назовите его chat_mc. В кадр 1 этого слоя перетащите экземпляр компонента Chat. Используя панель свойств, назовите этот экземпляр chat_mc. Разместите этот компонент справа от экземпляра list_mc, как показано на рисунке 29.

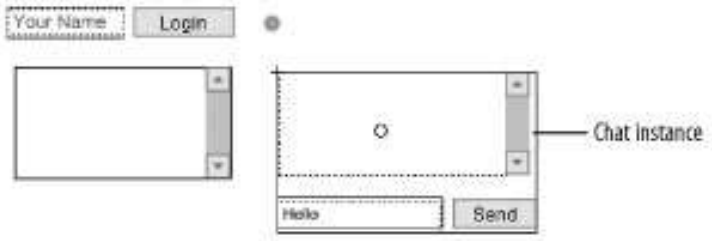

Рисунок 29. Экземпляр chat_mc

Создайте еще один слой и назовите его userColor_mc. В кадр 1 этого слоя перетащите экземпляр компонента UserColor. Используя панель свойств, назовите этот экземпляр userColor_mc. Разместите этот экземпляр ниже экземпляров SimpleConnect и ConnectionLight около верхнего левого края сцены, как показано на рисунке 30.

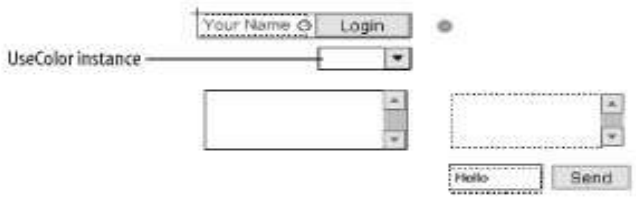

Рисунок 30. Экземпляр userColor_mc

Выберите экземпляр connect_mc на сцене и два раза щелкните на параметре компонента соединения в таблице параметров на панели свойств. В список значений добавьте имена экземпляров трех добавленных компонентов соединения list_mc, chat_mc и userColor_mc, как показано на рисунке 31.

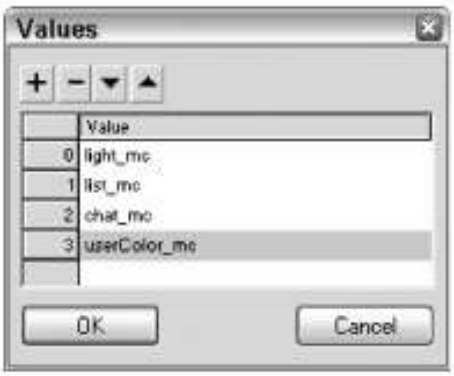

Рисунок 31. Обновление списка значений параметров компонентов соединения для компонента SimpleConnect

Сохраните Ваш Flash-документ. Чтобы видеть, что элементы чата работают должным образом, выберите File→ Publish Preview→Default - (HTML). Оставьте окно браузера открытым и вернитесь к параметрам среды Flash. Здесь, выберите Control→Test Movie и откройте еще одну версию Flash movie. Когда ролик загрузится, введите Ваше имя в поле username компонента SimpleConnect и щелкните на кнопке Login. Ваше имя должно появиться внутри компонента PeopleList и в среде Test Movie и в окне браузера. Теперь загрузитесь с другим именем пользователя в запущенный Flash movie из окна браузера. Снова Вы должны увидеть новое имя появляется в обоих экземплярах PeopleList. В каждом ролике выберите уникальный цвет для компонента UserColor. Наконец, введите сообщение в окно сообщений чата и щелкните на кнопку Send. Вы должны увидеть, что сообщение появляется в области истории в обоих экземплярах Chat. Цвет текста сообщения должен быть таким, который Вы выбрали в экземпляре отправителя.

Добавление аудио и видео к чату

Чтобы добавить аудио и видео возможности для чата, Вы можете добавить AudioConference, AVPresence или VideoConference компоненты в Flash-документ, созданный в прошлом разделе. В следующем шаге, мы добавим экземпляр компонента AVPresence и экземпляр компонента SetBandwidth к ролику:

Откройте component_chat.fla из предыдущей секции. Пересохраните этот документ как component_avchat.fla.

Создайте два новых слоя и назовите их av_1_mc и av_2_mc.

В первый кадр слоя av_1_mc перетащите экземпляр компонента AVPresence из панели компонентов. Поместите этот экземпляр слева от экземпляра ConnectionLight и сверху экземпляра Chat. На панели свойств назовите этот экземпляр av_1_mc.

Повторите предыдущий шаг для другого экземпляра компонента AVPresence. Поместите новый экземпляр в кадре 1 слоя av_2_mc, разместите его справа от экземпляра av_1_mc. Назовите новый экземпляр av_2_mc.

Создайте новый слой bandwidth_mc. В первый кадр этого слоя перетащите экземпляр компонента SetBandwidth. Разместите этот экземпляр ниже экземпляра userColor_mc. Используя панель свойств, назовите экземпляр bandwidth_mc. Ваша сцена сейчас должна выглядеть, как показано на рисунке 32.

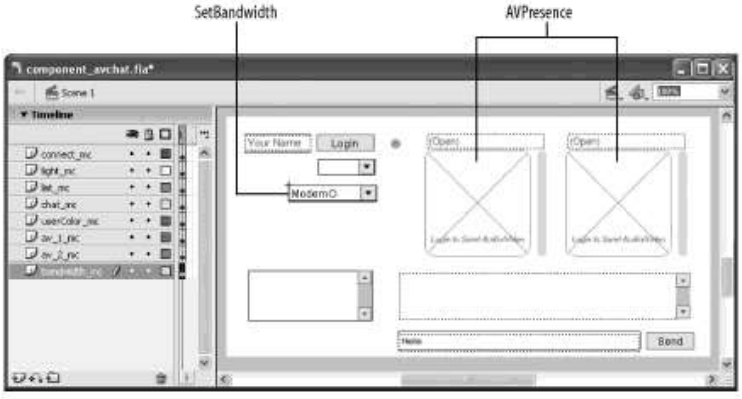

Рисунок 32. Новые компоненты соединения добавлены к сцене ролика

Выберите экземпляр connect_mc на сцене и два раза щелкните на параметре компонента соединения в таблице параметров на панели свойств. В список значений добавьте имена экземпляров трех добавленных компонентов соединения av_1_mc, av_2_mc, bandwidth_mc как показано на рисунке 33.

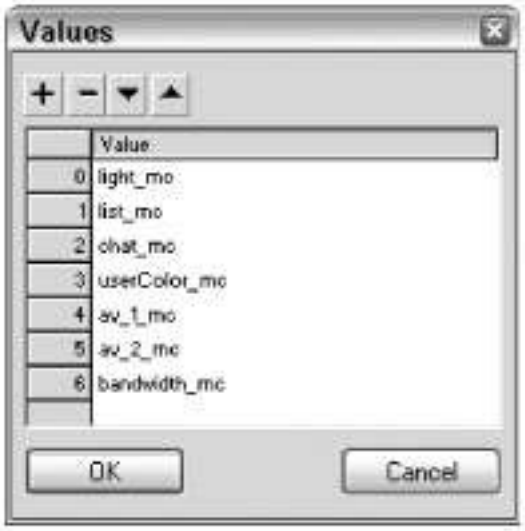

Рисунок 33. Обновление списка значений параметров компонентов соединения для компонента SimpleConnect

Сохраните Ваш документ и протестируйте его командами Publish Preview и Test Movie. Если у Вас есть web-камера или микрофон, подключенные к компьютеру, начните воспроизведение аудио/видео из экземпляра AVPresence Вашего Flash movie. Затем Вы должны видеть тот же самый аудио/видео поток, показанный другим экземпляром AVPresence. (Будьте осторожны и носите наушники в течение этого теста. Если Вы будете использовать открытый микрофон и регулярные колонки, Вы, вероятно, почувствуете резкую звуковую обратную связь.) Тестируя различные настройки bit rate, в компоненте SetBandwidth Вы увидите качественные изменения на аудио и видео выходах.

Отказ от компонента SimpleConnect

Если Вы создаете свое пользовательское FlashCom приложение и клиентский Flash movie, Вы вероятно будете иметь собственную систему логинов и элементы пользовательского интерфейса, содержащие поля логина и пароля. В таком случае, Вы будете соединять компоненты соединения с Вашим FlashCom приложением без использования компонента SimpleConnect. В этой последней секции, Вы узнаете какие возможности дает компонент ConnectionLight для приложения, при использовании Вашего собственного кода ActionScript.

Во-первых, создадим новое серверное приложение. На компьютере, где запущен FlashCom Server просмотрите папку applications и создайте новую папку под названием test_app в этой папке. В следующей секции Вы создадите клиентский объект NetConnection, который присоединит к приложению это пространство.

Создание клиентской части

Создайте новый Flash-документ (File → New) и сохраните этот документ как no_simpleconnect.fla.

Переименуйте Layer 1 в light_mc. В кадре 1 этого слоя перетащите экземпляр компонента ConnectionLight из панели компонентов. Назовите экземпляр light_mc, используя панель свойств.

Создайте новый слой и назовите его actions. Поместите этот слой над слоем light_mc.

Выберите первый кадр слоя actions и откройте панель Actions (F9). Добавьте следующий код:

```
app_nc = new NetConnection();
app_nc.onStatus = function (info) {
 trace("app_nc: info.code = " + info.code);
 if (this.initLight == null) {
  light_mc.connect(this);
  this.initLight = true;
 }
};
app_nc.connect("rtmp:/test_app");
```

Здесь же создайте новый объект NetConnection, который вызовет метод connect() экземпляра light_mc, когда вызван обработчик onStatus(). Внутренние свойства, названные initLight, используются чтобы отследить, был ли вызван метод connect() экземпляра light_mc, т.к. обработчик onStatus(), вероятно, будет исполнен более чем один раз в течение сессии приложения. Вам может понадобиться изменить RTMP URI, включая доменное имя Вашего FlashCom Server.

Сохраните Ваш Flash-документ и протестируйте его. Когда Flash movie загружается, Вы должны увидеть сообщение trace() обработчика app_nc.onStatus(). Если соединение прошло успешно, то статус-цвет экземпляра light_mc должен стать зеленым. Если соединение неудачно, то индикатор станет красным.

Вы можете продолжать изменять этот пример, путем добавления компонентов соединения. Поскольку Вы добавляете экземпляра компонентов, Вы можете упростить код обработчика onStatus(), путем создания массива резервных компонентов, ссылающихся на Ваши объекты компонентов. Например, следующий код создает объект component_array, который соединяет три компонента соединения в app_nc соединение:

```
app_nc.onStatus = function (info) {
 trace("app_nc: info.code = " + info.code);
 if (this.initComponents == null) {
  var components_array = [light_mc, bandwidth_mc, av_1_mc];
  for (var i in components_array) {
   components_array[i].connect(this);
  }
 this.initComponents = true;
 }
};
```

Заключение

Эта часть показала, как компоненты соединения инкапсулируют обычно необходимые возможности, такие как chat, video recording and playback, bandwidth control, user configuration и другие. Эти компоненты дают Вам возможность быстрого создания основных приложений и, затем возможность улучшить их с помощью дополнительных возможностей. Вы увидели, как быстро создать чат и добавить к нему аудио и видео.

Сейчас, когда у Вас уже есть базовое представление о создании простых FlashCom application, можно исследовать технические проблемы более подробно. В следующей части Вы рассмотрите создание и управление соединениями.

Лабораторная работа №3: Управление соединениями

Прежде, чем Flash ролик сможет обмениваться аудио, видео и другими данными с FlashCom Server приложением, он должен запросить сетевое подключение с сервером. Если прикладной экземпляр на сервере принимает запрос подключения, подключение TCP делается доступным и ролику, и экземпляру приложения для продолжения использования подключения. Мы видели, что компонент SimpleConnect может создать подключение. Эта часть охватывает подключения в большей степени, в том числе, как написать нестандартный код, чтобы обработать различные изменения в состоянии подключения так же, и различные типы ошибок.

Сетевое подключение может управляться в пределах Flash ролика, используя объект NetConnection. Методы этого объекта используются, чтобы запрашивать, завершать и контролировать состояние сетевого подключения. Классы SharedObject и NetStream, так же как компоненты связи Macromedia, зависят от доступа к объекту NetConnection надлежащим образом для связи с FlashCom Server.

Создание подключения

Объекты NetConnection создаются с использованием конструктора NetConnection():

```
lobby_nc = new NetConnection();
```

В этом примере переменная lobby_nc хранит объект NetConnection, который может использоваться для попытки соединения с экземпляром приложения:

```
if (lobby_nc.connect("rtmp:/testLobby", userName, password)) {
 trace("Attempting connection... ");
}
else {
 trace("Can't attempt connection. Is the URI correct?");
}
```

В этом примере, URI rtmp:/testLobby - адрес экземпляра приложения, названного testLobby, который постоянно хранится на сервере, с которого ролик был загружен. Метод connect() всегда передает URI экземпляра приложения, к которому подсоединяется, сопровождаемый каким-нибудь количеством дополнительных параметров. В этом примере, имя пользователя и пароль добавлены после URI. Метод connect() возвращает false, если он обнаруживает неправильную URI; иначе, он возвращает true, и Flash пытается подключиться к серверу.

Не все неправильные URI обнаруживаются методом connect(). Вы должны всегда удостоверяться, что протокол является или rtmp, или rtmpt, сопровождаемый двоеточием; иначе, метод connect() может возвратить истину и затем выйти из строя.

URI может быть абсолютной URI, которая включает имя хоста, или относительной URI, которая этого не делает. После вызова метода connect() URI делается в нем доступной как свойство uri объекта NetConnection.

Абсолютные URI

Абсолютная URI это строка, которая включает протокол, хост, дополнительный порт, имя приложения и дополнительные экземпляры имен сегментов. Формат абсолютной URI:

```
rtmp[t]://host[:port]/applicationName[/instanceName]
```

Часть URI в квадратных скобках дополнительная. Абсолютная URI содержит всю информацию, необходимую для идентификации экземпляра приложения, к которому подключаемся. Следующие разделы описывают все части абсолютных URI.

Протокол

Первая часть абсолютных URI - протокол или схема подключения, должна всегда определяться как rtmp, rtmpt, или rtmps и следовать перед :// для абсолютных URI или перед :/ для относительных URI. RTMP предпочтительнее, потому что это - родной протокол для FlashCom взаимодействия. Добавление дополнительной буквы t в rtmpt указывает, что должно использоваться туннелирование HTTP. Туннелирование менее эффективно, потому что RTMP сообщения должны быть заявлены в заголовках HTTP, и сервер должен постоянно опрашиваться Flash ролик. Однако туннелирование может успешно создавать подключения через брандмауэры и прокси сервера, когда доступ RTMP невозможен. Туннелирование было первым добавлено в Flash Player 6.0.65.0 и FlashCom 1.5. Когда запущен броузер, Flash Player использует броузер для того, чтобы сделать HTTP подключение с сервером, на котором располагается RTMPT. Так как

броузер обычно также поддерживает SSL, Macromedia добавили способность использовать SSL, устанавливая rtmps протокол вместо rtmpt. Когда использовался RTMPS, Flash Player пытался установить туннелированное подключение по HTTPS к FlashCom. Даже при том, что FlashCom 1.5 не поддерживает SSL на прямую, можно воспользоваться преимуществами RTMPS, используя дополнительное серверное программное обеспечение или специализированные сетевые аппаратные средства для того, чтобы полностью скрыть взаимодействие в реальном времени между Flash роликами и FlashCom. Выбор конфигурации системы по внедрению SSL соединений кратко описан на сайте:
`http://www.macromedia.com/devnet/mx/flashcom/articles/firewalls_proxy04.html`

Если SSL сертификат используемый для авторизации браузера не опознается, любого соединяющегося через RTMPS попросят принять или установить сертификат прежде, чем связь может быть закончена. После этого, при клике Ok (Да) для согласия, соединение будет разорвано. Сетрификат должен быть установлен, и соединение повторено.

Хост и порт

Абсолютные URI всегда содержат имя хоста после знаков ://. Это необязательно, но Вы можете выставить порт на хосте следующим hostname с двоеточием и номером порта. По умолчанию порт FlashCom 1935, однако, он может быть сконфигурирован для принятия соединения на другом порту или портах. Если протокол rtmp установлен, и никакой порт не передан в URI, Flash ролик пытается соединиться с портом 1935. Если попытка соединения безуспешна, Flash Player автоматически пытается соединиться с FlashCom по порту 443. Если это также терпит неудачу, то Player пытается соединиться с портом 80. Наконец, если RTMP соединение не может быть создано ни на одном из этих трех портов, Player пробует на одном из предыдущих шагов сделать туннелированное (RTMPT) соединение по порту 80. Последовательность портов и протоколов приведена в Таблице 4.

Таблица 4. Последовательность соединения портов

Последовательность	Порт	Протокол
1	1935	RTMP
2	443	RTMP
3	80	RTMP
4	80	RTMPT

Последовательность в Таблице 4 будет верна только тогда, когда протокол в URI - rtmp, и никакой порт не определен. Если rtmpt определен в URI, и ниодин порт не работает, Flash ролик пытается соединиться с портом 80. Когда используется rtmps, по умолчанию порт - 443.

Если FlashCom сконфигурирован для ожидания на другом порту, отличном от 1935, 443 или 80, то порт должен быть определен в URI. Например, если сервер сконфигурирован, чтобы использовать порт 8080, URI может быть похожа на:
`rtmp://host.domain.com:8080/conferenceRooms/room_23`

Тем не менее, выбор другого порта не рекомендован, так как порт 1935 был назначен Internet Assigned Numbers Authority для Macromedia Flash Communication Server MX. Конфигурация продукта может также установить сервер для ожидания на более, чем один порт. Информация порта может быть установлена при использовании файла конфигурации Adaptor.xml с признаком <HostPort>. Файл Adaptor.xml расположен внутри каждой директории адаптеров в директории conf.

При установках по умолчанию на Windows файл Adaptor.xml располагается по следующему адресу:

```
C:\Program Files\Macromedia\Flash Communication Server
MX\conf\_defaultRoot_\Adaptor.xml
```

Если web сервер не запускается на порту 443 или 80, <HostPort> признак может включать в себя номера портов 1935, 443, и 80. Если никакой номер порта не подается в заданном объекте URI и удаленный брандмауэр блокирует доступ к серверу по порту 1935, но разрешает доступ через порт 443, то вторая попытка соединения будет успешной. Порт 443 обычно связан со скрытыми взаимодействиями по SSL. Ни FlashCom, ни Flash Player не обеспечивают непосредственную поддержку SSL.

Имя приложения

Имя приложения должно быть определено в URI. На сервере имя приложения – это название подкаталога в серверной директории applications. Часто каждое приложение – это подкаталог в директории Macromedia Flash Communication Server MX/applications. Однако сервер может иметь многочисленные виртуальные хосты, каждый из которых с его собственным каталогом applications. Каждый виртуальный хост связан с доменом, который помечается в директории applications. Для того, чтобы определить расположение директории applications, проверьте признак <AppsDir> в файле Vhost.xml. По умолчанию Vhost.xml расположен в подкаталоге директории, в которую был установлен FlashCom conf/defaultRoot/defaultVHost_. Сама директория applications никогда не является частью URI, как имя хоста, которое отображается в ней.

Имя экземпляра

Имя экземпляра определяет какой из экземпляров приложения будет устанавливать соединение. Оно не должно быть определено в пределах URI. Если этим пренебрегли, то по умолчанию используется имя _definst_. Эти два URI обращаются к одному образцу:

```
rtmp://host.domain.com/conferenceRooms
rtmp://host.domain.com/conferenceRooms/_definst_
```

Имена экземпляров приложений могут формировать иерархию. Например, приложение комнаты для дискуссий (чат) может обеспечивать несколько сессий по различным тематикам. Для обеспечения этой ситуации структура имен экземпляров может придерживаться схемы, в которой иерархия помогает сохранить сформированность единиц, аналогично следующей:

```
program/courseName/chatRoom
```

Так некоторые URI могут выглядеть следующим образом:

```
rtmp://my.university.edu/chatRooms/science/phys325/room8
rtmp://my.university.edu/chatRooms/motionPicture/film001/room8
rtmp://my.university.edu/chatRooms/motionPicture/film021/room3
rtmp://my.university.edu/chatRooms/motionPicture/film021/room4
```

Каждый из этих URI - адрес отдельного экземпляра приложения chatRooms, даже в том случае, если два из них содержат имя "room8". Имя экземпляра

```
/chatRooms/motionPicture/film001/room8
```

не то же самое, что и

```
chatRooms/science/phys325/room8
```

Относительные URI

Если файл *.swf* загружен в браузер с того же самого сервера, на котором работает FlashCom, вы можете использовать относительные URI, в которых знаки // и имя хоста опущены. Вот форма для относительных URI:

```
rtmp[t]:[:port]/applicationName[/instanceName]
```

Обратимся к ранее рассмотренному примеру rtmp:/testLobby, протокол – rtmp, одинарный слеш служит признаком относительной URI и testLobby – имя приложения.

Никакое имя экземпляра приложения не предусмотрено, так что экземпляр _definst_ не допустим.

Ожидание соединения

После успешного вызова метода connect() (который в особенности формирует URI) объект NetConnection должен ждать, чтобы заметить успешную попытку соединения. Это может занять некоторое время, поскольку могут случиться несколько вещей:

Сервер может не работать или быть недоступным, в этом случае попытка соединения в итоге не даст успеха.

Сеть может быть медленной или сервер может быть занят.

Экземпляр приложения может принять сетевое подключение TCP, но оставить его в ждущем состоянии, в то время пока он решает, отклонить его или принять соединение.

В конечном счете, объект NetConnection будет уведомлен через его метод onStatus() о том, что произошло. По умолчанию нет никакого метода onStatus(), так что вы должны определить программу обработки onStatus() перед тем, как вы вызовете NetConnection.connect(). Метод onStatus() известен как обработчик событий или обработчик обратного вызова, потому что он вызывается в ответ на наступление события (в этом случае, сервер отреагирует на попытку соединения). Обработчик onStatus() получает объект, содержащий информацию о событии, которое помогает интерпретировать ответ.

Вот стандартный метод onStatus(), который только выводит информацию, содержащуюся в объекте, который он получает:

```
NetConnection.prototype.onStatus = function (info) {
 trace("this.isConnected: " + this.isConnected);
 trace("   info.level: " + info.level);
 trace("    info.code: " + info.code);
 trace("info.description: " + info.description);
 if (info.application) {
  for (var prop in info.application) {
   trace("info.application." + prop + ": " + info.application[prop]);
  }
 }
 trace("\n");
};
```

Объект, полученный onStatus(), называнный в примере info, имеет три свойства: level, code и description. Свойство level указывает, является ли результат “error” или уведомлением “status”. Свойство code обладает наиболее полной информацией и может использоваться в обработчиках onStatus() для того, чтобы решать какое действие считать как наступление события. Предшествующий пример также использует свойство объекта NetConnection isConnected. Если соединение было установлено, isConnected – true, иначе isConnected – false.

Пример показывает простой тестовый ролик, сопровождаемый выводом сообщений, которые отображаются, если соединение успешно.

Пример 5. Короткий тестовый скрипт для вывода сообщений о соединении

```
// Some credentials to pass to the application.
userName = "Guest";
password = "Guest";

// First, define an onStatus() handler.
NetConnection.prototype.onStatus = function (info) {
  trace("this.isConnected: " + this.isConnected);
  trace("      info.level: " + info.level);
  trace("       info.code: " + info.code);
  trace("info.description: " + info.description);
  if (info.application) {
    for (var prop in info.application) {
      trace("info.application." + prop + ": " + info.application[prop]);
```

```
    }
  }
  trace("\n");
};

// Next, create a NetConnection object.
lobby_nc = new NetConnection();

// Finally, attempt to make the connection.
// The onStatus() handler will be invoked when the operation completes.
if (lobby_nc.connect("rtmp:/testLobby/", userName, password)) {
  trace("Attempting connection...");
}
else {
  trace("Can't attempt connection. Is the URI correct?");
}
```

Если соединение установлено, свойство code информационного объекта передает обработчику onStatus() содержание строки "NetConnection.Connect.Success":

```
Attempting connection...
this.isConnected: true
   info.level: status
    info.code: NetConnection.Connect.Success
info.description: Connection succeeded.
```

С другой стороны, соединение могло потерпеть неудачу. Возможные причины для провала включают в себя отсутствие приложения testLobby на сервере или если сервер не может быть найден. В этих случаях обработчик выведет отчет о состоянии “error” в свойстве level и дополнительную информацию о состоянии, доступную свойству code информационного объекта (в этом случае description пустое свойство).

```
Attempting connection...
this.isConnected: false
   info.level: error
    info.code: NetConnection.Connect.Failed
info.description:
```

Экземпляр приложения может также отклонить запрос соединения (например, если значения userName или password недопустимы). Если соединение отклонено, метод onStatus() вызван дважды с сообщением об ошибке и позже с обновлением состояния. Обратите внимание на значения info объектов свойств level и code в обоих случаях:

```
Attempting connection...
this.isConnected: false
   info.level: error
    info.code: NetConnection.Connect.Rejected
info.description: Connection failed.
info.application.message: This application has refused your connection.

this.isConnected: false
   info.level: status
    info.code: NetConnection.Connect.Closed
info.description:
```

Когда экземпляр приложения отклоняет попытку соединения, он может возвратить объект с дополнительной информацией. Если это так, то объект возвратился, поскольку свойство application объекта info передано в метод onStatus(). Для использования объекта application вам необходимо или знать его свойства заранее, или пройтись циклом for-in по всем его свойствам, как показано в примере 4. Если вы знаете свойства, которые использует конструктор скрипта приложения, к ним можно обратиться без цикла for-in. Например, свойство, названное message, может быть доступно следующим образом:

```
if (info.application) {
 trace("info.application.message: " + info.application.message);
}
```

Управление соединением

После того, как соединение успешно установлено, оно может быть разорвано различными событиями. Хорошо написанный скрипт Flash ролика контролирует состояние соединения и предпринимает соответствующие действия, в то время как происходят какае-либо изменения работоспособности. Это легко смоделировано в Вашем обработчике NetConnection.onStatus(), который уведомляется всякий раз, когда статус соединения изменяется. Когда соединение установлено впервые, объекты и компоненты, которые полагаются на него, могут начать использовать соединение. Когда соединение закрыто, эти объекты могут быть повреждены или ликвидированы. Особенно важно, что пользователь не испытывает внезапную и необъяснимую потерю выполняемых функций, когда сетевое соединение удалено сервером или упущено из-за сетевых проблем.

Компилятору иногда сложно исправить ошибки, когда скрипты определяют отладчик onStatus() после вызова метода connect().

```
nc = new NetConnection();
nc.connect(); // WRONG: do this after onStatus is defined!
nc.onStatus = function (info) {
 trace("info.code: " + info.code);
};
```

Если connect() вызван преждевременно, то сетевое соединение может быть установлено или соединение отклонено перед тем, как был определен обработчик onStatus(). Результатом будет то, что сообщения, символизирующие эти события, никогда не будут возвращаться обработчику. Метод onStatus() всегда должен подключаться перед вызовом connect(). Предыдущий пример должен выглядеть следующим образом:

```
nc = new NetConnection();
nc.onStatus = function (info) {
 trace("info.code: " + info.code);
};
nc.connect();
```

В основном, когда подключение закрыто, в переданном методу onStatus() объекте info значение level будет иметь “status” off и значение code "NetConnection.Connect.Closed". Это случится, если связь ухудшается, сервер отключает клиента, сервер останавливается или соединение завершается при использовании NetConnection.close(). Это также возможно, если значение level будет “ошибка” и значение code будет "NetConnection.Connect.AppShutdown".

Успешное подключение

Когда Flash устанавливает соединение, могут быть установлены любые потоки и совместно используемые объекты, которые полагаются на подключение. Проверка кода "NetConnection.Connect.Success" в обработчике onStatus() - простейший способ сделать это:

```
if (info.code = "NetConnection.Connect.Success") {
 // Initialize and connect dependent objects and components (not shown).
}
```

Решение проблем

Когда соединение не может быть установлено или упущено, на обработчик onStatus() можно послать столько возможных вариантов сообщений:

один код ошибки

код ошибки, следующий за состоянием закрытия

одно состояние закрытия

Свойство isConnected объекта NetConnection – false, когда любое из этих событий произошло. Когда бы ни произошел такой инцидент, вы можете требовать, чтобы ваш скрипт задерживал действие на некоторое время, как, например, информирование

пользователя. Простейший способ разрешить эту проблему – проверить свойство isConnected в обработчике onStatus():

```
if (!this.isConnected) {
 if (info.code = "NetConnection.Connect.Rejected") {
  // Tell the user the connection was rejected by the application.
 }
 else {
  // Give the user more info based on info.code and/or info.level.
 }
 // Clean up any objects and components that need them.
 // Change state by going to another frame on the main timeline.
}
```

Отображение сообщения всякий раз, когда isConnected – false, может позволить пользователю, видящему два сообщения, отклонить то сообщение, которое следует за закрытым сообщением. Чтобы избежать отображения двух сообщений обработчик onStatus() должен быть остановлен действием в соответствии со вторым сообщением. Единственный способ это сделать – полностью удалить обработчик onStatus() посредством установки свойств onStatus() в null после появления первого сообщения. Это можно сделать внутри метода onStatus():

```
this.onStatus = null;
```

Удаление обработчика onStatus() для того, чтобы могла происходить дальнейшая обработка сообщений, может выглядеть слишком радикальным решением. Тем не менее, свойства onStatus() позже могут быть перезагружены для того, чтобы метод мог обрабатывать сообщения. В качестве альтернативы работающему обработчику onStatus() может быть создан и применим для повторного соединения другой объект NetConnection. Другой подход состоит в том, чтобы создать дополнительное свойство объекта NetConnection, которое переключает обработку определенных сообщений. Когда сообщения ожидаются после вызова connect() или после статуса сообщения “success”, свойства устанавливаются в true. А когда соединение было закрыто, свойства могут быть установлены в false. Дальнейший пример показывает один из способов, как это можно сделать.

Закрытие соединения со стороны клиента

Другая потенциальная проблема состоит в том, что когда вы позволяете пользователю закрыть соединение с помощью NetConnection.close() обработчик onStatus() по прежнему принимает сообщение со статусом “close”. Часто маленькая заминка в демонстрации этого сообщения пользователю окажет большое влияние. Возможно, более просто непосредственно изменить состояние приложения, например, перемещая ползунок основной временной шкалы. С другой стороны, свойство может храниться в объекте NetConnection или метод onStatus() может быть установлен в null для того, чтобы избежать отображение сообщения завершения.

Пример использует свойство, названное handleCloseEvents, для того, чтобы показать была ли ошибка, и обработалось ли сообщения завершения. Это только один способ написания обработчика onStatus(), который использует этот тип флага, и он может вносить требуемые изменения. Рисунок 34 показывает временную шкалу для ролика, использованного в примере. Ролик может находиться в одном из трех состояний, каждое из которых изображено меткой в начале последовательности кадров. Init начинается на первом кадре и проигрывается только тогда, когда ролик загружается сначала; кадр Login появляется, когда ролик находится в отключенном состоянии, и пользователь может попытаться подключиться; кадр Connected появляется, когда ролик находится в подключенном состоянии.

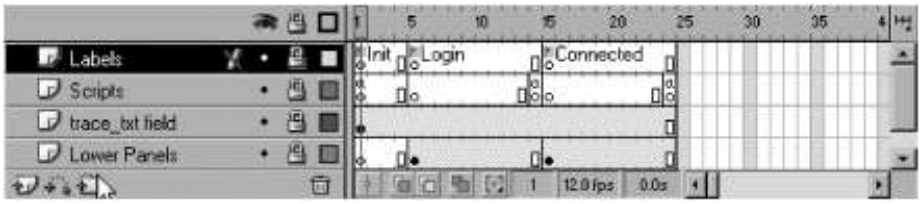

Рисунок 34. Временная шкала из примера

Скрипт в примере, расположенный в 1 кадре в слое Scripts главной временной шкалы, отображает пользовательские сообщения в текстовом поле. ActionScript переводит ползунок в кадр Login или в кадр Connected в зависимости от сетевого состояния.

Пример 6. Скрипт, обрабатывающий события, связанные с соединением

```
// writeln() writes messages into a text field named trace_txt
// and is a runtime alternative to trace() and the Output panel.
function writeln (msg) {
  trace_txt.text += msg + "\n";
  trace_txt.scroll = trace_txt.maxscroll;
}

/* A simple onStatus() handler that moves the playhead to Login if a
 * connection is closed and to Connected when a connection is established.
 * User messages are written via writeln() into a text window.
 * The this.handleCloseEvents flag is used to make sure the user never
 * sees redundant messages.
 */
NetConnection.prototype.onStatus = function (info) {
  // Always deal with successful connections.
  if (info.code == "NetConnection.Connect.Success") {
    this.handleCloseEvents = true;
    writeln("Success, you are connected!");
    gotoAndPlay("Connected");
  }
  // Handle messages when the connection is closed.
  if (!this.isConnected && this.handleCloseEvents) {
    if (info.code == "NetConnection.Connect.Rejected") {
      writeln(info.application.message);
      writeln('Did you use the username "Guest" and password "Guest"?');
    }
    else {
      writeln("Error: Connection Closed.");
    }
    this.handleCloseEvents = false;
    gotoAndPlay("Login");
  }
  // Handle remote method call errors here if you need to.
};

// Create a NetConnection object.
lobby_nc = new NetConnection();

// Called when the Connect button in the Login frames is clicked.
function doConnect () {
  lobby_nc.handleCloseEvents = true;
  if (lobby_nc.connect("rtmp:/testLobby/",
                       userName_txt.text, password_txt.text)) {
    writeln("Please wait. Attempting connection...");
  }
  else {
    writeln("Can't attempt connection. Is the URI correct?");
  }
}
```

```
// Called when the Disconnect button in the Connected frames is clicked.
function doDisconnect() {
  // User is requesting a close so we don't handle it in onStatus().
  lobby_nc.handleCloseEvents = false;
  lobby_nc.close();
  gotoAndPlay("Login");
}

gotoAndPlay("Login");
```

Сценарий, показанный в пример, нуждается в приложении, к которому необходимо подключится. Поэтому нам необходим на сервере в директории applications подкаталог testLobby, от которого загружался бы ролик. Однажды загруженный из директории, файл main.asc, показанный в следующем примере, может быть расположен в подкаталоге testLobby для того, чтобы обеспечить приложение собственной логикой. Этот серверный скрипт принимает соединения только от тех пользователей, которые зарегистрированы, используя имя пользователя “Guest” и пароль “Guest”. Отвергнутым клиентам пересылаются сообщения для отображения пользователям.

Пример 7. Серверный скрипт (main.asc) для принятия или отклонения соединений, основанный на имени пользователя и пароле

```
/* The application.onConnect() method handles each client
 * connection request. In this case, users who log in
 * with userName "Guest" and password "Guest" are
 * allowed to connect. Other connection requests are rejected.
 */
application.onConnect = function (client, userName, password) {
  if (userName == "Guest" && password == "Guest") {
    client.writeAccess = "";   // Don't give write access.
    application.acceptConnection(client);
  }
  else {
    application.rejectConnection(client,
      {message: "This application has refused your connection."});
  }
};
```

Всякий раз, когда Flash ролик подключается к экземпляру приложения, вызывается метод onConnect() объекта экземпляра application. Первый параметр, переданный onConnect() всегда объект Client, который представляет серверный Flash ролик. Остальные параметры - параметры, переданные в метод connect() в ролике. Поэтому, когда клиент пытается подключиться к FlashCom приложению testLobby, файл main.asc, показанный в примере, проверяет переданные имя пользователя и пароль. Полностью готовая версия могла проверять достоверность имени пользователя и пароля, опираясь на базу данных надежных пользователей.

Использование соединения

Объект NetConnection представляет сетевое соединение в пределах Flash ролика. Любой объект, который взаимодействует через сетевое соединение, должен иметь доступ к объекту NetConnection. Например, класс NetStream требует, чтобы объект NetConnection был передан в его конструктор:

```
myStream_ns = new NetStream(myNetConnection);
```

Подобным образом удаленный совместно используемый объектный метод connect() должен быть передан в объект NetConnection:

```
myRemote_so = SharedObject.getRemote("SOName", myNetConnection.uri, true);
myRemote_so.onStatus = onStatusFunction;
if (myRemote_so.connect(myNetConnection)) {
 trace("Connection ok so far...");
}
```

Macromedia разработала оба класса NetStream и SharedObject так, чтобы, с некоторыми исключениями, они могли быть использованы, как только им будет передан

объект NetConnection. В обоих случаях NetConnection должен быть таким объектом, чтобы пытаться создать подключение, даже в том случае, если соединение еще не установлено. Например, процесс пересылки данных может быть начат через вызов методов NetStream.send() или SharedObject.send() прежде чем установлено соединение. Данные удерживаются в очереди, пока создается соединение. Точно также объект NetStream может начать публикацию или проигрывание аудио или видео перед созданием подключения. В этом случае, когда соединение установлено, поток начнет публиковать или играть.

Важное исключение из этого - совместно используемые данные и общий объектный обработчик onSync(). Как дополнительная возможность, если обработчики onStatus() не созданы для объектов SharedObject и NetStream, обработчик onStatus() NetConnection'a, с которым они связаны, будет вызван и передан информационным объектам SharedObject и NetStream. В некоторых случаях, особенно, когда передаются данные, эти удобства полезны, но для больших частей данных не должны применяться. Достаточно просто управлять приложением обеспечивать пользователя лучшей информацией, если объекты, которые зависят от сетевого подключения, определяют их собственный обработчик событий onStatus() и ничего не делают, пока соединение не будет успешно установлено.

Повторное использование объекта NetConnection

Достаточног часто Flash ролик должен подключиться более чем к одному экземпляру приложения. Типичный пример, когда Flash должен впервые соединиться с lobby, таким образом, пользователь может выбрать для посещения один из чатов. В этом случае lobby может быть одним приложением, а чат может выполняться другим приложением. Соединение lobby может быть закрыто и затем может быть создан новый объект NetConnection с другим методом onStatus() для того, чтобы подключить приложение чат. Вместо создания нового объекта NetConnection Flash ролик может повторно использовать уже существующий. В теории Вы можете отключиться от одного приложения и подключиться к другому вызовом метода connect() с новыми флажками URI. Когда это произойдет, старое соединение закроется, а новое попытается установиться. Однако, случаются обычно две другие вещи:

Во-первых, перед тем, как соединение закроется, вы должны выполнить некоторую требуемую очистку, такую как закрывание объектов и компонентов, которые зависят от подключения. Во-вторых, вы должны выполнить некоторую предварительную работу, как минимум вы должны поместить на место обработчик onStatus() перед подключением к другому приложению.

Чтобы завершить предыдущий пример, создадим одно дополнительное приложение, названное testChat, доступным таким образом, что когда пользователь находится в lobby, он может щелкнуть на кнопку для того, чтобы посетить экземпляр testChat, названный room1. В этом случае главная временная шкала ролика требует отдельных кадров Login, Lobby и ChatRoom, как показано на рисунке 35.

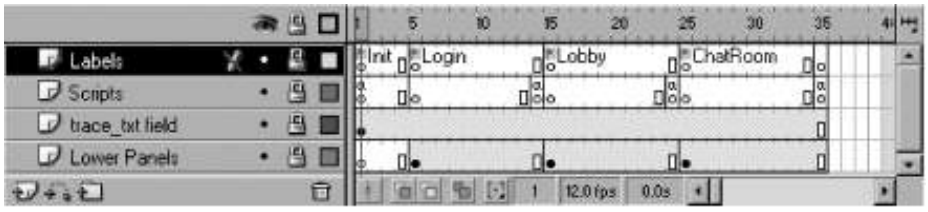

Рисунок 35. Временная шкала с состояниями Login, Lobby и ChatRoom

Кнопка Chat должна располагаться в кадре Lobby, а кнопка Lobby должна располагаться внутри кадра ChatRoom. Следующий пример демонстрирует функцию onChat(), которая должна вызываться при нажатии на кнопку Chat.

Пример 8. Подключение из lobby к chat room

```
function doChat(btn) {
  // Don't process the next close message.
  lobbyChat_nc.handleCloseEvents = false;
  // Close the connection to the lobby.
  lobbyChat_nc.close();

  // Set the onStatus() handler, defined in Next Example.
  lobbyChat_nc.onStatus = ChatRoom_onStatus;
  // Make sure events are handled by it.
  lobbyChat_nc.handleCloseEvents = true;

  // Try to connect to a chat room.
  if (lobbyChat_nc.connect("rtmp:/testChat/room1", userName, password)) {
    writeln("Please wait. Attempting chat room connection...");
  }
  else {
    writeln("Can't attempt connection. Is the URI correct?");
  }
}
```

В примере обработка события завершения сеанса связи и сообщений об ошибках соединения выключена, так что управление не передается обратно кадру Login, когда Flash вызывает lobbyChat_nc.close(). Пример назначает новый обработчик onStatus(), названный ChatRoom_onStatus(), объекту lobbyChat_nc следующим образом:

```
lobbyChat_nc.onStatus = ChatRoom_onStatus;
```

В конце, когда подключение предпринято, две глобальные переменные, userName и password, используются для того, чтобы отыскать имя пользователя и пароль для подтверждения, потому что текстовые поля больше не располагаются на Сцене, когда пользователь не находится в lobby. Следующий пример демонстрирует код функции ChatRoom_onStatus(). Она используется как новый обработчик onStatus() для объекта lobbyChat_mc, как показано в предыдущем коде:

Пример 9. Обработчик chat room onStatus()

```
function ChatRoom_onStatus(info) {
  // Always deal with successful connections.
  if (info.code == "NetConnection.Connect.Success") {
    this.handleCloseEvents = true;
    writeln("Success, you are connected to a chat room!");
    gotoAndPlay("ChatRoom");
  }
  // Handle messages when the connection is closed.
  if (!this.isConnected && this.handleCloseEvents) {
    // Handle close/error messages.
    if (info.code == "NetConnection.Connect.Rejected") {
      writeln(info.application.message);
      writeln('Did you use the username "Guest" and password "Guest" ?');
    }
    else {
      writeln("Error: Connection Closed.");
    }
    this.handleCloseEvents = false;
    gotoAndPlay("Login");
  }
  // Handle remote method call errors here if you need to.
}
```

Когда пользователь находится в chat room, он может вернуться в lobby, нажав на кнопку Lobby. Следующий пример показывает код функции doLobby(), которая вызывается, когда пользователь кликает на кнопку Lobby. Она очень похожа на функцию

doChat() в примере 8. Помимо этого она устанавливает Lobby_onStatus() как новый обработчик onStatus() для объекта lobbyChat_mc. Описание функции Lobby_onStatus() здесь не показано, но оно очень похоже на описание функции ChatRoom_onStatus(), продемонстрированного в примере 9.

Пример 10. Функция doLobby()

```
function doLobby(btn) {
  // Don't process the next close message.
  lobbyChat_nc.handleCloseEvents = false;
  // Close the connection to the chat room.
  lobbyChat_nc.close();

  // Set the onStatus() handler (definition of Lobby_onStatus() is not
shown).
  lobbyChat_nc.onStatus = Lobby_onStatus;
  // Make sure events are handled by it.
  lobbyChat_nc.handleCloseEvents = true;

  // Try to connect to the lobby.
  if (lobbyChat_nc.connect("rtmp:/testLobby/", userName, password)) {
    writeln("Please wait. Attempting lobby connection...");
  }
  else {
    writeln("Can't attempt connection. Is the URI correct?");
  }
}
```

Теперь, когда Вы увидели, как повторно использовать подключение, давайте посмотрим, как Вы можете одновременно использовать многочисленные подключения.

Многочисленные одновременные объекты NetConnection

Macromedia не рекомендует создание многочисленных подключений одного Flash ролика с серверам или серверу FlashCom в одно и то же время. Прежде всего, необходимо учитывать сможет ли сервер обработать столько одновременных подключений и передать столько данных, сколько через него проходит. Кроме того, FalashCom ограничен по лицензии на количество одновременных подключений. Поэтому, всякий раз, когда возможно, старайтесь избегать конструировать взаимодействие приложений, которые требуют многочисленных одновременных соединений. Однако Flash ролик может подключиться одновременно к различным приложениям или экземплярам одного и того же приложения; также он может многократно подключаться к одному экземпляру приложения, если это необходимо. Это свойство особенно полезно, когда приложения тестируются или при разработке больших, масштабных многоэкземплярных приложений.

Простой тест-скрипт в примере создает два подключения к одному и тому же экземпляру chat room.

Пример 11. Использование более одного объекта NetConnection

```
NetConnection.prototype.onStatus = function(info) {
  trace(this.name + ": " + info.code);
};
nc1 = new NetConnection();
nc1.name = "First connection";
nc2 = new NetConnection();
nc2.name = "Second connection";
nc1.connect("rtmp:/testChat/room1", "Guest", "Guest");
nc2.connect("rtmp:/testChat/room1", "Guest", "Guest");
```

Предполагая, что нет никаких проблем с сетью или с сервером, запись вывода теста будет выглядеть следующим образом:

```
First connection: NetConnection.Connect.Success
Second connection: NetConnection.Connect.Success
```

Тестирование и отладка сетевых подключений

Сетевые приложения представляют несколько дополнительных трудностей при отладке и тестировании программ, так как вовлекается более, чем одна система. Попытка подключения может потерпеть неудачу по нескольким причинам, включая следующие:

Флажок URI может быть неправильным (смотрите опечатки).

Имя пользователя и пароль могут быть неверными.

Максимальное количество зарегистрированных пользователей или максимальная пропускная способность сервера могли быть превышены.

Сеть могла ослабнуть или она ненадежна.

Подключение, которое прекрасно работало, во время тестирования вышло из строя из-за брандмауэра или из-за прокси сервера.

Часто наилучшим способом для того, чтобы понять, почему сетевое подключение повреждено, является отображение всех свойств информационных объектов, возвращаемых в метод onStatus(). К тому же, по крайней мере пока флажки URI корректны, возврат значений метода connect() должен всегда проверяться. Если он является false, то URI неверна и должна быть исправлена. желательно всегда начинать построение ролика с создания по умолчанию диагностического обработчика onStatus(), как описано в одном примеров рпнее. Когда соединение будет протестировано, вы можете переместить обработчик onStatus() куда захотите. К сожалению, проблемы не ограничиваются началом разработки. Клиентскому и серверному скриптам сложно обнаружить внезапную причину потери соединения при диагностике. Для того, чтобы помочь обнаружить проблему, создается простой тестовый ролик, который может запускаться вне среды разработки. На самом деле этот метод тестирования делает возможность соединения намного лучше.

Тестирование клиентского ролика

Превосходный способ помогать отлаживать подключения состоит в том, чтобы записать тестовый ролик, который включает несколько основных компонентов, например:

Большое текстовое поле прокрутки с полосой прокрутки.

Кнопки Connect и Close.

Поля для флажков URI и других параметров метода connect(), таких как имя пользователя и пароль.

Вместо использования операторов trace() используйте функцию writeln(), как в примере ранее, для вывода информации в текстовое поле прокрутки.

Использование отладчика NetConnection

Отдельно от отладчика Flash MX 2004 ActionScript, Macromedia разработали NetConnection Debugger с FlashCom Server'ом (с Flash Remoting). NetConnection Debugger может отображать связанные с соединением клиентские и серверные события, такие, как запросы подключения и удаление вызванных методов и их данных. Для получения NetConnection Debugger загрузите и установите Macromedia Flash Remoting MX Components с сайта:

```
http://www.macromedia.com/software/flashremoting/downloads/components
```

После первой установки, для использования NetConnection Debugger, расположите следующую команду include на первом кадре слоя Scripts вашей главной временной шкалы:

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

Затем выберите Window → NetConnection Debugger в строке меню Flash'а для того, чтобы запустить отладчик. Для того, чтобы видеть серверные события, вы должны зарегистрироваться на сервере под профилем администратора (Admin Service), используя имя пользователя и пароль администратора, как описано в в первой части. Если раздел NetConnection Debugger'а Filters, показанный в нижней части рисунка, закрыт, щелкните на маленьком треугольнике, чтобы его открыть. Введите имя пользователя и пароль, удостоверьтесь, что Flashcom_server, RTMP и Trace checkbox's выделены; и запустите ваш ролик в среде разработки Flash. Рисунок показывает NetConnection Debugger после неудачной попытки подключения к серверу.

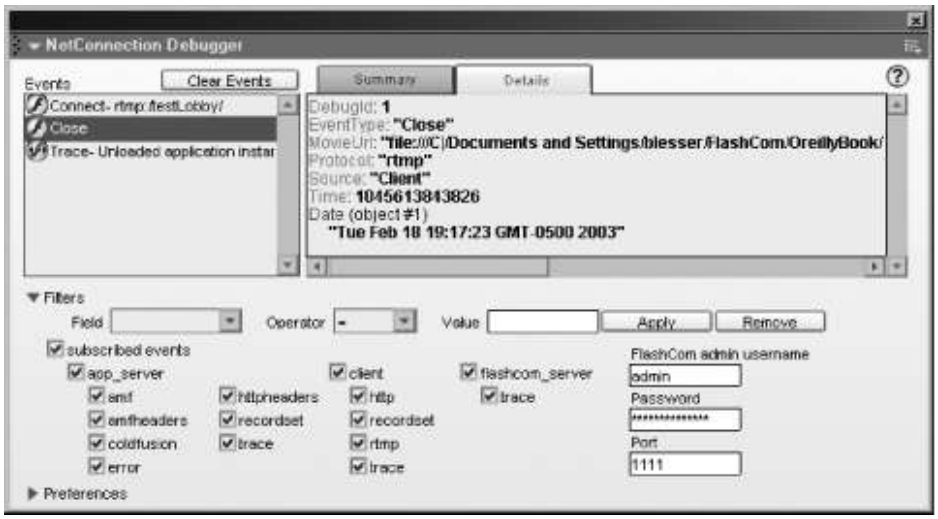

Рисунок 36. NetConnection Debugger после неудачной попытки подключения

Кликнув на событие из списка Events и выбрав вкладку Details, вы увидите все подробности о каждом из событий. Клиентские события обозначены логотипом Flash, серверные события обозначены логотипом FlashCom. На рисунке 36 показаны подробности события Закрыть. Следующее в списке Events – серверное событие, которое уведомляет, что приложение в итоге было выгружено, так как ни один клиент не подключился к нему. Если *.swf* был откомпилирован с файлом *NetDebug.as*, который включен в него, то он может быть запущен вне среды авторской разработки, и его события, связанные с подключением, все еще будут отображаться в NetConnection Debugger'е. Поскольку ваши проекты развиваются, и вы отлаживаете все более и более сложные скрипты, это еще одна серьезная причина для создания и сохранения нескольких простых тестовых роликов.

Удалите ActionScript, чтобы включить *NetDebug.as* в ваш итоговый ролик после того, как Вы больше не нуждаетесь в свойствах отладки. Он добавляет лишний размер файла к издаваемому *.swf*

Подклассификация класса NetConnection

Вы можете расширить возможности класса NetConnection построением на его основе подклассов. Создание подклассов позволяет вам инкапсулировать программу, которая работает на объекте NetConnection, в пределах методов подклассов, таких, как connect(), onStatus() или close(). В свою очередь это может привести к более модульному и простому содержанию приложений. Код для выполнения этого должен содержать оператор super. Следующий пример показывает простой образец расширения класса NetConnection и затем использования его в тестовом скрипте.

Пример использует синтаксис AS 1.0, поскольку он работает и в Flash MX, и в Flash MX 2004. Вам также предоставляется возможным компилировать ActionScript 1.0 в File → Publish Settings → ActionScript.

Пример 12. Простое расширение класса NetConnection

```
// Constructor function must call super().
function NetConnectionSubClass () {
  super();
}

// Subclass NetConnection by assigning an instance of it
// to the prototype object of the new subclass.
NetConnectionSubClass.prototype = new NetConnection();

// A simple demonstration onStatus() event handler that just
// writes out each info.code message.
NetConnectionSubClass.prototype.onStatus = function (info) {
  trace("info.code: " + info.code);
};

/* This connect() method may be passed any number of arguments.
 * The apply() method makes sure all arguments are passed into
 * the super.connect() method when it is called.
 */
NetConnectionSubClass.prototype.connect = function() {
  return super.connect.apply(super, arguments);
};

// Create an instance of the NetConnectionSubClass subclass.
lobby_nc = new NetConnectionSubClass();

// Attempt the connection.
if (lobby_nc.connect("rtmp:/testLobby/", "Guest", "Guest")) {
  trace("Attempting connection...");
}
else {
  trace("Can't attempt connection. Is the URI correct?");
}
```

Оператор super используется двумя различными способами в примере. Во-первых, он должен быть вызван внутри конструктора следующим образом:

```
super();
```

Вызов super() этим способом вызывает конструктор NetConnection (конструктор подкласса). Во-вторых, всякий раз, когда метод NetConnection должен быть вызван методом подкласса, оператор super может быть использован для вызова метода подкласса. В этом примере, когда вызван метод NetConnectionSubClass.connect(), он в свою очередь использует super для того, чтобы вызвать метод NetConnection.connect() следующим образом:

```
return super.connect.apply(super, arguments);
```

Использование метода apply() обеспечивает хороший, полный, успешный способ вызова метода подкласса connect(). Передача super как параметра гарантирует, что connect() вызван как метод суперкласса NetConnection, а передача arguments гарантирует, что все параметры переданы методу подкласса connect().

Более простой, но менее универсальный способ вызова метода NetConnection.connect() состоит в том, чтобы индивидуально передать отдельные параметры:

```
NetConnectionSubClass.prototype.connect = function(targetURI, userName,
password) {
 return super.connect(targetURI, userName, password);
};
```

В тех случаях, когда значения параметров должны регулироваться перед передачей в метод подкласса connect(), или тогда, когда не все параметры должны быть переданы, легче отказаться от использования метода apply().

Самостоятельно обратите внимание, что пример показывает основные механизмы создания подкласса класса NetConnection, но давайте посмотрим, какие преимущества он демонстрирует. Посмотрев снова на примеры ранее, вы увидите, что перед вызовом методов connect() или close() флажок handleCloseEvents должен устанавливаться каждый раз. Это дополнительная работа, которую не придется делать в функциях doConnect(), doChat() и doLobby().

На самом деле эти тонкости могут быть скрыты внутри методов подкласса NetConnection следующим образом:

```
// close() turns off the handleCloseEvents flag before closing the
connection.
LobbyChatConnection.prototype.close = function() {
  this.handleCloseEvents = false;
  super.close();
};
```

Если Вы измените метод connect() для того, чтобы также установить флажок handleCloseEvents, функция doLobby() может быть очень упрощена:

```
// Called when the Lobby button in a chat room is clicked.
function doLobby(btn) {
  lobbyChat_nc.close();
  lobbyChat_nc.connect("rtmp:/testLobby", handleLobbyConnection, userName,
password);
}
```

Сравните предыдущий код для функции doLobby() в примере ранее. Вы можете заметить, что кое-чего еще не хватает. В разделе этой части “Повторное использование объекта NetConnection” были написаны два обработчика событий onStatus(): один для управления подключениями lobby, другой для управления подключениями chat room. Функции doLobby() и doChat() были ответственными за обеспечение прав, обработчик onStatus() был основным перед вызовом connect(). В некоторых случаях это необходимо. Однако, когда обработчики onStatus() почти идентичны, есть более изящное решение. Главное отличие между обработчиками – кадр, который они перемещают ползунком, когда создано подключение. Более универсальный обработчик onStatus() может вызвать функцию, когда соединение установлено. В следующем примере обработчик onStatus() вызывает одну из двух функций в зависимости от того, какое приложение подключено:

```
// Called when a connection has been established to the testLobby app.
function handleLobbyConnection() {
  writeln("Success, you are connected to the lobby!");
  gotoAndPlay("Lobby");
}

// Called when a connection has been established to a chat room.
function handleChatConnection() {
  writeln("Success, you are connected to a chat room!");
  gotoAndPlay("ChatRoom");
}
```

Как обработчик onStatus() узнает, какую из функций вызывать? В этом примере ссылка на одну из них передана в метод connect() и сохранена в свойстве connectionHandler. Пример является законченным листингом тестового скрипта, в котором метод onStatus() использует свойство connectionHandler для того, чтобы вызвать или функцию handleLobbyConnection(), или handleChatConnection().

Пример 13. Тестовый скрипт подкласса NetConnection

```
// writeln() writes messages into a text field named trace_txt
// and is a replacement for using trace() and the Output panel.
function writeln (msg) {
  trace_txt.text += msg + "\n";
```

```
  trace_txt.scroll = trace_txt.maxscroll;
}

// LobbyChatConnection() is a constructor function for a NetConnection
subclass.
// It will not work unless super() is called within the function.
function LobbyChatConnection () {
  super();
}

LobbyChatConnection.prototype = new NetConnection();     // Subclass
NetConnection.
LobbyChatConnection.prototype.handleCloseEvents = true;  // Initial value.

// connect() is always passed four parameters, updates the handleCloseEvents
flag,
// and calls the superclass's connect() method.
LobbyChatConnection.prototype.connect = function (targetURI,
connectionHandler, userName, password) {
  this.connectionHandler = connectionHandler;
  this.handleCloseEvents = true;
  var result = super.connect(targetURI, userName, password);
  if (result) {
    writeln("Please wait. Attempting connection...");
  }
  else {
    writeln("Can't attempt connection to: " + this.uri);
    writeln("Is the URI correct?");
  }
  return result;
};

// close() turns off the handleCloseEvents flag before closing the
connection.
LobbyChatConnection.prototype.close = function() {
  this.handleCloseEvents = false;
  super.close();
};

/* onStatus() calls the current connectionHandler when a successful
connection
 * is made. It reports closed connections and errors except when closed
 * connections are expected. When a connection is closed, the playhead is
 * sent to the Login frame.
 */
LobbyChatConnection.prototype.onStatus = function(info) {
  // Always deal with successful connections.
  if (info.code == "NetConnection.Connect.Success") {
    this.handleCloseEvents = true;
    this.connectionHandler();
  }
  // Handle messages when the connection is closed.
  if (!this.isConnected && this.handleCloseEvents) {
    // Always handle rejection messages when they occur.
    if (info.code == "NetConnection.Connect.Rejected") {
      writeln(info.application.Message);
      writeln('Did you use the username "Guest" and password "Guest" ?');
    }
    else {
      writeln("Error: Connection Closed.");
    }
    this.handleCloseEvents = false;
    gotoAndPlay("Login");
  }
```

```
  // Handle remote method call errors here if you need to.
};

// Called when a connection has been established to the testLobby app.
function handleLobbyConnection () {
  writeln("Success, you are connected to the lobby!");
  gotoAndPlay("Lobby");
}

// Called when a connection has been established to a chat room.
function handleChatConnection () {
  writeln("Success, you are connected to a chat room!");
  gotoAndPlay("ChatRoom");
}

// Second, create a LobbyChatConnection object.
lobbyChat_nc = new LobbyChatConnection();

// When the Connect button in the Login frames is clicked, this
// function attempts to connect to the testLobby application.
function doConnect() {
  // Keep track of the username and password
  _global.userName = userName_txt.text;
  _global.password = password_txt.text;

  // Try to connect to the lobby.
  lobbyChat_nc.connect("rtmp:/testLobby", handleLobbyConnection,
                       userName, password);
}

// Called when the Lobby button in a chat room is clicked.
function doLobby(btn) {
  lobbyChat_nc.close();
  lobbyChat_nc.connect("rtmp:/testLobby", handleLobbyConnection, userName,
password);
}

// Called when the Chat button in the lobby is clicked.
function doChat(btn) {
  lobbyChat_nc.close();
  lobbyChat_nc.connect("rtmp:/testChat/room1", handleChatConnection,
userName, password);
}

// Called when the Disconnect button is clicked.
function doDisconnect() {
  lobbyChat_nc.close();
  gotoAndPlay("Login");
}

gotoAndPlay("Login");
```

Обработчик onStatus() в примере перемещает ползунок времени в кадр Login. Если необходимо, то универсальный механизм вызова функции может использоваться тем же самым способом, каким была вызвана функция connectionHandler().

Теперь давайте посмотрим, как создать подкласс NetConnection, используя ActionScript 2.0. Класс FCSConnector, определенный в примере 13, является единственным способом, чтобы сделать это. Подобно предыдущим примерам, он использует внутреннее свойство handleCloseEvents для того, чтобы хранить в памяти, должен ли он сообщать программе "NetConnection.Connect.Closed". Он также предназначен для того, чтобы воспользоваться преимуществом передающей события системы, представленной в UI компонентах, представленных во Flash 8. Всякий раз, когда вызван onStatus(), одно из

нескольких возможных событий посылается любым объектам, которые определяют себя как приемники события. События отправляются методом dispatchEvent(). Вы можете заметить в листинге, что метод dispatchEvent() никогда не определяется. В действительности, он добавляется к каждому объекту FCSConnector, когда класс EventDispatcher инициализирует объект в функции FCSConnector().

Пример 14. ActionScript 2.0 для подкласса NetConnection

```
class com.dima.pfcs.FCSConnector extends NetConnection {
  // EventDispatcher needs these.
  var addEventListener:Function;
  var removeEventListener:Function;
  var dispatchEvent:Function;
  var dispatchQueue:Function;

  // Internal data.
  var handleCloseEvents:Boolean = true;

  function FCSConnector() {
    super();
    mx.events.EventDispatcher.initialize(this);
  }

  function connectionClosed (type, info) {
    if (handleCloseEvents) dispatchEvent({target:this, type:type,
info:info});
    handleCloseEvents = false;
  }

  function onStatus (info) {
    switch (info.code) {
      case "NetConnection.Connect.Success":
        dispatchEvent({target:this, type:"onConnect", info:info});
        break;
      case "NetConnection.Connect.Rejected":
        connectionClosed("onReject", info);
        break;
      case "NetConnection.Connect.Closed":
        connectionClosed("onClose", info);
        break;
      case "NetConnection.Connect.Failed":
        connectionClosed("onFail", info);
        break;
      case "NetConnection.Connect.AppShutdown":
        connectionClosed("onClose", info);
        break;
      case "NetConnection.Connect.InvalidApp":
        connectionClosed("onReject", info);
        break;
      case "NetConnection.Call.Failed":
        dispatchEvent({target:this, type:"onCall", info:info});
        break;
    }
  }

  function connect() {
    handleCloseEvents = true;
    return super.connect.apply(super, arguments);
  }

  function close() {
    handleCloseEvents = false;
    super.close();
  }
}
```

Класс FCSConnector не просто создает событие onStatus и передает дальше информационный объект как часть события. Вместо этого метод onStatus() проверяет строку info.code, он принимает и отправляет одно из следующих событий: onConnect, onReject, onFail, onClose или onCall. Создание отдельных событий имеет преимущества и недостатки. Оно разрешает кому-либо использовать класс FCSConnector для получения только тех событий, которые его интересуют, и для написания отдельных функций для обработки каждого типа события. Тем не менее, в некоторых приложениях проще будет обработка одного события onStatus. В следующем примере приведен код для одной демонстрации ролика, который использует класс FCSConnector. Он импортирует FCSConnector, запоминая один его экземпляр в переменную nc, и затем добавляет главную временную шкалу как приемник для всех четырех, связанных с подключением, событий. Каждое событие посылает функции или методу одинаковое имя.

Пример 15. Импортирование и использование класса FCSConnector в главной временной шкале ролика

```
import com.dima.pfcs.FCSConnector;

// Create a new FCSConnector.
nc = new FCSConnector();

// Add the _root timeline as a listener for connection-related events.
// Note: Since this code is on the main timeline this refers to _root.
nc.addEventListener("onReject", this);
nc.addEventListener("onFail", this);
nc.addEventListener("onClose", this);
nc.addEventListener("onConnect", this);

// Handle "NetConnection.Connect.Failed" messages here.
function onFail(ev) {
 var info = ev.info;
 writeln("Can't reach the server.")
 writlen("Please check your network connection and try again.");
 writeln("info.code: " + info.code);
 connectButton.label = "Connect";
}

// Handle "NetConnection.Connect.Success" messages here.
function onConnect(ev) {
 var info = ev.info;
 writeln("You are now connected.");
 writeln("info.code: " + info.code);
 connectButton.label = "Disconnect";
}

// Handle "NetConnection.Connect.Rejected" messages here.
function onReject(ev) {
 var info = ev.info;
 writeln("Your connection attempt was rejected.");
 writeln("info.code: " + info.code);
 writeln("info.description: " + info.description);
 if (info.application) {
  for (var p in info.application) {
   writeln("info.application." + p + ": " + info.application[p]);
  }
 }
 connectButton.label = "Connect";
}

// Handle "NetConnection.Connect.Closed" messages here.
function onClose(ev) {
 var info = ev.info;
 writeln("Your connection was closed.");
```

```
 writeln("info.code: " + info.code);
 connectButton.label = "Connect";
}

connectButton.addEventListener("click", this);

function click(ev) {
 var button = ev.target;
 if (button.label == "Connect") {
  if (nc.connect("rtmp:/testLobby", userNameTextInput.text,
passwordTextInput.text)) {
   button.label = "Wait...";
   writeln("Attempting to connect...");
  }
  else {
   writeln("Can't attempt connection. Check the URI.");
  }
 }
 else {
  writeln("Connection closed.");
  button.label = "Connect";
  nc.close();
 }
}

function writeln(msg) {
 msgTextArea.text += msg + "\n";
 msgTextArea.vPosition = msgTextArea.maxVPosition;
 msgTextArea.redraw();// Fixes scrolling bug in TextArea as of Flash 7.2.
}
```

Компоненты связи без SimpleConnect

Часть, посвященная Communication Components (компонентам связи) продемонстрировала, как могут создаваться приложения при использовании компонент связи Macromedia. Macromedia выпустила компоненты SimpleConnect для того, чтобы управлять сетевым соединением и подключать другие компоненты к нему. SimpleConnect позволяет пользователям входить в систему, используя любое имя, когда они пытаются войти под логином, который уже кем-либо используется. Если вам необходимо разработать приложение, которое управляет пользовательской идентификацией по-разному, но вы хотите использовать компоненты связи Macromedia, есть два варианта. Первый – написание своего собственного компонента подключения. Другой вариант – не использовать компонент подключения вообще, как проиллюстрировано в заключительном примере предыдущей части. Следующий пример использует небольшой серверный скрипт, подкласс NetConnection, и компоненты связи для создания основного приложения чата с отдельными логинами и экранами чата. Приложение присваивает уникальные имена пользователя, не позволяет изменять имена, пока установлено соединение, и не дает возможности “прослушки”. Оно не предназначено для обеспечение lobby и для многочисленных чатов.

Создание приложения на сервере

Для использования компонент связи без SimpleConnect, файл приложения main.asc должен загрузить основу компонента и хранить имя пользователя для каждого клиента, которые подключается без этой основы. Минимальный файл main.asc показан в примере.

Пример 16. Минимальный файл main.asc, когда SimpleConnect не используется

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

application.onConnect = function (client, userName) {
```

```
  gFrameworkFC.getClientGlobals(client).username = userName;
  application.acceptConnection(client);
};
```

Пример позволяет любому человеку подключаться с любым именем пользователя. Чтобы отклонять подключения, в которых имя пользователя пустое или уже используется кем-то, необходимо выполнить работу немного посерьезней. Пример 17 демонстрирует листинг другого файла main.asc. Скрипт использует объект, названный users, для отслеживания userName, связанных с каждым Flash роликом. (Объект, типа users, в котором название элемента используется для того, чтобы обращаться к элементам массива, известен, как associative array, hash table или просто hash). Функция trim() используется для предобрабатывания каждого userName перед проверкой, является ли оно нулевым, пустой строкой или уже в объекте users.

Пример 17. Файл main.asc для приложения netConnectChat

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

// Trim any whitespace from before or after the userName.
// SSAS supports regular expressions, but client-side ActionScript does not.
function trim (str) {
  if (typeof str != "string") return "";  // Make sure str is a string.
  str = str.replace(/^\s*/, "");          // Trim leading spaces.
  str = str.replace(/\s*$/, "");          // Trim trailing spaces.
  str = str.replace(/\n/g,  "");          // Remove new lines.
  str = str.replace(/\r/g,  "");          // Remove carriage returns.
  str = str.replace(/\t/g,  "");          // Remove tabs.
  return str;
}

// Hash of client objects using the userName as a property name.
users = {};

// The onConnect() method rejects connection attempts
// where userName is invalid text or is already in use.
application.onConnect = function (client, userName) {
  userName = trim(userName);   // Remove leading and trailing whitespace.
  if (userName.length == 0) {  // If it is empty, reject it.
    application.rejectConnection(client, {msg: "Empty username."});
    return;
  }
  if (users[userName]) {    // If it is in use already, reject it.
    application.rejectConnection(client,
            {msg: 'The username "' + userName + '" is already in use.'});
    return;
  }
  // Store a reference to the client in the users hash.
  users[userName] = client;
  gFrameworkFC.getClientGlobals(client).username = userName;
  application.acceptConnection(client);
};

// When a client disconnects, remove the username from the
// users hash so someone can use it again.
application.onDisconnect = function (client) {
  var userName = gFrameworkFC.getClientGlobals(client).username;
  delete users[userName];
};
```

Размещение файла main.asc из примера в подкаталог applications, названный netConnectChat, позволяет этому серверному скрипту main.asc управлять приложением netConnectChat. Если вы разрабатывали скрипт main.asc, вы всегда могли проверить его с тестовым клиентом, попробуйте создать клиентскую часть приложения netConnectChat, описанную в следующем разделе.

Формирование клиента

Создайте Flash ролик для подключения нашего нового приложения netConnectChat. Пока Вы будете придерживаться ActionScript 1.0 и компонент v1 UI, от которых зависят компоненты связи Macromedia.

Использование компонент связи Macromedia без SimpleConnect – двух шаговый процесс: установка сетевого подключения к экземпляру приложения и затем передача объекта NetConnection каждому методу компонент connect(). Приложение netConnectChat использует компоненты PeopleList, Chat и UserColor для создания интерфейса простого тестового чата. Весь интерфейс может быть создан на одном кадре, как это популярно при использовании компонента SimpleConnect, или разные состояния ролика могут быть разбросаны по временной шкале. В этом примере Вы будете создавать клиентский ролик для этого приложения, используя временную шкалу. Рисунок 37 показывает временную шкалу и Сцену, когда курсор времени находится на кадре Chat.

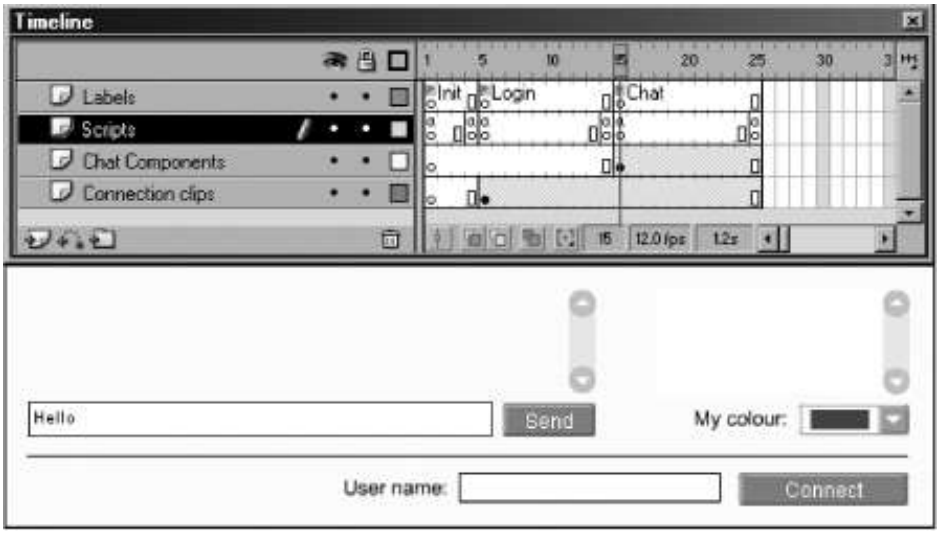

Рисунок 37. Временная шкала и Сцена с ползунком на кадре Chat

Кадр Chat включает в себя следующие видео ролики (компоненты) и текстовое поле:

chat_mc – Экземпляр компонента Chat;
peopleList_mc – Экземпляр компонента PeopleList;
userColor_mc – Экземпляр компонента UserColor;
userName_txt – Текстовое поле внизу Сцены;
connect_btn – Кнопка Connect внизу сцены.

Кадр Login содержит только поле userName_txt и кнопку connect_btn. Если бы Вы использовали компонент SimpleConnect для регистрации пользователя, то любой другой компонент связи должен был бы находиться в тех же самых кадрах временной шкалы, где и SimpleConnect. В этом примере компоненты связи не должны находиться в тех же самых кадрах, где и поле имя пользователя или кнопка Connect. Компонентам необходим только NetConnection, поскольку поле имя пользователя и кнопка Connect оба не должны быть включены в кадр Chat. Тем не менее, в состоянии Chat Вы изменяете значение кнопки Connect на Disconnect. При нажатии кнопки пользователь выходит из системы. Текстовое поле имя пользователя удерживается на Сцене для того, чтобы показать текущее имя пользователя, но оно блокируется во время чата.

Слой Scripts содержит все строчки скрипта для ролика, кроме двух. Три важных кадра – кадр Init, который содержит самый большой скрипт; а также кадры Login и Chat. Когда скрипт в кадре Init полностью завершил выполнение, ползунок перемещается на кадр Login. В кадре Login слоя Scripts есть только одна строка кода:

```
startLoginState();
```

Эта команда вызывает функцию startLoginState(), объявленную в первом кадре временной шкалы, для инициализации элементов на сцене:

```
function startLoginState () {
 connect_btn.setLabel("Connect");
 userName_txt.selectable = true;
}
```

Точно также в кадре Chat есть только вызов функции:

```
startChatState();
```

Функция startChatState(), также объявленная в первом кадре, инициализирует элементы в кадре Chat:

```
function startChatState() {
 connect_btn.setLabel("Disconnect");
 userName_txt.selectable = false;
 peopleList_mc.connect(nc);
 chat_mc.connect(nc);
 userColor_mc.connect(nc);
}
```

Значение на кнопе Connect обрабатывается обеими функциями по мере необходимости, а свойство selectable поля имя пользователя используется для того, чтобы блокировать или разблокировать поле userName_txt.

Когда ползунок (курсор времени) перемещается на кадр с меткой Chat, компоненты связи созданы, потому что это первый кадр, в котором они появляются. Но они также должны быть связаны с объектом NetConnection или с подклассом NetConnection, чтобы функционировать. Для этой цели функция startChatState() передает глобальный объект nc методу connect() каждого компонента.

Остальные скрипты на временной шкале (4, 14 и 24 кадры) содержат только вызов функции stop(). В дополнение к компонентам и видео ролику, расположенным непосредственно на Сцене, Библиотека содержит обозначение видео ролика AlertBox. Он содержит компонент MessageBox, названный alert_mc, из компонент Flash UI комплект 2, и обеспечивает метод отображения всплывающих сообщений. AlertBox временной шкалы видео ролика содержит код для установки заголовка MessageBox и текста сообщения:

```
alert_mc.setTitle("Alert");
alert_mc.setMessage(message);
```

Пример демонстрирует скрипт из главной временной шкалы.

Пример 18. Скрипт на главной шале времени

```
// A simple alert function and variables to pop a message dialog box on
stage.
alertLevel = 10;  // Level on which to display the alert dialog box.
upperLeft  = 10;  // _x and _y position of the alert dialog box.

// alert() displays msg in a pop-up alert dialog box.
function alert (msg) {
  attachMovie("AlertBox", "abox" + alertLevel, alertLevel,
              {_x: upperLeft, _y: upperLeft, message: msg} );
  // Increment the alertLevel and upperLeft position.
  alertLevel += 1;
  upperLeft += 10;
  if (upperLeft > 150) upperLeft = 10;
}

/* ChatConnection is a subclass of NetConnection designed to move the
playhead
 * to the Chat frame when a connection is made and move it back to the Login
 * frame when it is lost.
 */
function ChatConnection() {
  super();
}
```

```
ChatConnection.prototype = new NetConnection();    // Subclass NetConnection.
ChatConnection.prototype.handleCloseEvents = true;   // Initial value.

// connect() turns on the handleCloseEvents flag before calling
super.connect().
ChatConnection.prototype.connect = function() {
  this.handleCloseEvents = true;
  var result = super.connect.apply(super, arguments);
  if (!result) {
    alert("Invalid target URI: " + this.uri);
  }
  return result;
};

// close() turns off the handleCloseEvents flag before closing the
connection.
ChatConnection.prototype.close = function() {
  this.handleCloseEvents = false;
  super.close();
};

/* onStatus() reports closed connections and errors except when closed
 * connections are expected. When a connection is closed, the playhead is
 * sent to the Login frame. When it is opened, it is sent to the Chat frame.
 */
ChatConnection.prototype.onStatus = function (info) {
  if (info.code == "NetConnection.Connect.Success") {
    gotoAndPlay("Chat");
  }
  else if (!this.isConnected) {
    if (this.handleCloseEvents) {
      var msg;
      if (info.code == "NetConnection.Connect.Rejected") {
         msg = 'Connection Rejected!';
         if (info.application) {
           msg += '\n' + info.application.msg;
        }
      }
      else {
        msg = 'Error: Connection ' + info.code.split(".").pop();
      }
      alert(msg);
      this.handleCloseEvents = false;
      gotoAndPlay("Login");
    }
  }
};

// Main timeline button handler functions.

/* doConnect() is called whenever connect_btn is clicked. If the button label
 * is Connect, the NetConnection.connect() method is called. If the label
 * is Disconnect, the close() method is called.
 */
function doConnect(btn) {
  if (btn.getLabel() == "Connect") {
    if (!nc.isConnected) {
      nc.connect("rtmp:/netConnectChat", userName_txt.text);
      btn.setLabel("Waiting...");
    }
  }
  else if (btn.getLabel() == "Disconnect") {
    if (nc.isConnected) {
      userColor_mc.close();
```

```
      peopleList_mc.close();
      chat_mc.close();
      nc.close();
      gotoAndPlay("Login");
    }
  }
}
// Main timeline state change functions.

// Called after the Chat frame is entered to connect communication components
// and change the appearance or behavior of other Flash components.
function startChatState() {
  connect_btn.setLabel("Disconnect");
  userName_txt.selectable = false;
  peopleList_mc.connect(nc);
  chat_mc.connect(nc);
  userColor_mc.connect(nc);
}

// Called after the Login frame is entered to reset login components.
function startLoginState() {
  connect_btn.setLabel("Connect");
  userName_txt.selectable = true;
}

// Create the chat connection object.
_global.nc = new ChatConnection();

gotoAndPlay("Login");
```

Если вы тестируете ролик в нескольких окнах браузера, вы увидите, что вы сможете войти в систему, только если указанное имя пользователя уникальное.

Кроме этого обстоятельства, метод alert() заменен на writeln(). Одно отличие находится в обработчике onStatus(), который сообщает об ошибках немного по-другому:

```
msg = 'Error: Connection ' + info.code.split(".").pop();
```

Свойство info.code всегда содержит разграниченную точкой строку. Предыдущая команда использует метод split() для того, чтобы разбить строку на массив и затем метод pop() вытаскивет данные из массива для того, чтобы вернуть последнюю часть массива. Если info.code возвращает сообщение, типа "NetConnection.Connect.Failed", то предыдущая команда добавляет в конец к "Error: Connection" строку "Failed". Когда свойство code ненулевое, первый элемент в строке всегда класс объекта, с которым связано событие. Второй элемент – обычно название предпринятого действия, а третий – результат. Таблица 5 демонстрирует все значения code и level информационного объекта NetConnection, что касается FlashCom 1.5.

Таблица 5. Значения NetConnection'а code и level

Code	Level	Значение
NetConnection.Call.Failed	Error	На этом подключении был предпринят и не достиг успеха удаленный вызов метода. Свойство info.description содержит больше подробностей, включая название метода.
NetConnection.Connect.AppShutdown	Error	Это сообщение предполагается для того, чтобы быть возвращенным, когда экземпляр приложения вынужден закрыть пример, когда он за пределами памяти. Однако, что касается FlashCom 1.5, это сообщение никогда не бывает получено; вместо него приходит "NetConnection.Connect.Closed".
NetConnection.Connect.Closed	Status	Подключение было закрыто сервером или

Code	Level	Значение
		клиентом. Что касается FlashCom 1.5, несмотря на значение level "status", оно может означать ошибку, такую, как закрытие экземпляра приложения, потому что оно использует слишком много памяти.
NetConnection.Connect.Failed	Error	Попытка подключения не увенчалась успехом. Сервер не может быть достигнут или не работает.
NetConnection.Connect.InvalidApp	Error	Это сообщение предполагается для того, чтобы быть возвращенным, когда название приложения не зарегистрировано на сервере. Тем не менее, что касается FlashCom 1.5, это сообщение никогда не бывает получено; вместо него приходит "NetConnection.Connect.Closed".
NetConnection.Connect.Rejected	Error	Попытка подключения была отклонена сервером или приложением на сервере. Причина отклонения клиента зависит от нескольких факторов. Например, приложение может отклонить соединение, потому что пользовательские полномочия недействительны. Приложение может вернуть объект application как свойство информационного объекта. Свойство info.description будет содержать полезную информацию, если имя приложения не было найдено, если предельные ресурсы приложения были исчерпаны, или если лицензия сервера не позволяет больше подключений или использования большей пропускной способности.
NetConnection.Connect.Success	Status	Соединение было установлено.

Свойство info.description часто пустое, но для некоторых сообщений содержит важную дополнительную информацию. Когда сервер возвращает сообщение "NetConnection.Connect.Rejected", или сервер, или приложение отклонили подключение. Когда приложение отклоняет подключение, серверный скрипт имеет опцию для возвращения объекта application, чтобы объяснить причину отклонения соединения. Когда сервер отклоняет подключение, он возвращает информацию в свойстве description, которое также объясняет причину отклонения. Сообщение описания отклонения выглядит следующим образом:

```
[ Server.Reject ] : (_defaultRoot_, _defaultVHost_) : Application (appName)
 is not defined.
```

Таблица 6 перечисляет три типа сообщений сервера об отклонении, включенных в свойство description.

Таблица 6. Значения Description для сообщений NetConnection.Connect.Rejected

Сообщение	Смысл
Server.Reject	Сервер отклонил подключение, потому что приложения не существовало, из-за ошибки конфигурации или из-за некоторых других проблем, таких как дефектные сетевые данные.
Resource.Limit.Exceeded	Предельные ресурсы для сервера, виртуального хоста или приложения были исчерпаны. Например, максимально разрешенное количество экземпляров или совместно используемых объектов было превышено.

Сообщение	Смысл
License.Limit.Exceeded	Количество одновременных клиентских подключений или предел пропускной способности были превышены. См. информацию в Предисловии для лицензионного FlashCom'а.

Заключение

Не стоит недооценивать важность овладения искусством управления подключениями. Примеры, изученные в этой части, должны помочь вам и в разработке, и в отладке ваших приложений и их подключений. Теперь, когда Вы сделали несколько клиентских скриптов, пришло время переходить к созданию скриптов на сервере.

Лабораторная работа №4: Приложения, экземпляры и серверные сценарии

Приложение FlashCom может быть написано на сервере, используя Server-Side ActionScrip (SSAS) и доступные ему серверные объекты. С помощью SSAS можно контролировать, у кого есть доступ к приложениям и какие ресурсы доступны каждому пользователю. Можно контролировать удаленные Flash ролики, переключатся между потоками, объединять потоки между FlashCom серверами, получать доступ и обновлять базу данных.

Server-Side ActionScript несколько отличается от Client-Side ActionScript, использующийся ранее для написания Flash роликов. В этой части описывается, как написать Server-Side ActionScript и работать с объектами, которые контролируют экземпляры приложений и Flash ролики, соединенные с ними. Написание сценариев на сервере также важно, как и на Flash. Эта часть ознакомит вас с тем, что происходит на сервере.

Написание экземпляров приложений

На одном FlashCom сервере может находиться несколько различных приложений. У каждого приложения есть свой уникальный тип объекта, связанный со своим сценарием. Сценарий создается путем добавления подкаталога в каталог **applications** на сервере. Внешенее окружение обращается к каждому подкаталогу **applications** как к каталогу **registered application** который должен точно указывать на приложение FlashCom. В свою очередь сервер создает экземпляры приложений, доступные Flash роликам, которые пытаются подключиться к ним. Каждый каталог **registered application** можно назвать домашним каталогом приложения. Чтобы написать приложение необходимо добавить *main.asc* файл в домашний каталог приложения. Например, чтобы создать приложение courseLobby, в каталог **applications** надо добавить подкаталог **courseLobby**. Файл *main.asc* в каталоге **courseLobby** дает приложению уникальный тип. Файлы Server-Side ActionScript-а, такие как *main.asc*, являются текстовыми файлами, содержащими исходный код. Они могут быть созданы с помощью простого текстового редактора, включенного во Flash MX Professional 2004 (Flash Pro). Исходные файлы SSAS почти всегда имеют расширение *.asc* (хотя *.js* тоже возможно), на самом деле SSAS очень походит на JavaScript 1.5.

Если необходимы двухбайтовые символы, подобные Kanji, следует использовать текстовый редактор, поддерживающий кодировку UTF-8. К тому же метод имен двухбайтовых символов высшего порядка должен использовать оператор array вместо dot. Например:

```
obj = {};
obj.myMethodName = function() {
```

```
  trace("myMethodName");
};
obj["myMethodName"](); // Correct
obj.myMethodName();    // Correct

obj["myMethodName "] = function() {
  trace("myMethodName ");
};
obj["myMethodName"]();    // Correct
obj.myMethodName();       // Syntax Error
```

Экземпляры и источники

На FlashCom сервере могут выполняться одновременно самые разные экземпляры приложений. Для каждого экземпляра выделяется память, место на диске, поток и общее окружение объектов, а также запускается однопоточная копия файла *main.asc*. Экземпляры нужны для объединения пользователей и разделения ресурсов сервера между ними. Самым простым примером является – чат. Экземпляры чата могут запускаться одновременно, каждый для разных пользователей, создавая набор чат-комнат. Пользователи, которые соединились с *rtmp:/courseChat/room1* могут связаться друг с другом, используя потоки и общие объекты, связанные с экземпляром **room1**. Другие пользователи, присоединившиеся к *rtmp:/courseChat/room2* могут общаться друг с другом, используя экземпляр **room2**, не подозревая, о чем говорят в **room1**. Если экземплярам необходимо обменяться потоками или объектами, один экземпляр соединяется с другим по сети, чтобы получить доступ к его ресурсам.

Можно создать приложение, в котором один экземпляр одновременно обеспечивает работу “коридора” и множества комнат чата, делать это не рекомендуется. Если много пользователей одновременно будут использовать один экземпляр, производительность снизится. Так как ActionScript экземпляра работает в один поток, каждая удаленная функция, обращающаяся к серверу, включая доставку сообщений, будет выполняться последовательно

Сложно точно ограничить максимальное количество клиентов для одного экземпляра, потому что каждое приложение предъявляет различные требования к серверу. Экземпляр, который интенсивно использует Server-Side ActionScript, может поддерживать приемлимую работоспособность менее, чем для 20 клиентов, в то время как другие приложения работают с сотнями или даже тысячами.

Потоки и общие объекты, доступные для каждого экземпляра, размещаются FlashCom, используя соответствующий URI. Чаще всего это имя потока или общего объекта. Например, если Flash ролик проигрывает поток intro, тогда “intro” является соответствующим URI для экземпляра, с которым соединен ролик. Если URI экземпляра:

```
rtmp://my.university.edu/courseLectures/algebra101
```

тогда полный URI для ролика intro будет:

```
rtmp://my.university.edu/courseLectures/algebra101/intro
```

На практике такой URI никогда не используется, но желательно знать, что потоки и ресурсы взаимодействующих объектов существуют в области, определяемой URI. После попытки соединения Flash ролик запрашивает ресурсы экземпляра приложения, используя соответствующий URI. Например, ролик должен сначала запросить соединение с экземпляром прежде, чем пытаться опубликовать или проиграть поток, или получить удаленный объект:

```
nc.connect("rtmp://my.university.edu/courseLectures/algebra101");
```

После предпринятой попытки соединения Flash ролик может запросить, чтобы записанный или передающийся поток intro был воспроизведен:

```
ns = new NetStream(nc);  // Create a NetStream within a NetConnection.
videoArea.attachVideo(ns);//Attach the stream to a video object on the Stage.
ns.play("intro");     // Play a stream named intro.
```

Подобным образом, совместно использующийся объект streamList, содержащий список названий потоков, может быть доступен экземпляру algebra101. После того, как был сделан запрос на соединение с экземпляром algebra101, доступ к объекту streamList запрашивается следующим образом:

```
so = SharedObject.getRemote("streamList", nc.uri, true);
so.onSync = function (list) {
 trace("onSync> list.length: " + list.length);
};
so.connect(nc);
```

В предыдущих двух примерах имя потока и совместно использующийся объект - всего лишь родственные URI, располагаемые на одном FlashCom сервере. Соответствующий путь к потоку или совместно использующемуся объекту внутри экземпляра может включать систему имен по принципу каталога, что помогает разделять ресурсы по группам. Например, лекции могут быть сгруппированы по предмету так, что соответствующий URI для видео потока может быть vectors/intro в то время как другое видео располагается в matrices/intro. Метод play() клиента, обращающийся к одному из приведенных примеров будет выглядеть следующим образом:

```
ns.play("vectors/intro");
```

Совместно использующийся объект, доступный через соответствующий URI vectors/quizQuestions, будет создан следующим образом:

```
so = SharedObject.getRemote("vectors/quizQuestions", nc.uri, true);
```

Разрабатывая приложение можно нарисовать простое дерево каталогов, показывающее элементы экземпляра. На рисунке 38 показано расположение и соответствующие URI ресурсов для экземпляра algebra101 приложения courseLectures.

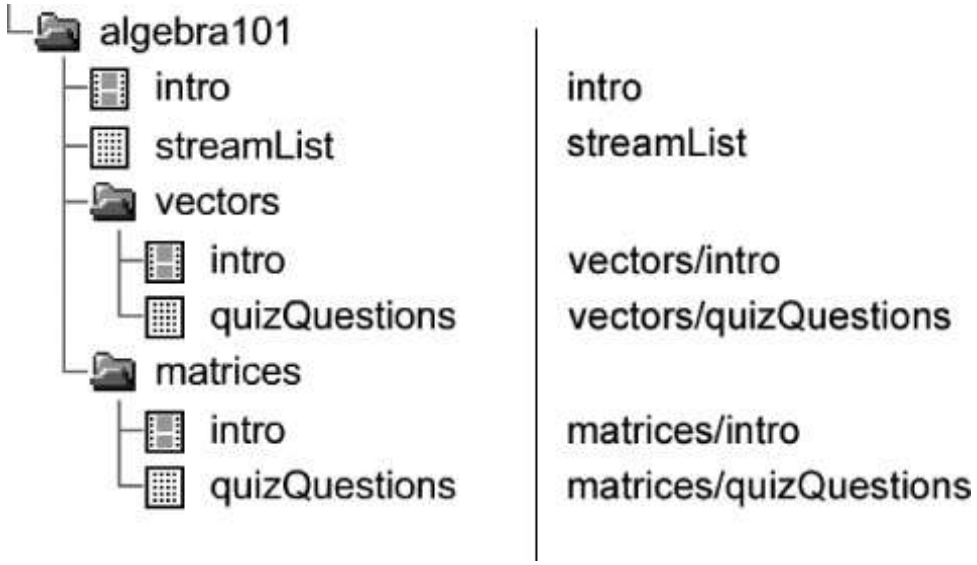

Рисунок 38. URI, соответствующие ресурсам экземпляра algebra101

Все записанные потоки и постоянные файлы объекта обычно хранятся в директориях **streams** и **sharedobjects** домашнего каталога приложения. Два каталога организованы иерархически, согласованно с соответствующими URI, ссылающихся на элементы экземпляра. Если потоки и совместно использующиеся объекты в этом примере хранятся на сервере, их можно найти в структуре каталогов на рисунке 39.

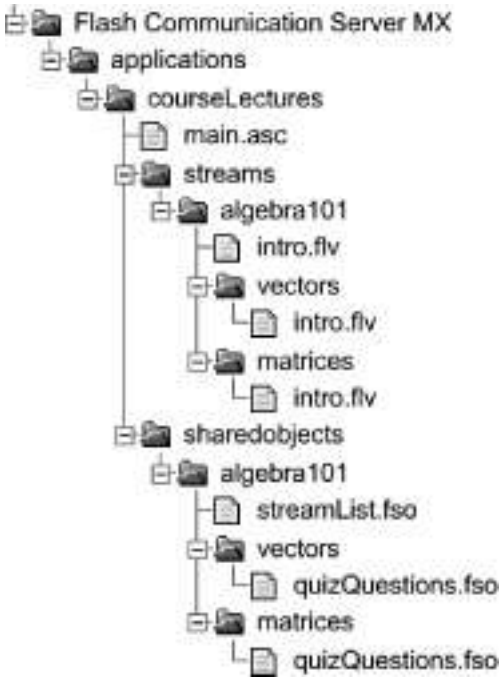

Рисунок 39. Расположение файлов общих объектов и потоков

Экземпляр с именем **algebra101** является родительским каталогом для каталогов **stream** и **sharedobjects**. В каталогах экземпляров таких, как **algebra101** расположены каталоги и файлы, которым соответствуют URI, показанные на рисунке 38. Например, путь к потоку, записанному в URI vectors/intro экземпляра **algebra101**, будет выглядеть следующим образом:

```
.../applications/courseLectures/streams/algebra101/vectors/intro.flv
```

Потоки могут использоваться всеми экземплярами приложения, образуя виртуальный каталог.

Конфликты имен ресурсов

Теоретически каждый экземпляр управляет своими собственными ресурсами: потоками и совместно использующимися объектами без вмешательства других экземпляров. Тем не менее, существует такой случай, когда имена потоков и совместно использующихся объектов экземпляра конфликтуют (т.е. два разных экземпляра стремятся использовать один и тот же ресурс); конечно же, этих конфликтов можно избежать. Используя предыдущий пример приложения courseLectures, представьте, что один клиент соединяется со следующим URI:

```
rtmp:/courseLectures/algebra101
```

а второй соединяется с другим URI:

```
rtmp:/courseLectures/algebra101/vectors
```

Это запустит два экземпляра приложения courseLectures: один экземпляр с именем **algebra101**, а другой с именем **algebra101/vectors**. Если первый экземпляр попытается обратиться к совместно использующемуся объекту с URI vectors/quizQuestions, а второй – к объекту с URI quizQuestions, они будут использовать один и тот же объект quizQuestions с URI:

```
rtmp:/courseLectures/algebra101/vectors/quizQuestions
```

FlashCom не способен решить такую ситуацию.

Два экземпляра никогда не должны напрямую пытаться обновить один и тот же совместно использующийся объект. Если более, чем одному экземпляру необходимо использовать совместно использующийся объект, это должно быть сделано через NetConnection между экземплярами.

Обычно один экземпляр выбирается как владелец совместно использующегося объекта или потока, а другой – соединяется с ним для доступа к его ресурсам.

Различия между Flash ActionScript и Server-Side ActionScript

Как упоминалось ранее, существует два различных вида ActionScript: client-side (клиентский) ActionScript (CSAS), который включает в себя как ActionScript 1.0 и ActionScript 2.0, и версия, которая выполняется на сервере – Server-Side (серверный) ActionScript (SSAS). Клиентский ActionScript используется для создания *.swf* клипов, которые воспроизводятся Flash плеером на компьютере пользователя. Server-Side ActionScript хранится в файлах с расширением *.asc*, которые запускаются на FlashCom сервере.

Начиная с Flash 5, клиентский ActionScript был основан на стандартах ECMA (Европейская Ассоциация производителей компьютеров), как и JavaScript. Стандартный язык назван ECMAScript и описывается в стандарте ECMA-262, доступном на сайте: *http://www.ecma-international.org*. Клиентский ActionScript никогда полностью не соответствовал стандарту ECMAScript, потому что Macromedia разработала язык под свои потребности, с целью сохранения обратной совместимости. Клиентский ActionScript также со временем эволюционировал. Ранние версии, поддерживаемые во Flash MX (и до сих пор поддерживаемые во Flash 8) отличаются от стандарта. Последние версии, поддерживаемые во Flash MX 2004 и более близкие к ECMA стандартам, чем ранние версии, называются ActionScript 2.0. Последняя версия ActionScript во Flash 8 снова сильно отличается от стандарта ECMA. Если вы хотите знать больше о различиях между Flash ActionScript 1.0 и ECMAScript, смотрите приложение D в ActionScript для Flash MX.

В отличие от клиентского ActionScript, серверный ActionScript соответствует стандартам ECMAScript, потому что Macromedia использовала интерпретатор Mozilla Spidermonkey JavaScript 1.5 во FlashCom. Дополнительная информация по этой теме и JavaScript 1.5 доступна на сайте:
`http://www.mozilla.org/js`

Этот раздел подчеркивает различия между клиенским ActionScript и серверным ActionScript, на которые следует обратить внимание.

Чувствительность к регистру

Server-Side ActionScript всегда чувствителен к регистру клавиатуры, в то время как клиентский ActionScript нет. Во Flash 5 и Flash MX большинство ActionScript'ов к регистру не чувствительны. Например, в Flash MX myVariable и MyVariable ссылаются на одну переменную. Так не происходит в серверных сценариях.

Многих разработчиков ставит в тупик принцип чувствительности к регистру в Flash MX 2004, который отчасти зависит от версии Flash (формата SWF) и версии ActionScript (версии компилятора), которую можно выбрать из File → Publish Settings → Flash. Определим для начала, что мы говорим о двух разных видах чувствительности в клиентском ActionScript: чувствительность к регистру в процессе компиляции и чувствительность во время прогона программы. Компилятор ActionScript 1.0 не чувствителен к регистру в отличие от ActionScript 2.0. Однако только версия выбранного SWF формата влияет на чувствительность к регистру во время прогона программы, а не версия ActionScript, используемая во время компиляции и не версия Flash плеера, в котором воспроизводится файл.

Чувствительность к регистру во время прогона программы сведена в таблице, по книге Colin Moock Essential ActionScript 2.0. Необходимо отметить, что ActionScript 1.0 и 2.0 иногда чувствительны к регистру при экспортировании Flash. Другие исключения чувствительности во время проигрывания приведены в пояснениях к таблице 7.

Таблица 7. Чувствительность к регистру языка, на котором написана программа, формата файла и версии Flash Player во время прогона программы

Ролик компилированный ActionScript 1.0 или 2.0	Воспроизведение в Flash Player 6	Воспроизведение в Flash Player 7
Flash Player 6- файлы с форматом .swf	Не чувствителен - a	Не чувствителен - a
Flash Player 7- файлы с форматом .swf	Не поддерживает - b	Чувствителен

Идентификаторы (т.е. переменные и собственные имена), названия функций, метки кадров и экспортируемые ID чувствительны в Flash Player 6-формат файла *.swf*. Но зарезервированные слова, такие как **if**, чувствительны к регистру даже в Flash Player 6.

Flash Player 6 не может воспроизводить файлы из Flash Player 7-файлы с форматом *.swf*.

Не смотря на то, пишете ли вы в клиентском ил серверном ActionScript, чтобы избежать лишних ошибок, рекомендуется использовать верхний и нижний регистры согласованно в любом коде.

Если вы осторожно используете заглавные буквы в именах или никогда не используете заглавные буквы во избежание различий между переменными или другими идентификаторами, у Вас никогда не возникнет каких-либо проблем. Как бы то ни было, в JavaScript практикуется использование заглавных букв для различия имен конструкторов и экземпляров класса. Например:

```
function User() {
}
// The user variable holds an instance of the User class
user = new User();
```

Во Flash 5 и Flash MX такое написание вызовет ошибку исполнения конструктора, но в SSAS различие между написанием User (класс) и user (переменная, хранящая экземпляр класса) состоит в их разной сущности. Чтобы избежать путаницы, чаще всего добавляют слово “Class” в название конструктора:

```
function UserClass() {
}
user = new UserClass();
```

Наследование

Серверный ActionScript не поддерживает формального определения классов и интерфейсов, как в клиентском (client-side) ActionScript 2.0. SSAS поддерживает прототип наследования, который несколько отличается от Flash ActionScript 1.0. Flash MX добавил оператор super в client-side ActionScript, чтобы конструктор и функции класса super было легче вызвать. Не смотря на то, что оператор super не является частью языка ECMAScript 3, определенного стандартом ECMA-262, он рассматривается для следующего стандарта. Оператор super не доступен в SSAS. Однако в некоторых случаях можно разрешить доступ подклассу к конструктору и функциям super класса в SSAS. Приведем пример, как можно вызвать конструктор или функции super класса в SSAS.

Пример 19. Вызов конструктора и функций класса super в SSAS

```
SuperClass = function(a) {
  this.a = a;
};

SuperClass.prototype.method = function() {
  trace("SuperClass method called. a is: " + this.a);
};

SubClass = function(a, b) {
  SuperClass.apply(this, arguments);
  this.b = b;
};
```

```
SubClass.prototype = new SuperClass();
SubClass.constructor = SubClass;

SubClass.prototype.method = function() {
  trace("In Subclass method. a and b: " + this.a + ", " + this.b);
  SuperClass.prototype.method.apply(this, arguments);
};

subClass = new SubClass(1, 2);
subClass.method();
```

Объектно-ориентированное программирование и прототипное наследование, использующее JavaScript, подробно описано в JavaScript: Полное руководство, Четвертое издание (O'Reilly) David Flanagan. Эта книга особенно полезна для написания серверных сценариев, так как она описывает JavaScript 1.5 – точно такой же язык, как и SSAS.

Подробно о прототипном наследовании в Flash 5 и Flash MX, а также о различных вариантах применения наследования без использования оператора super в Flash можно узнать на сайте:

`http://www.quantumwave.com/flash/inheritance.html`

Это также ценный ресурс о написании серверных сценариев.

Многие из доступных объектов не могут стать подклассом, используя методы родительского класса. Например, общие объекты не создаются в результате вызова конструктора. Вместо этого они создаются с помощью статической функции SharedObject.getRemote() на стороне клиента или get() на сервере. Часто бывает, что проще работать с классами, чем наследовать от них свойства и методы.

Однократное выполнение контекста

В Client-side ActionScript предусмотрен специальный объект _global, в котором хранятся глобальные переменные, которые существуют вне области временной шкалы. Например:

```
// Assume a _global object already exists.
_global.x = 3; // OK in client-side AS; however, error in SSAS.
trace (x);   // Will output: 3 in Flash MX and later.
```

Если переменная в client-side ActionScript не найдена на временной шкале, а у объекта _global такое же имя, то можно обратиться к объекту _global.

В SSAS нет временной шкалы. Весь код на сервере выполняется в одном глобальном контексте, который связан с каждым экземпляром приложения. Поэтому нет смысла вводить объект _global. Можно создать объект и назвать его _global, но он не будет работать также как в client-side ActionScript. Вообще следует избегать создания объектов с таким именем в SSAS, потому что это может вызвать некоторые трудности при использовании Flash Remoting.

Когда все необходимое загружено в файл *main.asc*, файл *netservices.asc* создает новый объект _global и задает его свойства. Файл RecordSet.asc также задает свойства объекту _global. Кроме того *netservices.asc* определяет методы unshift() и registerClass() класса Object; так как все классы наследуются от Object, каждый объект отражает эти методы, когда его свойства пронумерованы в цикле for-in. Если надо использовать объект _global в SSAS, необходимо проверить, существует ли он, прежде чем создавать новый и избегать конфликтов со свойствами _global, определенными в *netservices.asc*.

Доступ к неопределенным переменным (undefined)

Часто в client-side ActionScript можно видеть проверку на неопределенные переменные:

```
if (xyz == undefined) {    // Don't do this on the server!
 trace("xyz is undefined.");
}
```

Однако выполнение предыдущего кода вызовет ошибку ссылки на сервер, потому что переменной xyz не существует. Чтобы это обойти, можно воспользоваться оператором typeof:

```
if (typeof xyz == "undefined") {
 trace("xyz does not exist");
}
```

Использование typeof необязательно в теле функции, где переменная является локальной, так как была объявлена или передана через параметры функции. Это будет правильно как в SSAS, так и в client-side ActionScript:

```
var xyz;
if (xyz == undefined) {
 trace("xyz is undefined.");
}
```

Также как это:

```
someFunction = function (xyz) {
 if (xyz == undefined) {
  trace("xyz is undefined.");
 }
};
```

Операторы try/catch/finally

Если вы просматривали код взаимодействующих друг с другом компонентов, выполняемых на сервере в папке scriptlib/components, вы видели, как часто используются операторы try/catch. Например, компонент FCChat на сервере будет определен только один раз с помощью catch блока:

```
try {var dummy = FCChat;} catch (e) { // #ifndef FCChat
 // The FCChat class is defined here...
}
```

Операторы try/catch/finally поддерживаются client-side ActionScript 2.0, но AS 2.0 не поддерживает проверки на тип и ссылку в процессе выполнения сценария. Подробнее о try/catch/finally в ActionScript 2.0 можно узнать в главе по обработке исключительных ситуаций в книге Essential ActionScript 2.0. Для описания использования try, catch, finally, throw, и объектов Error в JavaScript (которые применяются в SSAS), читайте JavaScript: Полное руководство, Четвертое издание.

Использование try/catch предпочтительнее события ошибка (ошибка типа и ссылки), так как позволяют перехватить и обработать эти ошибки, которые в противном случае остановят выполнение сценариев на сервере. Вы уже рассматривали пример ошибки ссылки (ошибка, вызванная попыткой обратиться к несуществующей переменной). Ошибка типа возникает при попытке получить доступ к свойствам неопределенного или нулевого объекта. Следующий небольшой сценарий SSAS прерывает свое выполнение, когда возникает ошибка следуещего типа. Если запустить его, используя App Inspector (тестирующая программа представленная в части, касаемой администрирования и настройки сервера и использовавшаяся далее в той же части в рассказе об отладке приложений), можно увидеть сообщение об ошибке “status has no properties” и последующая функция trace() никогда не будет правильно выполнена:

```
status = null;
trace(status.property); // Causes a type error
trace("Unreachable trace statement.");
```

С другой стороны, заключив тот же код в оператор try позволяет избежать ошибки:

```
try {
 status = null;
 trace(status.property);
 trace("Unreachable trace statement.");
}
catch (e) {
 trace("Error name and message: " + e.name + ", " + e.message);
}
```

```
finally {
 trace("No matter what,this statement will be executed.");
}
trace("Processing can continue here...");
```

Если этот простой сценарий запустить в App Inspector, появится следующее сообщение:

```
Error name and message: TypeError, status has no properties
No matter what, this statement will be executed.
Processing can continue here...
```

Несмотря на то, что в ActionScript 2.0 есть операторы try/catch/finally, он не поддерживает проверку на тип и ссылку в процессе выполнения программы. Этот же сценарий, запущенный в Flash MX 2004 выведет следующее сообщение, потому что `trace(status.property);` выдает значение “ undefined ”, а не вызывает ошибку:

```
undefined
Unreachable trace statement.
No matter what, this statement will be executed.
```

#include и import в сравнении с load()

Server-Side ActionScript использует функцию load() вместо директивы #include (общепринятой в client-side ActionScript 1.0), чтобы загрузить другие исходные файлы ActionScript в сценарий. Оператор import не поддерживается в SSAS и в них отсутствует понятие библиотек класса, собранных в группы как ActionScript 2.0. Читайте Essential ActionScript 2.0 для подробного описания классов и групп client-side ActionScript.

Работа экземпляров приложения

Экземпляр приложения – процесс, запускаемый на FlashCom сервере. У каждого экземпляра есть:

- Собственная машина сценариев;
- Пространство имен потоков и совместно используемых объектов;
- Данные длительного хранения в пространстве своих имен.

Два экземпляра не могут совместно использовать и обновлять одни и те же данные, но они могут воспроизводить тот же самый поток в потоке виртуального каталога.

FlashCom включает в себя накопители процессов для обработки сценариев, и каждое приложение использует один процесс из этого накопителя для выполнения серверного сценария, в ответ на такие события как загрузка экземпляра или подключение клиента Flash к экземпляру. Серверные сценарии должны быть разработаны для быстрой обработки событий, а затем освобождают используемый процесс. Когда в одном из приложений появится ошибка, это не повлияет на другие экземпляры.

На FlashCom может запускаться несколько экземпляров одного приложения. Каждый экземпляр обычно создается FlashCom в ответ на запрос соединения со стороны клиента. FlashCom получает запрос клиента на соединение и проверяет переданный ему RMTP-адрес. RMTP-адрес, направленный клиентом – URI, переданный в метод NetConnection.connect(). FlashCom извлекает имя приложения, а затем запускает экземпляр этого приложения. Если в RMTP такого имени не было найдено, создается экземпляр с именем _definst_. Иначе создается экземпляр, названный в URI. В зависимости от того, как приложение было создано и как был сконфигурирован сервер спустя некоторое время после отсоединения от него последнего клиента, экземпляр будет уничтожен. Между временем создания и уничтожения сценария может произойти много событий. Чтобы упростить этот процесс, разделим его на три части: запуск, середина жизни и остановка.

Запуск

Экземпляр обычно запускается FlashCom'ом, когда первый клиент пытается к нему подсоединиться. Однако его можно запустить и при помощи App Inspector или Административной консоли, или экземпляр приложения может запускаться сразу после запуска сервера, если в файле *Application.xml* установить тег <LoadOnStartup>.

Необходимо отметить, что существует только два ключевых объекта, имеющих отношение к экземпляру и работе с клиентом. Объект application – одноэлементный объект, который представляет экземпляр приложения. У каждого экземпляра есть свой собственный объект application, который является экземпляром класса Application. Как показано в примере, следует добавить динамические методы непосредственно в экземпляр application вместо Application.prototype. Например:

```
application.onConnect = function (client, userName, password) {};
```

Объект Client представляет клиентский Flash ролик, подключающийся к экземпляру приложения. Для каждого объекта application существует более одного объекта Client (один для каждого пользователя). Методы и свойства классов Application и Client описаны в Словаре Macromedia's Server-Side Communication ActionScript, доступном на сайте:

```
http://download.macromedia.com/pub/flashcom/documentation/FlashCom_SS_ASD.pdf
```

Стадии запуска:

- Загружается, компилируется и исполняется файл *main.asc*.
- Загружаются и исполнятся все файлы, использующие метод load().

Вызывается метод application.onAppStart(). Метод onAppStart() вызывается только один раз и чаще используется для инициализации объекта application и совместно используемых объектов и потоков экземпляра.

Если экземпляр был запущен в результате запроса клиента на подключение, используя NetConnection.connect(), вызывается метод application.onConnect() который передает объект Client, представляющий собой подключаемый Flash ролик клиента. Любые дополнительные параметры, переданные в методе connect() также передаются в метод onConnect(), где запрос клиента на подключение может быть отклонен, принят или находиться в процессе ожидания.

Середина жизни

Как только экземпляр приложения был запущен, любое количество клиентов может подключиться или отключиться от него. Всякий раз, при попытке Flash ролика подключиться, вызывается метод application.onConnect() и передается объект Client, представляющий собой Flash ролик клиента. Объекты Client с разрешенным доступом хранятся в массиве application.clients. Когда клиент отсоединяется, вызывается метод application.onDisconnect() и передается объект Client, представляющий отсоединившегося клиента. Клиент может отсоединиться из-за проблем в сети или неполадок на удаленном компьютере. FlashCom может не сразу определить, что клиент отсоединился. Поэтому какое-то время объект Client не будет представлять подключенный Flash ролик.

Остановка

Экземпляр может быть выгружен сервером из-за слишком большой загрузки памяти или остановлен вручную, используя Admin Service. Исключая такие события, экземпляр обычно прекращает работу спустя некоторое время после отсоединения последнего пользователя. Задержка между отсоединением последнего пользователя и прекращением работы приложения предназначена для того, чтобы минимизировать число запусков экземпляра. Эта задержка управляется тремя тегами в трех типах конфигурационных файлов XML, как описано в таблице 8.

Таблица 8. Теги остановки экземпляра приложения

Название тега	Значение по умолчанию	Расположение	Описание
<ApplicationGC>	5	Server.xml	Интервал, в минутах, надо ли приложению проверять экземпляры на уничтожение.
<AppInstanceGC>	20	Vhost.xml	Интервал, в минутах, проверка экземпляров приложения на уничтожение.
<MaxAppIdleTime>	1200	Application.xml	Несколько секунд начиная с того момента, как последний клиент отсоединился перед уничтожением экземпляра.

Значения по умолчанию значат, что каждые 5 минут у приложения запрашивают, прошло ли 20 минут с того момента, как они пытались удалить свои экземпляры. Если прошло 20 минут, а последний клиент отсоединился больше, чем 1200 секунд назад (20 минут), экземпляр будет уничтожен. Если сервер не очень занят, значения по умолчанию приведут к тому, что экземпляры будут уничтожены между 20 и 25 минутами после отсоединения последнего клиента. Это может напоминать длинную задержку, но это разумный компромисс между уничтожением бесполезно работающего экземпляра и дополнительных: остановки и перезагрузки экземпляра при попытке подключения другого клиента.

Прежде чем окончательно завершить работу экземпляра, вызывается метод application.onAppStop(). Если функция onAppStop() возвращает false, экземпляр не завершает работу. Метод onAppStop() обычно используется для проверки элементов объекта, которые в дальнейшем могут потребоваться, прежде чем экземпляр будет удален.

Тестовый запуск простого сценария

Типичная тестовая программа helloWorld – самая маленькая, выводящая результат на экран. Не смотря на то, что серверные программы обычно не выводят результаты пользователю, простая программа helloWorld до сих пор очень полезна. Функция trace() в SSAS выводит текстовые сообщения для NetConnection Debugger, App Inspector, и системных журналов на сервере. Во время разработки App Inspector – главный инструмент, позволяющий разработчикам загружать, выгружать и перезагружать приложения после внесения изменений в сценарий, должен быть протестирован.

В примере показан небольшой main.asc сценарий, реализация программы helloWorld в SSAS. В нем демонстрируются стандартные методы обработчика событий объекта application.

Пример 20. Тестовый сценарий helloWorld

```
application.onAppStart = function() {
  trace("onAppStart> " + application.name + " is starting at " + new Date());
};

application.onStatus = function(info) {
  trace("onStatus> info.level: " + info.level + ", info.code: " + info.code);
  trace("onStatus> info.description: " + info.description);
  trace("onStatus> info.details: " + info.details);
};

application.onConnect = function(client, userName, password) {
  client.userName = userName;
  client.writeAccess = "/public";
  client.readAccess = "/";
  application.acceptConnection(client);
  trace("onConnect> client.ip: " + client.ip);
  trace("onConnect> client.agent: " + client.agent);
```

```
  trace("onConnect> client.referrer: " + client.referrer);
  trace("onConnect> client.protocol: " + client.protocol);
};

application.onDisconnect = function(client) {
  trace("onDisconnect> client.userName: " + client.userName)
  trace("onDisconnect> disconnecting at: " + new Date());
};

application.onAppStop = function(info) {
  trace("onAppStop> application.name: " + application.name);
  trace("onAppStop> stopping at " + new Date());
  trace("onAppStop> info.level: " + info.level);
  trace("onAppStop> info.code: " + info.code);
  trace("onAppStop> info.description: " + info.description);
};
```

Рассмотрим самые важные методы обработчика ошибок объекта application, с использованием этого примера. Они вызываются автоматически, когда запускается приложение, когда клиент соединяется, когда отсоединяется или приложение должно быть закрыто.

application.onAppStart()

Когда экземпляр приложения получит первый раз доступ, компилируется сценарий и выполняется любой глобальный код (т.е. код вне обработчика ошибок). После этого вызывается метод application.onAppStart(). В примере свойство application.name выводит имя запущенного экземпляра. Имя всегда будет выводиться в формате appName/instanceName. Например, "helloWorld/_definst_" имя по умолчанию для приложения helloWorld. Если экземпляр находится в каком-либо каталоге, свойство name может содержать такую строку, как "courseChat/chem101/room1".

application.onStatus()

Метод application.onStatus() получает дополнительные сообщения для серверных объектов Stream и NetConnection, у которых нет обработчика onStatus(). Обработчик onStatus() не вызывается в примере, но его объявляют, чтобы показать полученные свойства информационных объектов.

application.onConnect()

Метод application.onConnect() показывает взаимодействие между переданными объектами application и client. Объект client является экземпляром класса Client, который содержит информацию о попытке соединения Flash ролика, например, IP-адрес. Экземпляр класса Client всегда передается в метод onConnect(). В этом примере каждый экземпляр назван как client. Разница в применении заглавных букв не приводит к конфликту имен между именем класса Client и отдельным объектом client, передаваемом в onConnect(). Если это кажется путанным или предпочтительнее присваивать условные имена, которые привычнее при написании кода в Flash MX, тогда вместо использования client как имени объекта, можно использовать newClient:

```
application.onConnect = function(newClient, userName, password) {
 newClient.userName = userName;
 application.acceptConnection(newClient);
};
```

Наряду с IP-адресом (свойство ip) у каждого объекта client есть свойства, содержащие имя агента пользователя, ссылку на страницу и протокол соединения. Другие свойства объекта client включают в себя readAccess и writeAccess, которые могут использоваться для контроля относительных URI, к которым у клиента есть доступ и, следовательно, какие потоки и совместно использующиеся объекты доступны каждому

клиенту. В примере клиенту позволено запросить любой ресурс в каталоге public или его подкаталоге, потому что writeAccess является "/public". Клиенту также разрешено считать (получить доступ) в любой каталог, так как readAccess был установлен "/", что является корневым каталогом URI внутри экземпляра.

В объект client могут быть добавлены свойства, в этом примере свойство userName добавляется динамически. Переменная userName, переданная в метод onConnect() является именем, под которым клиент входит в систему, код хранит его как свойство также названное userName объекта client таким образом, что оно доступно даже после работы обработчика onConnect().

Обработчик onConnect() в примере позволяет клиенту соединиться с экземпляром приложения, вызывая application.acceptConnection(). Запросы клиента на соединение также могут быть отклонены, используя application.rejectConnection(). Вызывать эти методы внутри onConnect() необязательно. Если onConnect() возвращает true, доступ клиенту разрешен. Соединение разрывается, если onConnect() возвращает false. Что еще более важно, если onConnect() ничего не возвращает или возвращает null, клиент остается в состоянии ожидания и не может связаться с сервером. Соединения с сервером для клиентов все еще могут приниматься или отклоняться вне метода onConnect(), вызывая application.acceptConnection() или application.rejectConnection().

Если метод application.onConnect() не определен, то все соединения клиента приняты или у всех клиентов есть глобальный доступ к чтению и записи, и установление readAccess и writeAccess эквивалентно "/".

application.onDisconnect()

Метод application.onDisconnect() вызывается, когда FlashCom обнаруживает, что клиент, имеющий доступ или ожидающий разрешения, отсоединился от экземпляра приложения. В примере, свойство client.userName определяет, кто покинул сервер.

application.onAppStop()

И самое последнее, application.onAppStop() вызывается перед самым завершением работы приложения. Он передает информационный объект, который определяет причину завершения работы, он может предотвратить закрытие экземпляра приложения, возвращая значение false.

Использование App Inspector для запуска сценариев

Чтобы проверить сценарий подобный описанному в примере, надо создать текстовый файл main.asc (или загрузить его из журнала web-сайта) и сохранить его в подкаталоге applications с именем helloWorld. Найти клип app_inspector.swf (расположенный в каталоге FlashCom 1.5 flashcom_help/html/admin). Этот .swf является тестирующим инструментом, известным как App Inspector.

Запустите клип app_inspector.swf, открыв его в Flash Player, или загрузите страницу app_inspector.html в браузере и введите положение сервера в адресной строке. Если FlashCom запущен на вашем компьютере, введите localhost в адресной строке. Если FlashCom запущен на удаленном сервере, введите IP-адрес или имя хоста. В полях Имя и Пароль введите имя пользователя и пароль администратора, которые были выбраны в процессе установки FlashCom.

После соединения введите helloWorld в поле App/Inst выберите мышью кнопку Load, как показано на рисунке 40. Если экземпляр не загружается, появится информационная иконка рядом с кнопкой Load. Нажмите на нее мышью, чтобы узнать, в чем ошибка, исправьте и пересохраните сценарий, если это необходимо, и попробуйте снова.

Рисунок 40. Загрузка приложения в App Inspector

Если загрузка экземпляра helloWorld/_definst_ прошла не успешно, нажмите на кнопку View Detail как показано на рисунке 41.

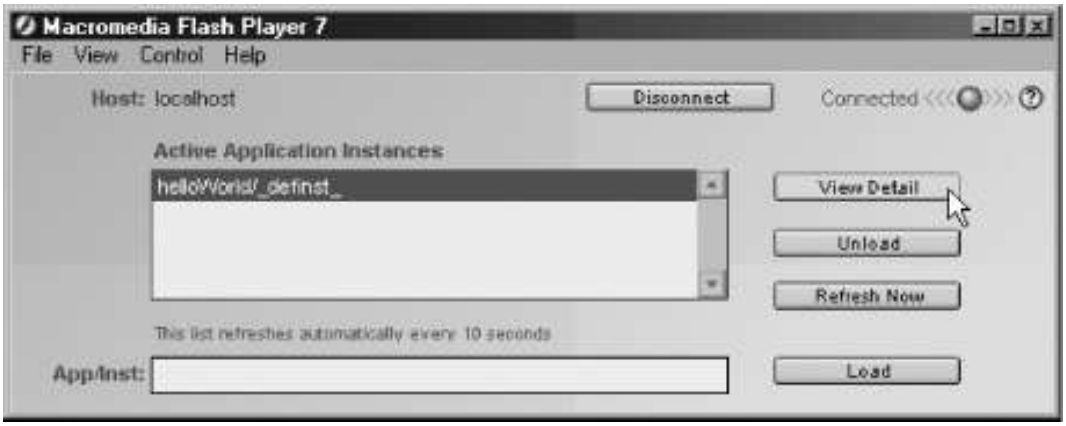

Рисунок 41. App Inspector после загрузки приложения helloWorld

Если загрузка экземпляра helloWorld/_definst_ прошла успешно, выберите в App Inspector закладку Live Log. Нажмите кнопку Reload App, чтобы перезапустить экземпляр так, чтобы вы могли видеть результат вывода метода application.onAppStart() тестового сценария.

Рисунок 42 показывает результат в области Live Log App Inspector-а после перезапуска приложения helloWorld.

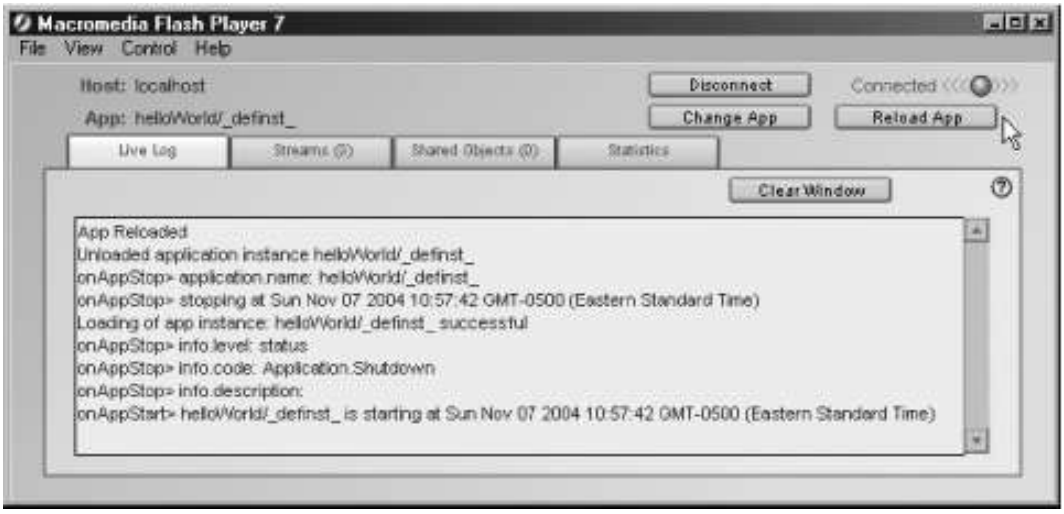

Рисунок 42. App Inspector после загрузки helloWorld/_definst_

В журнале регистрации показано, что экземпляр приложения завершил работу и был перезапущен, но сообщения об этом не появится. Системные сообщения и сообщения trace() очень часто смешиваются. Например, системное сообщение "Loading of app instance: helloWorld/_definst_ successful" появляется между сообщениями сгенерированными функцией trace() во время разгрузки метода onAppStop() от экземпляра. Сообщения, сгенерированные сценарием в предыдущих примерах, появляются только в методах onAppStart() и onAppStop(). Чтобы увидеть сообщения, относящиеся процессу подключения клиента и отсоединения от экземпляра приложения, требуется ролик, который подключается и отключается от экземпляра. Тестовый ролик очень важный инструмент. Хороший, но простой ролик позволит:

- Ввести или выбрать разные экземпляры приложения, к которым надо подключиться
- Ввести имя пользователя и пароль или другие параметры соединения
- Прочитать любое полученное или сгенерированное сообщение состояния системы или сообщения об ошибке
- Легко добавлять нужные свойства

На рисунке 43 показана Live Log область, когда связь с клиентом установлена. IP-адрес 127.0.0.1 показывает, что клиент работал на той же системе, что и FlashCom сервер.

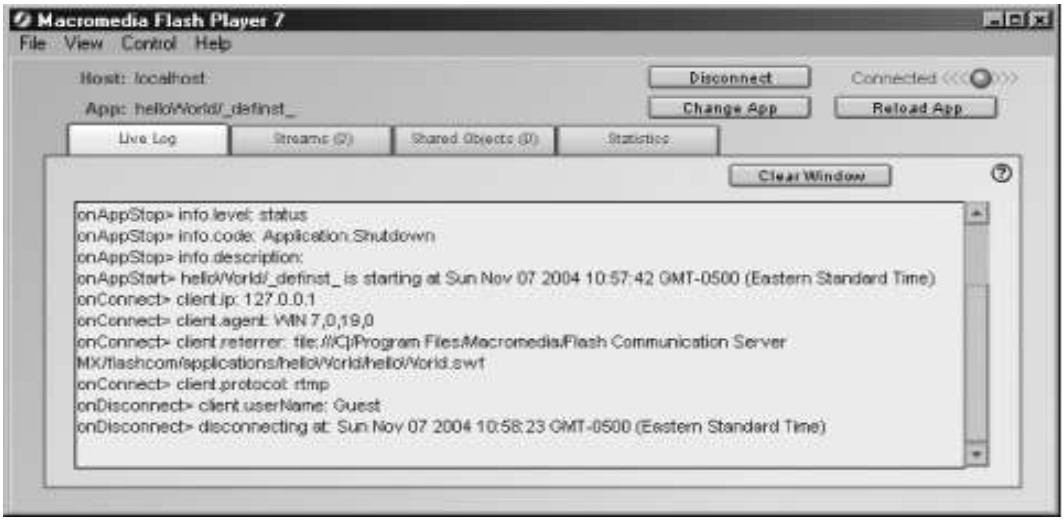

Рисунок 43. App Inspector после подключения и отключения клиента от helloWorld/_definst_

Через 23 минуты после отсоединения клиента экземпляр завершает работу и выводится следующий результат на рисунке 44 в App Inspector.

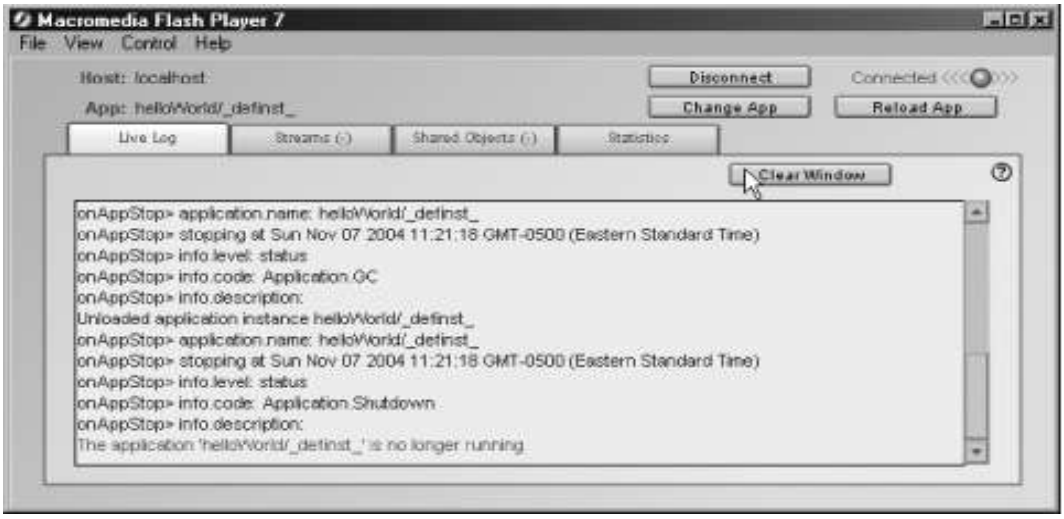

Рисунок 44. The App Inspector после завершения работы экземпляра helloWorld/_definst_

Результатом в Live Log области App Inspector-а являются такие параметры, как время загрузки и завершения работы, IP-адрес и имя пользователя клиента, а также когда клиент подключается и отсоединяется. App Inspector и серверный метод trace() необходимы при тестировании и отладки серверных сценариев.

Более реалистичный пример

Создание и тестирование простого main.asc сценария для helloWorld – самый важный шаг к более близкому знакомству с новыми возможностями FlashCom. Однако пока приложение helloWorld знакомит только с объектами application, Client и info, и оно не очень практично. Хотя реализованный *main.asc* в следующем примере не полнофункциональный, он демонстрирует многие из уже описанных ранее методов, работающих более стандартным способом. В примере будет продемонстрировано:

- Принятие или отклонение запроса клиента на соединения основано на использовании имени пользователя и пароля;
- Более простое добавление свойств и методов объекту Client путем добавления свойства объекту вместо отдельных свойств и методов;
- Ограничение пользователей ‘гость’, когда достигнуто максимальное число пользователей;
- Использование серверной функции setInterval() для периодичного обновления информации о соединившихся клиентах.

Код в примере не экономит место на web-сайте в отличие от предыдущей версии. Внимательно прочтите приведенный ниже код.

Пример 21. Более реалистичный main.asc

```
users = { Brian: {password: "secretPassword1", role: "author"},
     Robert: {password: "secretPassword2", role: "author"},
     Justin: {password: "secretPassword3", role: "author"},
      Joey: {password: "secretPassword4", role: "author"},
      Peldi: {password: "secretPassword5", role: "author"},
      Guest: {password: "Guest",      role: "guest"}
};
access = {author: {readAccess: "/", writeAccess: "/"},
      guest: {readAccess: "/public", writeAccess: ""}
};
MAXCONNECTIONS = 5;

function User (client, userName, role) {
 this.client = client;
 this.userName = userName;
 this.role = role;
 this.connectTime = new Date().getTime();
}

User.prototype.getTimeConnected = function () {
 var now = new Date().getTime();
 return (now - this.connectTime)/1000; // seconds
};

User.prototype.getPingTime = function () {
 var client = this.client;
 if (client.ping()) {
  return client.getStats().ping_rtt / 2;
 } else {
  return "Client is not connected."
 }
};

User.prototype.getConnectionInfo = function () {
```

```
 return " IP: "       + this.client.ip +
     ", user name: "    + this.userName +
     ", connection time: " + this.getTimeConnected() +
     ", ping time: "    + this.getPingTime();
};

function listCurrentUsers () {
 trace("-------------- Current Users --------------");
 var i, user;
 var total = application.clients.length;
 trace("There are " + total + " current users.");
 for (i = 0; i < total; i++) {
  user = application.clients[i].user;
  trace(user.getConnectionInfo());
 }
 trace("-----------------------------------------");
}

application.onAppStart = function () {
 trace(application.name + " is starting at " + new Date());
 setInterval(listCurrentUsers, 60000);
};

application.onConnect = function (client, userName, password) {
 trace("Connection attempt from IP: " + client.ip + " userName: " +
userName);
 var user = users[userName];
 if (!user || !password || user.password != password) {
  application.rejectConnection(client, {msg: "Invalid user name or
password."});
  return;
 }
 if (application.clients.length >= MAXCONNECTIONS && user.role == "guest") {
  application.rejectConnection(client, {msg: "Chat is already full."});
  return;
 }
 client.user = new User(client, userName, user.role);
 client.writeAccess = access[user.role].writeAccess;
 client.readAccess = access[user.role].readAccess;
 application.acceptConnection(client);
 trace("Connection accepted at " + new Date());
};

application.onDisconnect = function (client) {
 trace(client.user.userName + " is disconnecting at: " + new Date());
 trace(client.user.userName + " was connected for " +
 client.user.getTimeConnected() + " seconds.");
};

application.onAppStop = function () {
 trace(application.name + " is stopping at " + new Date());
};
```

Далее рассмотрим главные выполняемые функции кода.

Аутентификация и настройка

Клиенты, соединяющиеся с приложением, могут быть аутентифицированы несколькими способами.

В примере показан упрощенный подход – с закодированным объектом users и ассоциативным массивом объектов. Доступ к содержащимся в нем объектам можно получить, используя собственные имена как Brian, Robert или Justin. В объекте содержатся свойства password и role для каждого пользователя. Например, в объекте users.Brian

содержатся password и role пользователя Brian. Получить доступ к паролю Brian-а можно используя точечное обозначение или оператор в квадратных скобках. В примере из метода onConnect() восстанавливается объект из users следующим образом:

```
var user = users[userName];
```

Если в userName содержится строка, совпадающая с именем свойства объекта users, переменная user присваивается ссылке на объект и у нее будут свойства password и role. Например, если userName “Brian”, то user.password будет secretPassword1. Если в userName не содержится собственное имя, существующее в users, значение user будет undefined. Все это принимается во внимание в цикле if:

```
if (!user || !password || user.password != password) {
 application.rejectConnection(client, {msg: "Invalid user name or
password."});
 return;
}
```

Каждая часть условия необходима для предотвращения ошибок и проверки подлинности пользователя. Например, если не найдено ни одного пользователя, переменная user не будет содержать ссылки на объект и выведет false. В этом случае в подключении клиенту будет отказано. Если пользователь ввел пустой или неправильный пароль в SSAS password выведет false и в доступе опять будет отказано. Если user примет значение объекта и password будет не пустой строкой, свойство user.password успешно обрабатывается и сравнивается с паролем клиента.

Убедитесь, что переменная содержит объект, прежде чем пытаться получать доступ к любому из свойств. Иначе на сервере появится ошибка типа данных, и выполнение сценария остановится до тех пор, пока сценарий экземпляра не будет размещен в другом потоке FlashCom, при вызове обработчика ошибок или другой функции обратного вызова. Ранее рассматривались операторы try/catch/finally, с помощью которых можно избежать ошибок останавливающих выполнение сценария.

Если пользователь успешно прошел аутентификацию, объект client может быть настроен для хранения информации о пользователе, контроле доступа к ресурсам и дополнительных методах. Чтобы собрать аналогичную информацию и методы вместе, надо добавить объект как новое свойство объекта client:

```
client.user = new User(client, userName, user.role);
```

Конструктор User также может быть использован для дальнейшей инициализации клиента без заполнения лишней информацией метода onConnect(). В примере конструктор User получает время соединения и хранит в объекте user наряду с userName, role и копией ссылки на client.

Доступ на чтение и запись для совместно использующихся объектов и потоков контролируется поиском свойств объектов readAccess и writeAccess в объекте access:

```
client.writeAccess = access[user.role].writeAccess;
client.readAccess = access[user.role].readAccess;
```

Например, если свойство user.role “guest” предыдущие выражения будут иметь вид:

```
client.writeAccess = "";
client.readAccess = "/public";
```

Пустая строка значит, что клиент не имеет права записывать или создавать любые совместно использующиеся объекты или публиковать любые потоки в этом экземпляре. Однако клиент может считать данные и воспроизвести потоки в каталоге или подкаталогах public. Если значение user.role “author”, то:

```
client.writeAccess = "/";
client.readAccess = "/";
```

Символ “/” означает, что у клиента есть доступ к любому ресурсу.

Использование объекта Client.prototype

В предыдущем примере, перед разрешением на доступ, каждому объекту client задается свойство объекта user. Таким образом, у каждого клиента будет информация о

каждом пользователе, например, его имя, а также будут содержать общие методы такие, как user.getConnectionInfo(). В некоторых приложениях, где у пользователя нет совместно использующихся объектов, самый эффективный способ добавить общие методы каждому объекту – прикрепить их к Client.prototype. В следующем примере показана часть сценария с использованием объекта prototype:

Пример 22. Использование объекта Client.prototype

```
Client.prototype.getTimeConnected = function () {
 var now = new Date().getTime();
 return (now - this.connectTime)/1000; // seconds
};

application.onConnect = function (client, userName, password) {
 client.connectTime = new Date().getTime();
 application.acceptConnection(client);
};

application.onDisconnect = function (client) {
 trace("Client connected for: " + client.getTimeConnected());
};
```

Класс Client не может стать подклассом, потому что конструктор Client никогда не вызывается непосредственно в сценарии. Если необходимо расширить класс Client, необходимо использовать композицию. Одним примером композиции является добавление объекта user каждому клиент-приложению, а затем распределение такой работы, как получение информации о соединении клиент-приложения. Для принятия на себя обязательств от имени клиента могут быть добавлены и другие объекты. Более подробно о добавлении методов прототипу класса можно найти в книге ActionScript для Flash MX: Полное руководство. Для полного рассмотрения композиции в сравнении с наследованием можно найти в Essential ActionScript 2.0.

Ограничение числа подключений клиентов

Число разрешенных подключений доступно через свойство length массива application.clients. Ограничение числа клиентов, которые могут подключиться к приложению в любое время, - основная причина проверки свойства length и отклонения запросов клиентов, когда достигает определенного размера. В некоторых приложениях может быть необходимо разрешить неограниченное количество пользователей одного типа и ограничить количество другого. В примере доступ запрещен гостям, когда число клиентов с доступом достигает определенного значения:

```
if (application.clients.length >= MAXCONNECTIONS && user.role == "guest") {
 application.rejectConnection(client, {msg: "Chat is already full."});
 return;
}
```

Этот пример показывает, как отклонить запрос на соединение. Можно использовать application.disconnect() в любое время для отсоединения клиента уже с разрешенным доступом.

Выполнение периодических обновлений при помощи setInterval()

Часто необходимо выполнить несколько периодичных действий. На стороне клиента это можно сделать несколькими способами: используя временную шкалу или событие enterFrame. Однако они не доступны на сервере. В серверном коде можно использовать setInterval() для вызова функции или метода. Функция или метод могут быть вызваны один или несколько раз через постоянные интервалы. В методе onAppStart(), в примере, вызывается setInterval() и передает ссылку в функцию listCurrentUsers():

```
setInterval(listCurrentUsers, 60000);
```

В этом примере listCurrentUsers() вызывается каждую минуту (каждые 60000 миллисекунд). В свою очередь listCurrentUsers() повторяется для массива

application.clients и вызывает каждый метод user.getConnectionInfo(). Этот метод использует свойства объекта client, свой объект user и методы класса User для вывода информации о клиенте. Все свойства и методы класса Client описаны в Macromedia's Server-Side Communication ActionScript Dictionary (доступном в PDF формате на приведенном ранее сайте, а также в формате LiveDoc на сайте http://livedocs.macromedia.com/flashcom/mx2004).

На экран можно вывести различные значения свойств класса Client. Методы ping() и getStats() класса Client делают больше, чем просто снабжают статической информацией о клиенте. Вызов ping() сразу же посылает текстовое сообщение по кругу от сервера к Flash клиенту и обратно. Чтобы вернуться, сообщению может понадобиться некоторое время, поэтому метод ping() возвращает true, если предыдущее ping сообщение было возвращено клиентом (или сервер полагает, что клиент все еще подключен) или false, если клиент не подключен:

```
if (client.ping()) {
 return client.getStats().ping_rtt / 2;
}
else {
 return "Client is not connected."
}
```

Если ping() возвращает true, вызов метода getStats() возвратит информационный объект, свойство ping_rtt которого содержит время кругового прохода в миллисекундах от последнего вызова ping(). Метод ping() должен использоваться с осторожностью, так как он посылает сообщение с самым высоким приоритетом, потенциально задерживая другие RTMP сообщения. При использовании ping() для определения времени ожидания в сети, не следует вызывать этот метод очень часто. По умолчанию в Macromedia's ConnectionLight принят интервал вызова ping() в 2 секунды.

Обмен информацией между экземплярами

Серверному сценарию доступен класс NetConnection. Аналогичный клиентскому классу NetConnection, он может устанавливать сетевую связь между экземплярами на отдельном сервере или между приложениями на FlashCom сервере. Когда одно приложение пытается подсоединиться к другому, оно создает и использует объект NetConnection почти таким же образом, как и клиентский ActionScript. Экземпляр, подключающийся к другому экземпляру, обрабатывается вторым экземпляром также как клиент. Экземпляр, который получил запрос на подключение от другого экземпляра, будет передан в метод application.onConnect() объекта Client. В этом случае объект Client представляет собой первый экземпляр серверного ActionScript, а не клиентское Flash приложение.

Экземпляр может в любое время установить или разорвать соединение с другим экземпляром. Обычно соединение происходит, когда экземпляр запускается первый раз и закрывается, когда экземпляр завершает свою работу. Например, приложение чата может соединиться с приложением, объединяющим комнаты, с тем, чтобы дать ему знать, сколько пользователей находится в комнате чата и насколько они активны. Пример кода SSAS демонстрирует приложение чата, соединяющегося с приложением, объединяющего комнаты, при запуске и разрывающем соединение при отключении от сервера:

```
NetConnection.prototype.onStatus = function (info) {
 trace("NetConnection.onStatus> info.code: " + info.code);
 if (info.code == "NetConnection.Connect.Success") {
  // Initialize remote shared objects here.
 }
 else if (!this.isConnected) {
```

```
  // Handle close and other connection problems here.
 }
 if (info.code == "NetConnection.Call.Failed") {
  // Handle failed remote method calls here.
 }
};

application.onAppStart = function () {
 trace(application.name + " is starting at " + new Date());
 lobby_nc = new NetConnection();
 lobby_nc.connect("rtmp:/localhost/chapter4/lobby", "Room",
"secretPassword6");
};

application.onAppStop = function () {
 if (lobby_nc.isConnected) {
  lobby_nc.close();
 }
 trace(application.name + " is stopping at " + new Date());
};
```

Единственное отличие между серверным кодом и клиентским кодом, которое можно найти во Flash ролике, состоит в том, что не может быть использован соответствующий URI (должно существовать имя хоста) и не доступно HTTP туннелирование. Соединяющийся экземпляр представлен объектом Client, переданный в метод экземпляра onConnect(). Если использовались имя пользователя и пароль, соединение может быть принято или отклонено на их основе.

В конце попытки соединения не требуется специального кода чтобы обработать запросы на соединение от других экземпляров. Однако иногда очень полезно различать запросы от приложения FlashCom и Flash ролика. Один способ различить это состоит в том, чтобы проверить свойство ip от поступающего объекта client. Если IP-адрес 127.0.0.1, тогда соединение с того же самого сервера. Однако если плохо сконфигурированный прокси-сервер запущен на том же самом хосте, все клиенты могут иметь один и тот же IP-адрес 127.0.0.1. Подобным образом, если соединения ожидаются от другого FlashCom сервера, можно проверить IP-адрес удаленного сервера. Существуют другие способы различить типы клиентов:

- Передать информацию в дополнительном параметре метода connect();
- Проверить свойство client.agent на значение: "FlashCom/1.5.2";
- Проверить свойство client.referrer на значение "rtmp://_defaultVHost_:1935/chapter4/room".

К сожалению, к этим подходам требуется дополнительная стадия аутентификации. Небольшой отрывок кода, который последует далее, может быть помещен в метод onConnect() для обработки связей экземпляра перед попыткой обработки связи Flash ролика.

Предполагается, что в коде уникальный userName получается в результате соединения экземпляра таким образом, что объект client может храниться в объекте roomList. Уникальным именем может стать имя экземпляра комнаты чата:

```
if (client.ip = "127.0.0.1" && client.agent.indexOf("FlashCom") == 0) {
 if (password == "room54780561Password") {
  roomList[userName] = client;
  return true; // Accept the connection
 }
 else {
  trace("Invalid room connection attempt.");
  return false; // Reject the connection
 }
}
```

Запросы на соединение могут быть приняты или отклонены, возвращая значения true или false метода onConnect() в предыдущем примере.

Теперь вы лучше понимаете, как управлять соединениями, теперь рассмотрите механизм расположения кода и файлы конфигурации.

Имя сценария и их расположение

В приложении содержатся один или более файлов SSAS (открытого типа). Рассмотрите подробнее структуру файлов в серверном приложении.

Главный файл сценария приложения

Когда начинает выполняться экземпляр приложения, он ищет для запуска главный сценарий. У файла может быть одно из двух имен, и он может храниться в одном из двух мест. Файл может быть назван *main* с расширением *.asc* или *.js* (*main.asc* или *main.js*) или у него может быть такое же имя, как и у домашнего каталога приложения с тем же расширением. Например, если имя домашнего каталога приложения courseChat и если не существует файла *main.asc*, то будет загружен файл *courseChat.asc*. Главный файл сценария может храниться в одном из двух мест: в домашнем каталоге приложения или в подкаталоге с названием **scripts**. У файлов с расширением .asc имеется приоритет над файлами с расширением *.js*.

Использование load() для присоединения других сценариев

Если главный файл был загружен и любой глобальный код (код, который определяется вне обработчика событий, например, onConnect()) внутри выполнен, может быть загружен другой исходный файл. Метод load() принимает на компиляцию и исполнение относительный путь к внешнему файлу. Например, файл *main.asc* в домашнем каталоге приложения может загрузить файл из каталога **scripts** следующим образом:

```
load("scripts/myFile.asc");
```

В относительном пути не может использоваться символ “..” для обозначения выхода на уровень вверх в дереве каталогов, поэтому должен существовать другой путь для загрузки исходных файлов, являющихся общими для более чем одного приложения. При установке FlashCom создается каталог с именем **scriptlib**, где содержатся общие файлы сценария Macromedia. В каталоге **scriptlib** содержатся файлы для Flash Remoting и компонентов связи. Если файл не найден, сервер попытается найти его через относительный путь, начиная с каталога **scriptlib**. Например, когда главный файл загружает компоненты связи, он вызывает:

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

Файлы также могут располагаться в подкаталоге каталога **scriptlib**. Например, исходный файл *.asc* для отдельных компонентов связи находятся в каталоге scriptlib/components.

Расположение каталога **scriptlib** можно изменить с помощью тега <ScriptLibPath> файла *Application.xml*, к нему можно добавить дополнительный путь к каталогу с библиотекой. Например, файл *Application.xml* можно разместить в домашнем каталоге приложения. Тег выглядит следующим образом и зависит от пути, по которому был установлен сервер:

```
<ScriptLibPath>
C:\Program Files\Macromedia\Flash Communication Server MX\scriptlib
</ScriptLibPath>
```

Его можно модифицировать для добавления второго пути, отделенного от первого “;” (заметим, что строка разбита для удобочитаемости):

```
<ScriptLibPath>
C:\Program Files\Macromedia\Flash Communication Server MX\scriptlib;
C:\Program Files\Macromedia\Flash Communication Server MX\securitylib
```

```
</ScriptLibPath>
```

У тега <ScriptLibPath> есть ряд преимуществ. Можно создать свою собственную библиотеку сценариев без их размещения в каталоге **scriptlib** и возможного редактирования или перезаписи файлов Macromedia. Размещая файлы *Application.xml* в домашней директории отдельных приложений, можно создавать для них библиотеки. Если необходимо добавить библиотеку всем приложениям, в пределах виртуального хоста, следует добавить путь в файле *Application.xml* в каталоге виртуального хоста.

На рисунке 45 показано размещение нескольких исходных файлов для приложения courseChat. Каталог **securitylib** был создан для хранения сценариев клиента, использующихся любым приложением, а папка **scripts** – для хранения только сценариев приложения courseChat.

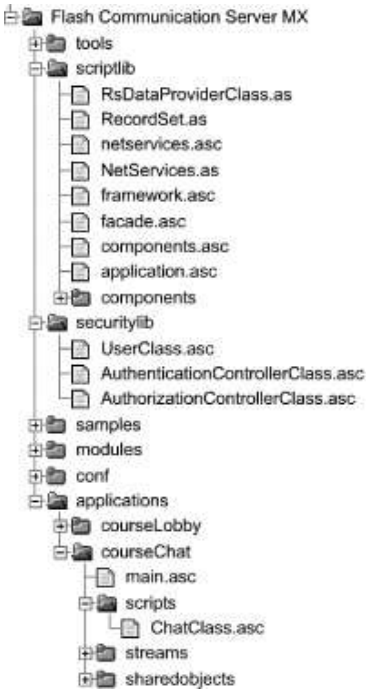

Рисунок 45. Расположение файлов сценариев

Первые две строки следующего исходного кода загрузки SSAS содержатся в файлах, которые используют удаленный Flash или компоненты связи. Оставшиеся строки кода загружаются в исходном файле из каталогов **scripts** и **securitylib** (последний сконфигурирован через тег <ScriptLibPath> в файле *Application.xml*):

```
// Load files from the scriptlib directory.
load("netservices.asc"); // Load the files required for Remoting
load("components.asc"); // Load the component framework and components

// Load a file from the scripts directory.
load("scripts/ChatClass.asc");

// Load files from the securitylib directory.
load("UserClass.asc");
load("AuthenticationControllerClass.asc");
load("AuthorizationControllerClass.asc");
```

Каталог scripts не требуется. Все исходные файлы могут храниться в домашнем каталоге приложения.

Динамическая загрузка файлов сценария

Тот факт, что файл компилируется и исполняется перед загрузкой другого файла, приводит к следующей особенности. В файле, который решает какие файлы будут загружены, возможно написание глобального кода, основанного на имени экземпляра

приложения или других факторах. Например, предположим, что нам необходимо чтобы экземпляр _definst_ вел себя как коридор, пока любой другой экземпляр приложения работает как комната. Весь файл *main.asc* мог бы содержать всего один оператор if, например:

```
if (application.name == "courseChat/_definst_") {
 load("lobby.asc");
}
else {
 load("room.asc")
}
```

Тогда каждый файл *lobby.asc* и файл *room.asc* будет определять разные наборы методов приложения и поэтому экземпляр _definst_ будет вести себя иначе, чем другие экземпляры.

Тестирование и отладка серверных файлов сценария

В настоящее время не существует отладчика, доступного для тестирования и отладки серверных сценариев, который бы обеспечивал непосредственную проверку данных или пошаговую проверку кода. Основной способ получить представление о состоянии исполняющегося сценария – использовать функцию trace() для вывода информации. Отследить вывод можно разными способами.

Отладчик NetConnection, рассмотренный в предыдущей части в разделе “Использование отладчика NetConnection”, сведет события trace() в таблицу Events, а результаты в таблицу Details. Эта функция наиболее полезна, когда вы работаете над Flash роликом и хотите просмотреть реакцию сервера на изменения, вносимые вами в ролик и его повторяющееся подключение к FlashCom.

Для тестирования и исправления серверного кода основным инструментом является ролик App Inspector (*app_inspector.swf*). Всякий раз, когда в код будут внесены изменения, экземпляр должен быть повторно загружен для того, чтобы исходные файлы, содержащие обновленный код, были повторно компилированы и выполнены. App Inspector может быть в любое время использован для загрузки, перезагрузки и выгрузки экземпляра приложения. Также легко можно выбрать любой запущенный экземпляр из списка и просмотреть статистику соединений и пропускной способности, статус потока и совместно использующегося объекта, а также журнал регистрации, который включает системные сообщения и прослеживает результаты.

В отличие от других сред разработки сервера результаты трассировки трудно перевести в текстовый файл для дальнейшего изучения. Однако во FlashCom существует средство для записи результатов трассировки в потоковый файл. Чтобы активировать приложение на запись (термин Macromedia для записи результатов трассировки), в файле Application.xml должен содержаться XML тег и следующее значение:

```
<RecordAppLog>true</RecordAppLog>
```

Со временем при записи информации могут создаваться очень большие файлы, поэтому не рекомендуется вносить эти изменения в файле *Application.xml* для всего виртуального хоста. Файл *Application.xml* можно расположить в домашнем каталоге разрабатывающегося приложения. Когда запись включена, потоковый файл с расширением *.flv* с именем как у экземпляра будет сохранен в папке с именем как у приложения в каталоге **applications/admin/streams/logs/application**. Файл можно прочитать, используя Flash Communication Server Log reader, доступный в пакете Macromedia. Смотрите:

```
http://www.macromedia.com/support/flashcom/ts/documents/flashcom_logging.htm
```

Существуют другие варианты для вывода результатов из системного журнала. Flash Remoting может использоваться для отправки данных на сервер приложений, где

они могут храниться в базе данных или заносятся в текстовый файл. Когда используется Remoting вместо trace() должна вызываться функция записи. Подобным образом Flash клиенты могут получать данные через вызов удаленных методов, где они могут сортировать и анализировать данные перед их представлением пользователю.

Организация тестовых сценариев

Во время разработки и тестирования программы load() делает проще процесс тестирования различных файлов SSAS. Если бы Вам пришлось создавать новый домашний каталог приложения для каждого тестового сценария, все окончилось бы тем, что у Вас стало много разных папок с тестами. Существует одна тонкость: следует использовать разные экземпляры для тестирования различных сценариев в одном домашнем каталоге приложения. Для этого надо создать файл *main.asc* с такой строкой кода:

```
load(application.name.split('/').pop() + ".asc");
```

Затем в домашнем каталоге можно создавать столько тестовых файлов, сколько нужно. Чтобы запустить тестовый файл, подключитесь к экземпляру своего приложения с тем же именем. Предположим, например, вы работаете с приложением courseChat и хотите запустить разные версии своего главного файла приложения. У вас может быть несколько главных файлов с именами *mainVersion1.asc, mainVersion2.asc* и так далее. Можно запустить любой из них при помощи App Inspector, который загрузит правильное имя экземпляра такое, как courseChat/mainVersion2, а затем перезагрузить его для проверки результатов. Если требуется взаимодействие с клиентом, тест-клиент может быть настроен для соединения с rtmp:/courseChat/mainVersion2.

Использование файла main.asc для загрузки других файлов из домашнего каталога приложения хорошо подходит для тестирования других версий главного файла. Когда главный файл отлажен и готов к использованию, его можно переместить в рабочий каталог и переименовать в main.asc.

Тестирование отдельных объектов и функций – очень важная часть разработки приложений, если вы не хотите засорить домашний каталог приложения большим количеством тестовых файлов. В качестве решения этой проблемы может быть создан файл *main.asc*, который содержит этот короткий сценарий:

```
this.tempPath = application.name.split("/");
this.tempPath.shift();
load(this.tempPath.join("/") + ".asc");
//delete this.tempPath; // Uncomment this if you want to delete tempPath.
```

Сценарий будет загружать файлы, основанные на полном имени экземпляра. Например, если приложение-клиент соединяется с экземпляром rtmp:/courseChat/testBed/userClassTests, то будет загружен и выполнен файл *userClassTests.asc* в каталоге **testBed**.

Разработка взаимодействующих приложений

Разработка взаимодействующего приложения значит больше, чем разработка приложений FlashCom и его экземпляров. Взаимодействующее приложение может включать более одного приложения FlashCom и много экземпляров приложений. Самый простой пример – работающие вместе приложения чата и “коридора”, обеспечивающие работу чата. Пользователи могут посещать различные экземпляры “коридора” приложения “коридор” прежде, чем соединяться с экземпляром приложения чат-комнаты.

Приложения и экземпляры приложений - основные средства, обеспечиваемые FlashCom для распределения возможностей сервера и объединения взаимодействующих приложений. Пользователи должны быть сгруппированы в отдельных приложениях с тем,

чтобы поддерживать производительность системы и распределять ресурсы такие, как совместно использующиеся объекты и потоки.

Для разработки взаимодействующих приложений необходимо планировать, какие будут доступны приложения и экземпляры, какую работу они должны выполнять и какие средства, такие как потоки, совместно использующиеся объекты и доступ к базе данных, нужны для их реализации. Очень часто при проектировании необходимо учитывать, как экземпляры будут создаваться и кем управляться, например другим экземпляром, и как клиент-приложения будут взаимодействовать с ними.

Заключение

В этой части было рассказано об объекте application и классе Client, а также работе с ними. Эти объекты используются во всех приложениях FlashCom.

Web-программирование. Серверный ActionScript.
Николаев Дмитрий Геннадьевич, Штенников Дмитрий Геннадьевич
Учебное пособие
Редакционно-издательский отдел СПбГУИТМО
Лицензия ИД № 00408 от 05.11.99
Компьютерная верстка: Николаев Д.Г., Штенников Д.Г.
Дизайн обложки: Николаев Д.Г., Штенников Д.Г.
Художественное оформление: Николаев Д.Г., Штенников Д.Г.
Подписано к печати 30.08.06. Тираж 300 экз. Заказ №
Отпечатано на ризографе в Центре издательских систем
СПбГУИТМО