FLASH CLIENT MANUAL

Beam Jive Consulting

Table of contents

1 Introduction

1.1 About the document

The purpose of this document is to introduce you to the Offbeat Flash client. All examples are in the ActionScript 2 (Flash MX 2004 and Flash MX 2004 Professional) format.

It is recommended to read also the Offbeat Manual, which describes the protocol and the concepts of the server side programming with the Offbeat server.

1.2 What can I do with the Offbeat server and Flash Client?

With Offbeat server and Flash Client, it is possible to:

- Create data driven applications
- Create communication applications (chats, whiteboards…)
- Create controlling and monitoring applications
    - computer monitoring and remote control
    - software monitoring and remote controlling
    - device monitoring and controlling
    - real-time statistics
- Any kind of distributed systems

1.3 What can I not do with it?

It is not possible to create video or voice communication applications with the Offbeat server. For file uploads and file downloads, some HTTP-server should be used.

1.4 How does it work?

1.4.1 Communication

The Offbeat server is a TCP socket server that uses XML as the communication protocol. In a TCP socket connection, the client-server connection, unlike in HTTP, is continuously open. This means that the client may receive data from the server as push messages. This makes it possible to create real-time communication applications, such as chats and whiteboards.

The response times are very fast when there is no latency of creating the connection for each request. A typical round-trip time (send request -> process request -> receive response) for an Offbeat request is only a few milliseconds. The fastest round-trip times in the tests were as small as one millisecond. This means that the client may send up to 1000 requests in one second (depends on the hardware and the application design).

1.4.2 The communication model

The Offbeat communication model is based on a request-response model. A server programmer creates a server application, which consists of one or more Java class files. The Offbeat Flash Client can then call the server applications. It is possible to send variables to the server

application. The server application can read the request variables and create the response message dynamically. The response messages are always XML documents.

The Offbeat Flash client uses an asynchronous communication model (just as most of the Flash communication objects). This means that you always define a handler function for the response messages. You can process the response as soon as it is received from the server. There are six types of messages defined in the Offbeat protocol (see the Offbeat manual for more details):

- Requests
- Responses
- Push messages
- Client disconnect messages
- Log messages
- Acknowledgement messages

There are detailed descriptions of each message type in the following chapters.

2 Installing the client components

The Offbeat Flash Client component can be installed by using the Macromedia Extension Manager application that is freely available from the Macromedia web site. The installation can be done by clicking the .mxp. This will automatically start the Macromedia Extension Manager and install the component. When Flash is restarted, the component will appear in the Components panel.

3 How to use the client component

After installing the Offbeat Flash Client component successfully, it will appear in the Components panel of your Flash authoring environment. The client component can then be dragged on to the stage and the client component properties can be adjusted from the component properties panel. The component properties are explained in the following table:

Property	Description
Host	The host name or IP address of the computer where the Offbeat server is running. The default value is 127.0.0.1, which connects to the localhost (same computer).
Port	The port number of the Offbeat server. The default value is 8384. Please note that the Flash clients are not able to connect to ports below 1024 without a policy file.
onConnect	The function that will be called when the connection has been established with the server. The function takes one Boolean argument, which tells if the connection was made successfully.
onClose	The function that is called when the connection is closed (or lost).
onPushMessage	A function that is called when a push message is received from the server.
onClientDisconnect	A function that is called when a client has disconnected from the server. To get these messages, you need to register with the application.
onLogMessage	A function that is used to handle the incoming log messages.

When the component is on the stage, you should give it an instance name. The instance name is used to control the connection. To assign an instance name to the client, select the component on the stage and write a descriptive name to the Component text box in the Properties panel.

The component can be used also directly from ActionScript. To accomplish this, drag the component to the stage and delete it. This will add the component to your project's library. After that you would create a new client by writing:

```
// Create new client
var myClient = new com.bjc.offbeat.OffbeatClient( "127.0.0.1", 8384 );

// Define handler functions
myClient.onPushMessage = "myOnPushMessage";
myClient.onConnect = "myOnConnect";

// Connect to the server
myClient.connect();

...
```

4 Connection handling

This chapter will show you how to connect to a server, how to close the connection and how to handle the events correctly.

4.1 Defining host and port

To connect to Offbeat server, you need to know the correct host and port. Host is the IP address or the host name of the computer where the Offbeat server is running. Port is the port number that the server is listening. If the Offbeat server is on the same computer with the client, the host should be 127.0.0.1 or localhost.

The host and port parameters can be set from the Properties panel or with ActionScript. The following code shows how to define the host and port parameters with ActionScript:

```
var myConn:OffbeatClient = new OffbeatClient( "127.0.0.1", 8384 );
```

4.2 Settings the callback functions

The callback functions (handler functions) can be set from the component properties panel or by using ActionScript. The following code listing shows how to define the callbacks with ActionScript:

```
myConn.onConnect = "myOnConnect";
myConn.onClose = "myOnClose";
myConn.onPushMessage = "myPushHandler";
myConn.onLogMessage = "myLogHandler";
myConn.onClientDisconnect = "myOnClientDisconnect";

function myOnConnect( success:Boolean ):Void
{
   // The connection is OK
   if( success )
   {
   }
   // Connection failed
   else
   {
   }
}

function myOnClose():Void
{
   trace( "Connection was closed" );
}

function myPushHandler( data:XML, file:String, clientID:String ):Void
{
   // Handle push message
}

function myLogHandler( msg:String ):Void
{
   trace( "I got a log message: " + msg );
```

```
}

function myOnClientDisconnect(clientID:String, clientName:String):Void
{
    // A client has disconnected, remove from lists etc...
}
```

4.3 Opening the connection

When the handler functions have been set, it is time to open the connection. The connection can be opened simply by calling the connect() method. The following example show how to do it:

```
myConn.connect();
```

When the connection has been opened, the client will automatically call the handler function specified in the onConnect variable.

4.4 Closing the connection

To close an open connection, the close() method should be called. A call to the close() method will close the connection and call the handler function defined in the onClose property. The following code shows how to call the close() method:

```
myConn.close();
```

5 Sending requests

The requests are used to call an application on the server. The connection should be opened before sending the first request. It is possible to pass data to the server application by setting request variables. In the following example we send a basic request to the server:

```
var req = myConn.newRequest( myHandler, "MyApp/MyFile.xma" );
myConn.send( req );
```

In the example we sent a request to the application MyApp's file called MyFile.xma. In the server, there is an application called MyApp that contains a Java class file called MyFile.class. The file extension .xma has to be used when calling the server applications. The first parameter 'myHandler' is the function that will be called when the response arrives from the server.

It can be seen, that the connection object is used to create a new request. The method newRequest creates a new com.bjc.offbeat.XMLRequest object that will be converted to XML and sent to the server. The setVar(name:String, value:String) method of the XMLRequest class can be used to set request variables. The following example shows how set request variables:

```
var req = myConn.newRequest( saveHandler, "News/SaveNews.xma" );
req.setVar( "title", "Offbeat is a server" );
req.setVar( "text", "Offbeat really is a server!" );
myConn.send( req );
```

In the server application, the request variables can be read, and the news can be saved to a database.

If there is no need to set the handler function, the parameter may be an empty string. This is the case in many chat-like applications.

6 Receiving messages

6.1 General

There are five different kind of messages that a Offbeat Flash Client can receive. The most common type of message to receive is a response message. Response messages are generated on the server on your request. Push messages come from other clients.

6.2 Receiving normal response messages

Normal response messages a reply messages to your requests. The response messages are generated on the server by server applications. The response messages are always XML documents. The response may contain any data. The response message may contain for example database query results or a server generated timestamp or what ever you decide to add to the response message on the server.

A response message is handled in the handler function that was defined in the request. The following example shows how to send a request and how to handle the response:

```
// Send some request
function sendRequest():Void
{
   var req = myConn.newRequest( myHandler, "Test/Test1.xma" );
   myConn.send( req );
}

// The handler function
function myHandler( response:XML, errors:Number ):Void
{
   if( errors == 0 )
   {
          // Do something with the data ...
   }
   // There are errors, handle correctly
   else
   {
          // Do some error handling (loop though the errors)
   }
}
```

The first parameter to the handler function contains the XML formatted response message. The second parameter, errors, contains the number of error that occurred when generating the response on the server. If errors variable is zero, the response is OK and it can be processed. If there is one or more errors in the response, the error can be handled and an appropriate error message can shown to the user. There are two errors in the following response message:

```
<?xml version="1.0" encoding="UTF-8"?>
<MSG TYPE="0" FILE="metafile.xma" REQUEST_ID="123123" ERRORS="2">
   <ERROR CODE="0">Error description 1</ERROR>
   <ERROR CODE="3">Error description 2</ERROR>
</MSG>
```

6.3 Receiving push messages

The push messages are also generated by the server. The push messages are always sent by some other client that is using the same application. The push messages are also XML documents that can be handled in the same manner as the normal response messages. The following example show how to handle push messages:

```
// Set the push message handler
myConn.onPushMessage = "myPushHandler";

// ... open connection etc ...

// The handler function
function myPushHandler( data:XML, file:String, clientID:String ):Void
{
    switch( file )
    {
        case "myServerFile1.xma":
            // Do something with the message
            break;

        case "myOtherServerFile.xma":
            // Do something
            break;

        default:
            // No handler specified, do nothing
            break;
    }
}
```

By looking the code listing above, it can be seen that there are three variables that can be used to handle the push message. The first parameter, data, contains the push message data in XML object. The second parameter, file, can be used to check which server file generated the push message. The third parameter, clientID, is the unique client ID of the user who sent the message. If you have got a list of users from the server earlier, the client ID can be used to resolve for example the name of the sender.

6.4 Receiving client disconnect messages

The client disconnect messages are automatically generated by the Offbeat server when a client disconnects (or loses the connection). To receive the client disconnect messages, client has to register with an application. This can be done on the server application by calling: application.register(String name). This feature can be used to remove disconnected clients from lists etc. The following example shows how to set the handler function and how to handle the incoming disconnect messages:

```
myConn.onClientDisconnect = "myOnClientDisc";

function myOnClientDisc( clientID:String, clientName:String ):Void
{
   // Remove client from lists etc…
}
```

6.5 Receiving log messages

A client can receive log messages that are generated in the server application. On the server, the method user.receiveLogMessages(true), has to be called. This makes it easy to create simple monitoring features to applications. The following example shows how to set the property and how to define the handler function for incoming log messages:

```
// Set the callback for log message handler
myConn.onLogMessage = "myOnLog";

// Define the handler function
function myOnLog( msg:String ):Void
{
   trace( "I received one log message: " + msg );
}
```

6.6 Receiving acknowledgement messages

Acknowledgement messages are generated and sent by the server when the response message is not sent to the client who did the request. For example, if you send a chat message to another user, the server application probably will not send the same message back to you. The server sends an acknowledgement message back to you.

You do not have to do anything with the acknowledgement messages, they are used internally in the Offbeat clients. When you use the Offbeat Flash Client debug feature, you may see acknowledgement messages tracing to the output window.

6.7 Handling errors

Only the normal response messages may contain errors. The push messages will not be sent if an exception or error occurs in the server application.

335

7 Methods and property listing

OffbeatClient(host:String, port:Number) : Void

The constructor of the class. Creates a new instance of the OffbeatClient class. Can be used in the following way:

```
var myConn = new com.bjc.offbeat.OffbeatClient( "127.0.0.1", 8384 );
```

Please notice that the name of this class was XMLClient in the first beta version.

Parameters:

host: Host name or IP address of the Offbeat server

port: The Offbeat server port number

Returns:

Nothing

OffbeatClient.send(request:XMLRequest) : Boolean

Sends a request to the Offbeat server. The following example shows how the request can be sent through an open connection:

```
// First create a request
myRequest = myClient.newRequest( myCallback, "theFile.xma" );
// Then send it
myClient.send( myRequest );
```

Parameters:

request: The request object that will be sent to the server

Returns:

True if the sending succeeds, false otherwise

OffbeatClient.newRequest(cbk:Function, file:String) : XMLRequest

Get a new request object. The OffbeatClient initializes the request object so that it is ready to be used.

Parameters:

cbk: The callback function that will be called on the response

file: Name of the file that is called. The file extension should be .xma.

Returns:

New XMLRequest object

OffbeatClient.connect() : Void

Connect to the server. When the connection has been established, the OffbeatClient will call the

method that is specified in the onConnect property.

Parameters:

None

Returns:

Nothing

OffbeatClient.close() : Void

Closes the connection to the server. Calls the method that is defined in the onClose property of the OffbeatClient object.

Parameters:

None

Returns:

Nothing

OffbeatClient.isConnected() : Boolean

Can be used to check if the connection to the server is open.

Parameters:

None

Returns:

True if the connection is open, false if the connection is closed.

OffbeatClient.onPushMessage : String

A property that can be used to define the function that is called when a push message has been received. The push message handler function should always take three parameters: data:XML, file:String, clientID:String. The following code listing shows the basic way to set the push message function:

```
// Set the push message handler
myConn.onPushMessage = "myPushHandler";

// ... open connection etc ...

// The handler function
function myPushHandler( data:XML, file:String, clientID:String ):Void
{
    switch( file )
    {
        case "myServerFile1.xma":
            // Do something with the message
            break;

        case "myOtherServerFile.xma":
            // Do something
            break;

        default:
            // No handler specified, do nothing
```

```
            break;
    }
}
```

OffbeatClient.onConnect : String

With this property it is possible to set a callback function that is called after the connection has been made. The callback function takes one Boolean parameter that tells if the connection was successfully opened or not. The following example shows how the property should be set and also the format of the callback function.

```
// Set the callback
myConn.onConnect = "myOnConnect";
// Connect
myConn.connect();

// The callback function
function myOnConnect( success:Boolean ):Void
{
    if( success )
    {
        // The connection is now open
    }
    else
    {
        // Failed to create the connection
    }
}
```

OffbeatClient.onClose : String

By setting this property, it is possible to call a function when the connection is closed. The callback function will be called also when the OffbeatClient.close() method is called. The following example shows how to set the property and how to define the onClose callback function:

```
myConn.onClose = "myOnClose";

// The callback
function myOnClose()
{
   // The connection was closed
}
```

OffbeatClient.onClientDisconnect : String

Property that can be used to define a callback method that will be called when a client disconnect message has been received. The Offbeat server sends the client disconnect messages automatically to all clients who have registered to the same application as the disconnecting client (See the Offbeat user manual for further details). The following example shows how to set the property and how to define the callback function.

```
myConn.onClientDisconnect = "myOnClientDisc";
```

```
// The callback function
function myOnClientDisc( clientID:String, clientName:String ):Void
{
    // A client has disconnected, remove from lists etc...
}
```

OffbeatClient.onLogMessage : String

This property can be set to handle incoming log messages. See the Offbeat user manual to find out more about logging. The following example shows how to set the property and how to define the callback function that handles the log messages:

```
myConn.onLogMessage = "myOnLog";

// The log message handler function
function myOnLog( msg:String ):Void
{
    // Show the message etc...
}
```

OffbeatClient.debug : Number

This property can be set to receive debug information from the OffbeatClient class. The debug feature can be used only in the Flash design environment. It produces debug messages by using the AS trace function. The default value of the property is 0 (no debugging). The debug can be set to 1 and 2 to receive debug information. The debug level 1 shows only basic information about the events, but the debug level 2 shows also the data that is handled.

OffbeatClient.filePrepend : String

A property that can be used to add text in front of all filenames that are used in the requests. This is handy when the application directory may change. The following example shows how to add a path to all requests:

```
myConn.filePrepend = "myAppDirectory/";
// Request that will call file "myAppDirectory/myFile.xma"
myRequest = myConn.newRequest( myCbk, "myFile.xma" );
```

XMLRequest.setVar(name:String, value:String) : Void

The method can be used to set request variables. The request variables can be read by the server application. The following example shows how to send information about a person to the server application:

```
myRequest = myConn.newRequest( myCallback, "SavePerson.xma" );
// Set variables
myRequest.setVar( "first_name", "John" );
myRequest.setVar( "last_name", "Doe" );
// Send
myConn.send( myRequest );
```

Parameters:

name: The name of the request variable to set

value: Value of the request variable to set

Returns:

Nothing

Version history

Version	Date	Author	Description
1.0	7.1.2004	Kai Hannonen	First version
1.1	9.9.2004	Kai Hannonen	Changed the main class name to OffbeatClient (used to be XMLClient). Changed the OffbeatClient.newRequest description (the parameter was changed from String to Function type).