ИНТЕРНЕТ

Штенников Дмитрий Геннадьевич,
Николаев Дмитрий Геннадьевич

ОСНОВЫ ИСПОЛЬЗОВАНИЯ MACROMEDIA FLASH COMMUNICATION SERVER (FLASH MEDIA SERVER) ДЛЯ СОЗДАНИЯ ИНТЕРНЕТ-ПРИЛОЖЕНИЙ (часть 2)

В предыдущей статье были затронуты аспекты использования Macromedia Flash Communication Server для создания Интернет-приложений на основе серверного программирования на языке ActionScript, также были рассмотрены преимущества использования Flash-приложений для создания динамического контента. В данной статье будет продолжен рассказ про создание Интернет-приложений.

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 в существующем каталоге обслуживания или в базе данных, прежде чем войти в чат или просмотреть видео потоки. Системы баз данных могут использоваться, чтобы хранить большое количество информации, с которой FlashCom не может легко справляться, например, миллионы записей, которые хранят местоположение видео сообщений в видео почтовой системе. Каждая запись может содержать почтовый текст и местоположение зарегистрированного видео сообщения в пределах приложения FlashCom.

...чтобы хранить большое количество информации... например, миллионы записей...

Flash Player и FlashCom могут взаимодействовать с серверами веб-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 быстрее обратиться к прикладным серверам.

Flash Remoting – технология запроса/ответа, которая позволяет скриптам в клиенте Flash или FlashCom вызывать удаленные методы на прикладном сервере. Он использует HTTP, чтобы посылать и получать данные в AMF (Action Message Format). Удаленные методы могут извлекать инфор-

мацию 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 или без FlashCom. Дело в том, что Flash Remoting достаточно гибка, это позволяет FlashCom связываться эффективно, несмотря на отсутствие прямой поддержки XML или серверного доступа.

Рисунок 1 иллюстрирует некоторые опции связи для Flash и FlashCom. Flash-Клиент может обращаться к приложению сети непосредственно, как и FlashCom.

В некоторых случаях клиенты могут соединяться и с прикладным сервером и с FlashCom. В других случаях FlashCom может соединяться с прикладным сервером и передавать информацию от клиента или клиенту. 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 разрашены, 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 полномочия SSL могут использоваться сервером. Для получения дополнительной информации смотрите статью: http://www.macromedia.com/devnet/mx/flashcom/articles/firewalls_proxy.html

...комплексные ... данные преобразовываются в последовательную форму и параллельную автоматически.

Подготовка к работе

Вы можете загрузить Flash Communication Server бесплатно с сайта Macro-

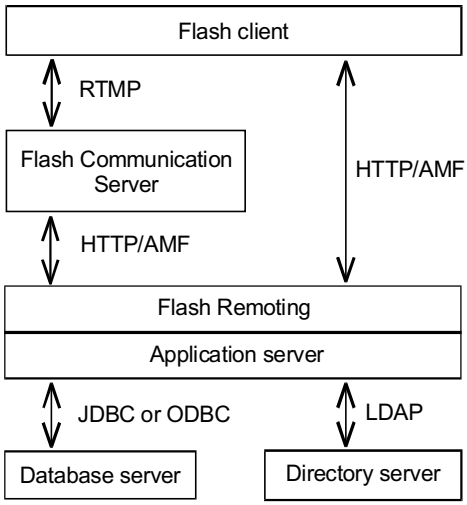


Рисунок 1. Опции связи для Flash и FlashCom

media. Более подробную информацию о загрузке и выборе линцензирования смотрите на сайте: http://www.macromedia.com/software/flashcom

Установка FlashCom:

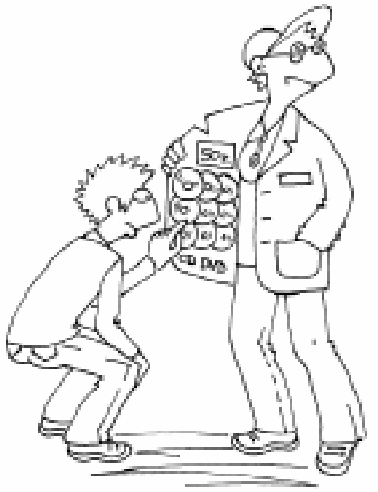
Загрузите бесплатную версию программы разработчика: http://www.macromedia.com/cfusion/tdrc/index.chm?product=flashcom

Как только вы устанавили FlashCom, надо загрузить и установить самый последний модуль обновления: http://www.macromedia.com/support/flashcom/downloads_updaters.html.

Модули обновления обычно не добавляются к инсталляционным файлам самого последнего выпуска, поэтому 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 расположены 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.



Когда каждый ... должен искать информацию, уникальную для него, прямое подключение к прикладному серверу — это обычно лучший подход.

Подробная документация о сервере, включая инсталляционные команды, доступна 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. Под Windows инсталлятор назван FlashComInstaller.exe, и вы можете просто запустить его. Под Linux вы должны разархивировать и затем инсталлировать файл в 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 находится 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. Смотрите ссылку: 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 в одном окне и наблюдать его движение в другом movie. Если это не работает, то вам, вероятно, придется вручную запускать сервер. В системе Windows необходимо будет выбрать **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 менеджер лицензий, контролирующий количество клиентов, которым позволяют соединиться в одно и то же время, и разрешает серверу исполь-

зовать полную пропускную способность канала связи. Есть также ограничения на издания сервера, которыми 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 вычислении пропускной способности и программных потребностей лицензии для Macromedia Flash Communication Server MX. Вы можете найти ее 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, используя одно из двух Flash movies, обеспеченных FlashCom. В FlashCom 1.5.2 версии вы найдете их в подкаталоге flashcom_help\html\admin инсталляционного каталога. Приложение (movie) Administration Console (admin.swf) может использоваться для обновления информации о лицензии, запуска и остановки прикладных запросов и просмотра на сервере диагностической информации.

App Inspector (app_inspector.swf) может использоваться, чтобы запускать и останавливать запросы приложений, контролировать прикладные ресурсы запроса, и выводить на дисплей сообщения `trace()`.

Чтобы использовать любой movie, запустите movie и войдите в **Admin Service**, используя имя и пароль администратора, который вы определили, когда установили 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.

ПРОСТЕЙШЕЕ ПРИЛОЖЕНИЕ «HELLOVIDEO»

Приложение helloVideo предназначено для демонстрации многих вещей, описанных в этой части: публикации и проигрывания потоков, обновления и реагирования на изменения в общедоступном объекте. Идея состоит в том, чтобы кратко осветить формирование 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 приложения на сервере требует, чтобы вы нашли каталог приложений и добавили подкаталог, названный helloVideo. При заданной по умолчанию инсталляции каталог приложений распо-

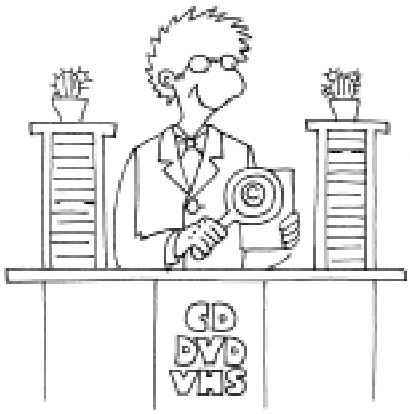
Некоторые общие средства межсетевой защиты...

Листинг 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);
};
```

ложен в: C:\Program Files\Macromedia\Flash Communication Server MX\applications.

Как только вы создали helloVideo-подкаталог, то одновременно 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 простого текста типа того, который включен в Flash MX Professional 2004 или Dreamweaver MX 2004. Листинг 1 показывает исходный текст, который вы должны добавить в файл main.asc.



Полное описание продукта и лицензирования может быть найдено здесь...

Файл будет загружен, откомпилирован, и выполнен сервером, когда первый клиент попытается соединиться 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()`**, а когда movie попробует разъединится, будет вызван **`application.onDisconnect()`**.

Файл main.asc, представленный в листинге 1, предназначен для следующего:

– 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 о 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 связан 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 файл main.asc загружен сервером (когда первый клиент пытается соединиться). Метод **application.onAppStart()** вызывается сразу, после того как файл main.asc загружен и выполнен. Метод **onAppStart()** 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.

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 листинге 1 имя, которое пользователь вводит, – второй параметр **name**.

Когда вызывается **onConnect()**, мы имеем опцию отклонения или принятия подключения или отправление 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 отправить клиенту его пользовательский **ID**, вызывая **setID()**, который был представлен ранее под «Удаленные 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, который клиент издаст:

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

Метод **setProperty()** сохраняет имя параметра в слоте, названном по имени строки **ID** клиента.

Позже, когда клиент отключается, щелкая кнопкой **Disconnect (Разъединить)**, или переходит в 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 общедоступного объекта, указав, что она не связана никакой длиной:

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

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"

Рисунок 2. Интерфейс HelloVideo.

Приложение помещает `ID` назад в массив `idPool` и присваивает 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 создаст поток, названный по имени его пользовательского `ID`, для контроля изменений в общедоступном объекте users, обнаруживая уникальный `ID` каждого пользователя, кто соединяется. Он использует пользовательский `ID` в общедоступном объекте users при запуске каждого отдаленного пользовательского потока.

СОЗДАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

На рисунке 2 изображен интерфейс пользователя для клиента helloVideo.

Интерфейс создан, используя четыре movie clip и несколько компонентов. В приложении присутствует один movie clip (MC) для каждого пользователя. MC содержит вложенный видео объект, метку, текстовое поле (**TextInput** компонент). **TextInput** компонент в каждом MC отображает имя каждого пользователя, введенного в поле `My Name`. Обратите внимание на **TextInput** компонент внизу экрана, содержащего текст «Robert». **TextInput** компонент 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:

Создайте приложение Flash и установите его размеры 320×480, 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**. Укажите **Export** для опции **ActionScript** (щелкните кнопкой **Advanced**, чтобы отобразить эту опцию, если ее не видно). Настройте также поле **Identifier** в **GuestVideo**.

Когда символ добавлен в библиотеку, 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 этого необходимо проделать следующие действия:

– добавить видео в библиотеку (откройте **Library panel** и выберите **New Video** из панели настроек, как показано на рисунке 3);

– перетащите видео объект из библиотеки на сцену и установите его в точке регистрации символа **GuestVideo** (используйте поля **X** и **Y** в панели **Properties**, как показано на рисунке 4, 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 4) и дайте **TextInput**-компоненту примера имя **nameInput**;

– установите параметр **Text** из **Метки** к тексту **Name**, используя панель **Properties**.

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

– вернитесь 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 2, установите каждому имя (**instance name**) **guest_1, guest_2, guest_3** или **guest_4**, используя панель **Flash's Properties**);

– внизу добавьте два компонента метки, два компонента **TextInput** и один компонент кнопки (их позиции и размеры показаны на рисунке 2);

– задайте имена для **TextInput** → **statusInput**, **TextField** → **userNameInput** и для кнопки – **connectButton**.

Теперь основы созданы, можно программировать.

Рисунок 3. Вставка нового видео.

Установки NetConnection и просмотр его состояния:

Листинг 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. Метод **setID()** вызывается сервером, чтобы уведомить клиента о его уникальном пользовательском **ID**. Метод **onStatus()** вызывается всякий раз, когда происходят изменения в состоянии подключения. Взгляните на листинг 2 и попробуйте понять, что эти два метода делают.

Через некоторое время после подключения к серверу, **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 объект **NetStream** на **nc** сетевом подключении. Это подключает микрофон и камеру к потоку и издает его, используя **id** как имя

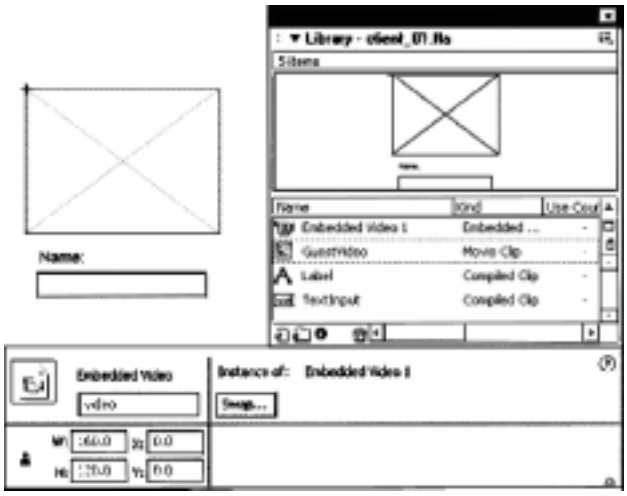


Рисунок 4. Использование меню **Library**, чтобы добавить **Видео**, **Метку** и **TextInput** к GuestVideo символу.

Листинг 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;
}
};
```

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**, на сцене сервер может обеспечить только один клип с тем же самым именем, которое было назначено пользователю. Свойство _root, 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. Метод **setID()** также вызывает метод **initUsers()** для установления общедоступного объекта пользователей.

Метод **onStatus()** 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 сооб-

Листинг 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;
}
}
```

щение, указывающее сетевое состояние подключения, отображено в statusInput текстовой области. Для соединения с сервером пользователь должен щелкнуть кнопкой Connect (connectButton). В это время кнопка блокируется, таким образом, пользователь не может сделать попытку второго подключения, не дождавшись результата первой попытки.

Метод **onStatus()** повторно запускает кнопку и ставит ее метку в Connect, пока подключение не будет установлено.

Создание подключения

В листинге 3 connectButton установлен, чтобы передать события щелчка **click()** функции всякий раз, когда кнопка нажата. Кнопка отображает одну из трех меток **Connect (Соединить)**, **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**, щелчок кнопкой закрывает текущее подключение, вызывая **nc.close()**.

Показ удаленных пользователей

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

Когда сервер вызывает метод клиента **setID()**, он вызывается методом **initUsers()**, показанным в листинге 4, для того чтобы установить объект общего доступа («зашаренный») и подключить его к пользователям группы объектов общего доступа. Листинг 4 демонстрирует три вещи: это получение общедоступного объекта, динамическое определение метода **onSync()** для него, и затем подключение его с использованием **nc**-сетевого подключения. Когда общедоступный объект 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 версии клиента, и **onSync()** будет вызван снова. Метод **onSync()** в листинге 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 в двух из них: "**change**" и "**delete**". Мы можем игнорировать остальное, потому что все модификации и стирание общедоступного объекта сделаны сер-

Листинг 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);
}
```

верным кодом в файле **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()** в коде листинга 4 производит проход по элементам массива от начала до конца массива информационных объектов и проверяет свойство кода каждого из объектов. 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.

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, 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, использующего **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 только 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 приложения helloVideo, то вы увидели большинство классов, работающих вместе, чтобы создать очень простое приложение видео конференции. Вы также увидели многие из основных методов, чтобы строить приложение связи.

Штенников Дмитрий Геннадьевич,
кандидат технических наук,
доцент кафедры «Компьютерные образовательные технологии»
СПбГУ ИТМО,

Николаев Дмитрий Геннадьевич,
ассистент кафедры «Компьютерные образовательные технологии»
СПбГУ ИТМО.