flashserver External for Max/MSP

Version 1.1

Written by Olaf Matthes.
Copyright © 2002-2006 Olaf Matthes – All rights reserved.

Content

Overview

The goal of this document is to describe the features and possibilities that the flashserver external adds to the Max/MSP environment and Macromedia Flash. This document assumes some familiarity with Max from a user's standpoint as well as knowledge of Macromedia Flash and it's ActionScript language.

The basic functionality of flashserver is to establish a connection between Macromedia's Flash - be it a standalone projector or a web Plug-In - to communicate with Max/MSP. The TCP/IP socket connection that gets created allows to exchange data between both programs in either direction over local network or the internet. It thus allows to build interactive web front-ends to Max/MSP or Max-controlled animations in Flash.

Due to it's capability of relaying data sent from one client to all other connected clients, even without any deeper Max knowledge, flashserver can be useful when building multi-user web interfaces. Using the `broadcast` message to send data to all connected clients from within Max/MSP allows to set up a system where for example the internal state of a Max patch is displayed to all connected users.

In the past years flashserver has proven it's stability and usefulness in several online projects. Among these are Pall Thayer's PANSE and the Streaps online mixing tool for internet broadcasts.
An installation made by Staalplaat Soundsystem used flashserver to control the state of a Flash-animated robot displayed to the user on a 42 inch plasma screen – that was mounted inside a man-sized fridge – at the European Media Art Festival 2004 in Osnabrück.

Thanks

For all the help and support in getting flashserver developed to it's current state I would like to thank August Black, Atau Tanaka, Pall Thayer and Oliver Thuns. Initial inspiration came from Eric Skogen.

Thanks for providing the Flash MX 2004 example component goes to Leo van der Veen from Collab.

The flashserver External

Creating the Object

Technically speaking, flashserver opens a listening network socket on the port specified as first creation argument and allows as many connections from Flash clients as the second argument specifies. The default looks like this:

When no creation arguments are specified, the default port number and number of clients that are shown above will be used. The maximum number of possible clients is 256 and the port number has to be greater or equal to 1024 to avoid collision with ports used by the operating system.

On systems that have a firewall installed (like Windows XP SP2) you have to poke a hole into the firewall for the port you are going to use in order to be able to connect to your machine over the network.

Inlets and Outlets

As the above image shows, flashserver has one inlet and four outlets. The inlet receives control messages discussed in the next section.

The outlets – from left to right – have the following functions:

- (anything) Received data from clients.
- (int) Number of currently connected clients.
- (int) Number of client data was received from last.
- (symbol) This client's IP or hostname.

The client numbers will be in range between 1 and the maximum number of clients set on object creation. However, in case two clients are connected and client number 1 disconnects, a newly connecting client will be client number 1 and client number 2 will keep it's number.

Supported Methods

Supported methods:

`send <client> <argument> (<more args>)`

Sends the specified argument(s) to the client specified as first argument. The client number is the number that can be obtained from the third outlet or using *prepend mode* (see below).

`broadcast <argument> (<more args>)`

The broadcast message is used to send data to all connected clients.

`prepend <state>`

The prepend message followed by either 1 or 0 turns the prepend mode on or off (off by default). When prepend is enabled, flashserver prepends the client number to the received data on output. This feature can be used to route data (using the route object) depending on which client was sending it.

`relay <state>`

The relay message followed by either 1 or 0 turns the relay mode on or off (off by default). When relay is enabled, flashserver relays all incoming data to all connected clients except the client that was sending it. This mode allows to distribute data generated by one user to all other connected users without having to write a complicated Max patch for it. Additionally, it also routes data that flashserver does not understand and can thus be used to relay 'hidden' information on top of the communication between Flash and Max.

`remote <state>`

The remote message followed by either 1 or 0 turns the remote mode on or off (off by default). When remote is enabled, flashserver allows to directly send values from Flash to specified receive objects in Max.
This feature also allows to control Max/MSP by sending data to 'max', as one would normally do in Max using a message `; max paths` or similar.

`kick <client>`

The kick message followed by a client number 'kicks' the specified client, i.e. it closes the connection to it and frees the client number to allow for a new connection. This will call the `XMLSocket.onClose` event handler in the Flash client.

`warn <state>`

The warn message followed by either 1 or 0 switches warnings (that get printed to the Max window) on or off. The default is on.

`block <mode>`

The block message followed by either 'wait', 'ignore' or 'kick' (or 0, 1 or 2) determines what to do in case client is not ready to receive any data. The default is 'wait' which might block flashserver until the client accepts the data (a warning will be printed to the Max window saying that flashserver blocked for a certain time).
With 'ignore' set, blocked clients get ignored and no data is send. With 'kick' the blocked client gets kicked and thus removed from the client list.

`interval <ms>`

The interval message followed by a time in milliseconds determines how often flashserver polls the network for new input. Please note that higher values give higher responsiveness but also increase the CPU usage. The default value is 5 ms.

`print`

The print message prints a list of all connected clients containing the client number, hostname and socket number to the Max window. It also prints the state of some of flashserver's internal values.

`open <filename>`

The open message – which is currently only supported on Windows – allows to execute any executable file from local disk. It can be used to open a Flash projector from within the Max patch. Executing Flash Movies (*.swf) is not possible.

Examples

In this section we'll show some examples, how data exchange between Flash and Max/MSP actually works. The examples provided require at least Macromedia Flash 5.0.

The basic functionality of flashserver is to establish a connection between Flash – be it a standalone projector or a web Plug-In – to communicate with Max/MSP. The TCP/IP socket connection that gets created by Flash allows to exchange data between both programs in either direction over local network or the internet. It thus allows to build interactive web front-ends to Max/MSP or Max-controlled animations in Flash.

ActionScript and Network Sockets

The basic building block for network communication in Flash is the `XMLSocket()` object introduced in Flash 5.0. It allows to establish a network connection to a server and to handle the data traffic on this connection.

A simple ActionScript example for establishing a connection to the flashserver help patch running on the same machine as the Flash movie would look like this:

```
max = new XMLSocket();
max.connect("localhost", 31337);      /* establish a connection */
max.onConnect = onMaxConnect;         /* function to be called when
                                         the connection has been made */
max.onClose = onMaxClose;             /* function to be called when
                                         the connection closes */
max.onData = onMaxData;               /* function to be called when
                                         some data arrives */


function onMaxConnect (success) {
          if (success) {
               msg = "Connection established!";
          } else {
               msg = "Connection failed!";
          }
}

function onMaxClose () {
          msg = "Lost connection to server!";
}

function onMaxData (doc) {
          msg = "Received data from server: " + doc;
}
```

Here, the connection is made to "localhost" which is the computer the Flash movie is run on. Port number is 31337 which is the default for flashserver. To connect to another machine, the hostname 'localhost' could be replaced with any other hostname or an IP address.

Note: due to security restrictions in Flash, the XMLSocket() method can connect only to TCP port numbers greater than or equal to 1024. Also, the XMLSocket() method can connect only to computers in the same subdomain where the SWF file (movie) resides. This restriction does not apply to movies running off a local disk.
This means that if you're planning to connect over the internet you have to run Max/MSP and flashserver on a machine that is on the same subnet as the web server that serves the SWF file.

A new feature from version 1.1test4 on allows the use of crossdomain.xml files to specify security features. You can use such a file (just place one in the Max search path) to define which IP addresses to allow to connect to flashserver. Check the supplied file for more details. By using such a file the above mentioned descurity restrictions can be avoided. To tell Flash to request the security settings add the Actions Script line

```
System.security.loadPolicyFile("xmlsocket://flashserverhost.com:31337");
```

to your code before actually opening the XMLSocket.

After establishing the connection the code example defines some event handlers that get called by Flash when the connection has been made, closes or some data arrives. Here, we just change a text that could be used to inform the user about the state of the connection.

Sending from Flash to Max/MSP

To send any data from Flash to Max/MSP we use the send() method of the XMLSocket() object. The data format is plain text as one would write it into a message box in Max/MSP. However, to send data that flashserver can understand, it has to end with a semicolon. The following example shows how to send a simple text string to Max/MSP:

```
max.send("hello 23 foo 17.3;");
```

max is the XMLSocket() object we already created in the last example. The text string "hello 23 foo 17.3;" will be output through flashserver's leftmost outlet as a list containing (symbol) 'hello', (int) 23, (symbol) 'foo' and finally (float) 17.3.

Since the data is parsed by Max/MSP and output as a list, it has to conform to certain Max standards. For example, semicolons or commas as part of the text have to be escaped using a backslash, like `"hello\, foo;"`. Placeholders that can be used in Max message boxes (like #1 or $1) are not allowed.

When ‘remote mode’ is activated it is additionally possible to send data to specified receive objects in the Max patch. The above method works unchanged, additionally the following code can be used:

```
max.send("; bla hello 23 foo 17.3;");
```

Here,the list `"hello 23 foo 17.3"` comes out the outlet of any **receive** object that has the name ‘bla’.

The syntax in Flash follows the normal Max syntax: when a semicolon (;) appears in a messagebox the first word after the semicolon is interpreted as the name of a receive object. The rest of the message (in Max) or the string (in Flash) is sent to all receive objects with that name.
If no arguments follow the receive name, a bang will come out the outlet of any **receive** object that has that name.

In case no matching receive object exists in your patch the data will be dropped without notice. In case remote mode is turned off (which is the default) data will be dropped as well.

Parsing Data received from Max/MSP

When data is sent from Max/MSP to Flash, it has the same plain text format as the strings we can send to Max/MSP from within Flash. To brake such a message down into a set of usable arguments the code shown below has proven quite useful:

```
function onMaxData(doc)
{
     argv = [];                    // create a new array
     doc = doc.substr(0, doc.length - 1);    // chop off ';' at the end
     argv = doc.split(" ");        // split at spaces and convert to array
     argc = argv.length;           // get number of elements in array

     for(i = 0; i < argc; i++)     // loop through all arguments
     {
          msg += "argument "+ i +" is "+ argv[i];
     }
}
```

In case an argument is know to be of integer type, the Flash function `parseInt(argv[i]);` could be used to obtain the integer value.

Another useful code snippet is the one to navigate a MovieClip to a given position. Imagine you have a MovieClip called *clip1* that has several frames. The message `"clip1 23"` sent from Max/MSP would now cause this MovieClip to start playing from frame number 23:

```
clip = argv[0];                  // get name of MovieClip
frame = argv[1];                 // get the frame number
_root[clip].gotoAndPlay(frame);
```

These two simple examples should give you an impression how to control Flash using flahserver. Experienced Flash users will of course be able to find several other ways to get Flash controlled.

Installation

System Requirements

This software is for use with Max/MSP 4.3 (and probably later). Separate versions are available for the Windows version and the Mac OS X version of Max/MSP. Max/MSP prior to version 4.3 or Max/MSP on Mac OS 9 is not supported.

The supplied Flash files are in Flash 5.0 format. However, some examples require Macromedia Flash MX or even Macromedia Flash MX 2004 because they make use of new features, like Flash Components.

The latest flashserver release can always be found at http://www.nullmedium.de/dev/flashserver/.

Installing flashserver

The installation of flashserver follows the standard Max/MSP ways. After extracting the archive, the object can be found in either the `build-mac` or `build-win` folders, depending on the operating system you're using.
There is a separate version for Max/MSP 4.5 available in the `build-max45` folder that was compiled using ProjectBuilder instead of Metrowerks CodeWarrior. It basically provides the same functionality as the 'traditional' version and is only provided for testing purposes. Further tests have to show whether it makes any difference performance-wise.

To permanently install flashserver, copy the object (flashserver or flashserver.mxe) from the corresponding build folder into the externals folder of your Max distribution (under 'Application Support' on Mac or 'Common Files' on Windows). Copy the help patch (flashserver.help) into the max-help folder of your Max distribution.

On systems that have a firewall installed (like Windows XP SP2) you have to poke a hole into the firewall for the port you are going to use in order to be able to connect to your machine over the network.

License

License Agreement

The Software is provided by Olaf Matthes (hereinafter referred to as the "AUTHOR"), free of charge and may be distributed free of charge, provided that this documentation is included unchanged with the software. You may not sell the software, nor may you take a fee or commission for providing the software to another person, nor may you include the software with or as part of other software that is sold for a fee without prior written permission from the Author. Source code for the software, if provided, can be re-used for educational or non-commercial purposes provided the Author is credited both in the product source and in the final product. The source code for this software may not be re-used for commercial purposes without negotiating and obtaining a commercial licensing agreement from the Author.
Currently, this software needs to be registered with the author, Olaf Matthes, to be used. Use of this program is illegal if you are not registered. Please note, that currently this program can only be used for research and experimental purposes and not for commercial use of any kind. If this program is used without registration or after a erasure request issued by the Author, the user is responsible for the illegal use of the program and agrees that any legal actions against the Author will be redirected to himself (the user) without contradiction.

Disclaimer of Warranty on Software

The software is provided "AS IS" and without warranty of any kind. The Author expressly disclaims all warranties, express or implied, including, but not limited to, the fitness for a particular purpose. The Author does not warrant that the software will be uninterrupted or error-free, or that defects in the software will be corrected. Furthermore, The Author does not warrant or make any representations regarding the use or the results of the use of the software in terms of their correctness, accuracy, reliability, or otherwise. No oral or written information or advice given by the Author shall create a warranty or in any way increase the scope of this warranty. Should the software prove defective, you (and not the author) assume the entire cost of all necessary servicing, repair or correction of the software, or the computer system with which it is used.

Limitation of Liability

Under no circumstances, including negligence, shall the Author be liable for any incidental, special, or consequential damages that result from the use or inability to use the software, even if the Author has been advised of the possibility of such damages.