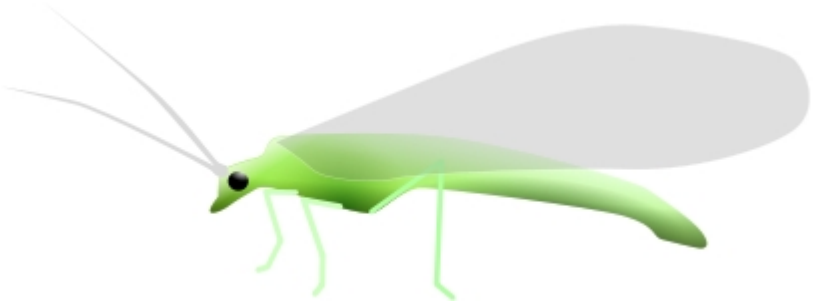
Lacewing Help

Introduction



Lacewing is a set of networking extensions (and also a C/C++/JS library) designed to replace the outdated MOO objects and provide users with a simple way to network their games and applications. It has its own IRC-like relay protocol, and also a webserver. The man behind the magic is none other than Clickteam's own Jamie McLaughlin, with Mathias Kærlev as support mainly for the Python implementation (pylacewing). This documentation is designed for Lacewing Build #20, but the core concepts and most of the actions/conditions/expressions should remain relevant for later versions.

A brief history

Lacewing started out as OINC (**O**INC **I**nternet/**N**etwork **C**ontroller), a replacement for MOO. As OINC was being developed however, Jamie experienced a hard drive crash and lost the source to OINC, forcing him to revert to an older version. At this point he renamed it Lacewing and created the liblacewing library that the client extension would use. There are now three parts to the Lacewing extension set; the Relay Client, the Relay Server, and the Webserver.

The Lacewing Relay Protocol

The Relay Client and Relay Server use the Lacewing Relay Protocol to communicate - it is an IRC-like communication protocol that allows for clients to connect to a server, have names, join channels, send messages to the channel to have them relayed to other peers in the channel, send messages to the server, and send messages to other peers in the channel. It also has support for each channel to have at most one channel master, which may optionally close the channel when they leave. Clients connected to a server may have the same name as other clients, so long as they are not in the same channel.

The main different between IRC and the LRP is that in IRC peer messages are per-server, and in LRP peer messages are per channel. Another difference is that there are three types of messages to be sent in the LRP; Text, Number, and Binary messages. messages sent to a channel are associated with a specific subchannel - channels have names, but subchannels are just numbers 0 through 255.

Core Concepts

To begin with, there are a few core concepts that you should understand to use Lacewing the intended way with ease. There are some key differences between MOO and Lacewing, such as when to set your name, that you need to understand. Make sure you understand channel/peer selection as well as the general concept of any given relay server.

Reading this Documentation

You may notice in some places, namely conditions and expressions, that at the beginning of the first paragraph there is a letter. This letter signifies a property of that condition or expression. For conditions you may see more than one letter.

- I - Immediate Condition
- N - Negatable Condition
- S - String-Returning Expression
- N - Number-Returning Expression

Concepts

Starting off

To begin with, it is important to understand how to connect to a server, set your name, and join channels. The

order is important: you must first connect to a server, then you must set your name. Only after you have set your name may you join a channel, request a channel list, or communicate with the server. Remember that connecting may not work and will give an error, the same applies to setting your name and joining channels.

Channel/Peer Selection

Just as MMF2 has object selection, Lacewing has peer and channel selection. You can't select yourself (just as you do not receive your own messages), so be careful when using the action to select the channel master - there is a condition to see if you are the channel master. Conditions that are fired (such as On Message, On Peer Join, etc, as well as loops) will automatically change your currently selected channel and, for peer related conditions, your currently selected peer. You can be connected to multiple channels at once, so it is important to monitor which channel messages are coming from by comparing to the selected channel expression. For this reason, it is important to first select the channel you will be dealing with right before you deal with it - the same applies to peers. You can also select peers by their ID, which is useful for games - you can store a peer's ID with their associated active object. If you have an ID or Name of a peer, you can get the other by selecting the peer and then using the expression you want.

Subchannels

Subchannels are integral values associated with each message you send, be it to the server, to a channel, or to a peer. The value can be any number in the range 0 to 255, inclusive, and is used per your standards to (often times) differentiate different kinds of information you will be sending. For example, you may use subchannel 0 for regular messages, subchannel 1 for IRC-styled /me messages, and subchannel 200 for commands. It is easiest to think of subchannels as literally being channels within channels, but having a number for a name instead of a string. There are 256 subchannels, so you shouldn't run out any time soon - especially since there are three kinds of messages.

Types of messages

In Lacewing there are three different kinds of messages that you can send to the server, to the channel, or to peers. The most common kind suitable to chat applications is the Text message, which is literally just plain text. The second type which is less commonly used is the Number message, which sends only an integer number in the same range that MMF2 supports. It cannot send floating-point values, such as 2.5. The third type of message, most commonly used for games, the the Binary message. You can send any combination of any kind of data you want, including strings, different types of numbers, files, or any data you want. This is often used to send things like X,Y coordinate pairs, or even X, Y, Angle, Velocity, Animation, Animation Frame, etc., which is why it is most suitable to games. If you want, you can also use it as a second half of a subchannel by using it to mimic a text message or number message.

Servers

A **server** is the hub of any network related activity, as clients usually connect to a single server to perform networking tasks. Servers cannot connect to clients or force them to connect. Once connected, a client is under full control by the server to the extent of the protocol. In Lacewing, this means that a server can disconnect you, change your name, change channel names, change the channel master, and send messages to you or to channels. Servers can (rarely) disallow you from leaving channels, but they cannot disallow you from disconnecting. Servers can force you to leave channels, but cannot force you to join channels. Servers are also responsible for generating and sending the channel list upon your request.

Channels

A **channel** is an isolated 'room' where peers can send messages to other peers in the channel. Peer messages function on a per-channel basis. Channels have two properties: they can be forced to close when the channel master leaves, and they can be hidden from their point of creation. Hidden channels simply do not get listed with the channel list. You do not have to be connected to any channels to send messages to the server, however you can be connected to multiple channels at once.

Attempting to join a channel that does not exist results in the creation of that channel - this is the *only* time that the hidden and close on leave properties are modified. At this point you become the channel master, which is simply an automatically selected client to be treated as the channel controller. There can either be one channel master or no channel master, depending on whether the channel is set to close when the channel master leaves. Other than this, the channel master is just a normal peer. The only other time a channel is closed when a peer leaves is if that peer is the last peer in that channel, at which point the channel is destroyed.

Channels break one rule of selection: when the On Leave Channel condition is triggered, you are not in the channel, meaning you cannot determine the name of the channel you left - this is most likely a flaw. This is contrary to the On Peer Disconnect condition, where the peer still exists and you can access the peer's name and ID.

Peers

A **peer** is a client in a server. Peers can only interact with each other when they are in channels. Peers control everything that the server does not - for instance, in a multiplayer game, peers are the people playing the game and connecting to the same server to play together. In Lacewing's case, multiple games on the same server each occur within their own channel. Peers can also send messages to each other in private, but only via Peer A -> Server -> Peer B. Two or more peers may have the same name so long as they are not in the same channel and do not enter the same channel. Peers are usually identified by a unique ID number, so having two peers with the same name is no problem, because peers cannot interact outside of channels.

While the server is responsible for simulating games to keep cheating to a minimum, you (you are a client/peer) should only process your own movement and information, and only display other peers. It is important to remember to think of yourself as one peer interacting with others when programming with Lacewing. Peers should never take it upon themselves to fill in the role of the server (the channel master is an exception), and peers should not be held responsible for leaving channels themselves when being or forced to do other actions. For applications, it is up to you to decide how peers should interact with each other and the server.

Clients

"**Client**" is just another name for "Peer". From the server's point of view, Peers are its Clients, and from each Client's point of view, Clients are Peers. You should consider yourself as a client rather than a peer, however.

Channel Masters

The **Channel Master** is the peer designated as, well, the master of a channel. The channel master is a peer/client like any other, and special treatment and/or special behavior should be decided upon by you, the programmer. If you want to take advantage of the channel master, design your events to handle both the case when you are the channel master and when you are not. Most events will be the same either way and do not need to check this. There can be either one channel master or no channel master per channel, and the server may change the channel master. Channels may close when the channel master leaves, or they may stay open in which case there will simply be no channel master. Be careful in your events: you cannot select yourself, so check to see if you are the channel master before selecting the channel master.

Binary messages

Based on an article from Jamie.

Binary messages (previously called Stacks) are mixed-content messages. They can contain any combination of, and in an order, strings, numbers, files, some other form of data, etc. Because you design the structure of the messages yourself, you can, for example, use them to transfer the state of an object. For example, instead of using Text messages and separating different parts of the message with a token (such as a comma (,) or a pipe(|)), you can use binary messages to send them in a more condensed format.

Consider sending an X-Y coordinate pair:
104,178
As a string, this is 7 characters in length, plus a null termination character to mark the end of the string, meaning that you will be sending 8 whole bytes for every movement!

Bytes	49	48	52	44	49	55	56
Text	1	0	4	,	1	7	8

Using a stack message, however, you can condense the message size to 4 or even 2 bytes:

```
104 178
```

In the above example, two bytes (104 and 178) are sent, rather than 8. This is a huge bandwidth saver, as it is four times smaller than the Text message. This is important for fighting lag in games, as movement happens (usually) 60 times per second.

Using binary messages

A binary message must be created before it can be sent. To do this, you 'push' bytes onto the end of the message. The bytes can make up strings, numbers, or files. For numbers, there are different data types that use up a different number of bytes:

```
Byte (takes up 1 byte)

  Range when signed
    -128 to 127

  Range when unsigned
     0 to 255

  Does not allow decimals.


Short (takes up 2 bytes)

  Range when signed
    -32768 to 32767

  Range when unsigned
     0 to 65535

  Does not allow decimals.


Integer (takes up 4 bytes)

  Range when signed
     -2147483648 to 2147483647

  Range when unsigned
      0 to 4294967295

  Does not allow decimals.


Float (takes up 4 bytes)

  Range
    3.4e +/- 38

  Allows decimals.
```

Building a binary message is easy, but reading it can be slightly harder. Indexes start at 0, meaning the first byte in a binary message is byte number 0, the second is byte number 1, the third is 2, etc. Consider a binary message comprised of the frame of an animation (eg 3), the size of a circle (eg 572), the score (eg 31333337), and the percent completion of the level (eg 100):

```
ACTIONS                 STACK CONTENT
=======                 =============

Push byte 3             0           3           (byte)

Push short 572          0           3           (byte)
                        1           572         (short)
                        2

Push integer 31333337   0           3           (byte)
                        1           572         (short)
                        2
                        3           31333337    (integer)
                        4
                        5
                        6

Push byte 100           0           3           (byte)
                        1           572         (short)
                        2
                        3           31333337    (integer)
                        4
                        5
                        6
                        7           100         (byte)
```

To read the stack upon receiving it, you would get the animation frame by getting the unsigned byte at 0, then a byte later at byte 1 you would read the size of the circle as an unsigned short. Remember though that a short takes up two bytes of space, so you would read the score as a signed integer two bytes later at byte 3. An integer is four bytes, so four bytes later at byte 7 you could read the unsigned (or signed) byte that represents the percent progress through the level. That makes all eight bytes accounted for. If you had sent this as a text message: "3,572,31333337,100" that would be 18+1 bytes, making for a whopping 19 byte message! That's over twice the 8 byte binary message!

You may now wonder why you can't choose to send a signed or unsigned type, but you have to choose when reading a number. For any given type, signed and unsigned take the same amount of of memory and, in fact, are represented exactly the same way! because the data is the same, the only reason to interpret it as signed (having a positive or negative sign) or unsigned (only being positive) is whether you want to be able to have it negative. As seen above, an unsigned byte has 256 different values, 0 to 255. A signed by *also* has 256 different values, -128 through 127. This is because the signed byte takes the unsigned byte's range 128 to 255 and uses it instead for -128 to -1. This means that unsigned 128 is the same as -128, 129 is the same as -127, and 255 is th same as -1.

There is an oddity in Lacewing: there exists and expression to read an unsigned integer. However, MMF2 can only process at most signed integers or double precision floats. This means that, while Lacewing may give MMF2 the data and expect it to interpret is in unsigned form, MMF2 interprets it in signed form instead. So, don't use this expression - it may be removed eventually. It is currently provided only for consistency and legacy.

Sending verus Blasting

Sending a message uses TCP, whereas **Blasting** a message uses UDP. Where TCP messages are guaranteed to make it to their destination alive and in the same order they were sent, UDP messages are not. Sometimes, a UDP message can be dropped, or it can arrive out of order. So why would anyone use such a protocol? Because it is much faster than TCP is. Generally you would Send a message with TCP if it was a chat message, whereas you would always want to Blast messages with UDP if they were position updates for an object. Object positions need to update quickly for the game to be playable. If a message is dropped or sent out of order, it will often be fixed in no time because you are sending so many so fast. If you decide to only send messages when said object moves, you should take precaution and make sure the last message is Sent and not Blasted to guarantee that it is in the correct spot when standing still.

Multiple channels

While connected to a server, it is possible to be in multiple channels at the same time. An example of the usage of this is an MDI chat, similar to our well known Gwerdy chat (which uses MOO unfortunately). It is important to both set and check which channel a message is going to/coming from, making multiple channels one of the more confusing concepts to utilize. Another problem is that when leaving a channel, you cannot know which channel has been left, as the

channel is destroyed and thus not selected for the On Leave Channel condition.

Prefixing channel names

When dealing with games and software that will utilize multiple channels on a server, it is important to prefix channel names with a unique string on a per game/application basis. For instance, instead of "Lobby" for a channel name, which could be use by many different games and applications on a given server, it would be best to prefix the name: "[My Game/Software] Lobby". The prefix can also be used to filter channels from the channel list to show only channels related to your game or software. You can, of course, always remove the prefixes from start of the channel name so that the end user does not know anything about the magic in the background.

Channel Listing

Channel lists are the list of non-hidden channels currently on the server, including each channel's name and peer count. Peers can request a channel list from the server at any time.

Examples

Lace Chat (original forum link)

This example will only work properly in MMF2 Developer. It shows how to make a Gwerdy-like MDI chat with Lacewing.

Bugs

Bugs

- In the Relay Client, there is no way to know which channel has been left without keeping track of all channels manually;
- Both the Relay Client and Relay Server extensions have expressions to get the Unsigned form of a 32-bit integer from a binary message, which MMF2 promptly truncates to a Signed 32-bit integer;
- In the Relay Server extension, there is no way to loop the clients in a channel with a loop name despite there being conditions for such a thing;
- In the Relay Server extension, the On <...> message to peer sent/blasted, it is ambiguous as to whether the to client or from client is selected, and there does not appear to be a way to determine the other client;
- Lacewing Relay Server->Conditions->Channel->Client->On channel clients loop/finished (obvious copy paste error);
- Lacewing Relay Server->Expressions->Client->Connection time (removed expression, menu item left in);
- Lacewing Relay Server->Expressions->Client->Get client protocol implementation (removed expression, menu item left in);
- In the Webserver object, there is no way to set local connection data, despite there being expressions to retrieve such data;
- Lacewing Webserver->Expressions->Guess MIME type for filename (parameter list is incorrect);

Extension Set

The **Lacewing** set of extensions allows you to easily create advanced networked applications and games. The three extensions currently in the set include the Relay Client, the Relay Server, and the Webserver.

Lacewing Relay Client

The **Lacewing Relay Client** is the main object of the Lacewing extension set, as it allows you to connect to a server, join channels, and interact with peers. Beyond that, it is up to you, the programmer, to design the way that peers should interact with each other and the server, be it for a game or for an application such as a chat program. Think of the Relay Client as representing your application, interacting with other instances of the same application. You shouldn't design programs with different code for different peers (eg: don't design separate sender and receiver programs) - instead, design your events such that you treat yourself as a peer equal to all other peers.

The Lacewing Relay Client is the only part of Lacewing that has been ported to Flash (AS3).

Actions

Connect

This action is used to start the connection process and attempt to connect to a server. This action takes one string parameter, which is the server (and optionally the port) to connect to. Here are some examples:

- "ledev.org" - only the server host name is specified, port 6121 will be used by default
- "ledev.org:6124" - both the server host name and port are specified, connection will occur on port 6124
- "127.0.0.1" - an IP address is specified, default port 6121 will be used.
- "localhost" - same as "127.0.0.1"

Disconnect

This action terminates an active connection. It does not take any parameters. You literally disconnect from the server, leaving any channels you were in, meaning that if you were the channel master and you chose to close the channel when you leave, the channels will close. Channels will also close if you were the last peer in that channel.

Set Name

This action is used to initially set your name or to change it after it has been set. It takes one string parameter, which is the new name you want to set for yourself. The server may deny the name or change it, so it is important to use the Self Name expression rather than expecting to have the name you requested. Names may have any characters (including spaces and new lines) and my be any length other than zero. Two clients on a server may have the exact same name at the same time as long as they are not in and do not enter the same channels. Names maintain their case but are case insensitive. Thus, three users "Jamie", "jamie", and "jamie" can be on the same server but not the same channels.

Channel Listing

Request list

This action requests the list of public channels from the server. See Channel Listing for more information.

Loop listed channels

This action loops through the last received channel list, but does not have a loop name for efficiency (and legacy).

See Channel Listing for more information.

Loop listed channels (with loop name)

This action is the same as Loop listed channels, but uses a loop name and fires the related events. It takes one string parameter, which is the name of the loop. The name can be anything at all, including a blank string "", and may contain anything at all, including new lines. See Channel Listing for more information.

Channel

Select

By name

This action allows you to select a channel you are currently in by specifying its name. It takes one string parameter, which is the name of the channel you want to select.

Loop

This action starts a nameless loop of all the channels you are currently in.

Loop (with loop name)

This action is the same as Loop Channels, but allows you to specify a name for the loop. It takes one string parameter, which is the name of the loop. The name can be anything at all, including a blank string "", and may contain anything at all, including new lines.

Join

This action allows you to join channels, which are the only places where you can communicate with other peers. It takes, in order, a string parameter that is the name of the channel (which can be anything), then a number which, when 0 means public channel, and when 1 means private channel, and another number which, when 0 means to keep the channel open if the channel master leaves, and when 1 means the close the channel if the channel master leaves. The last two parameters only apply if the channel does not yet exist, in which case it is created and you automatically become the channel master. The server may change which peer in a channel is the channel master.

Leave

This action is used to leave a channel. It takes not parameters; the channel it will request to leave is the currently selected channel. Normal servers will not deny channel leaving, only customized servers may deny this. It is always possible to leave a channel you are not allowed to leave by disconnecting - there is no way for a server to prevent you from disconnecting.

Peer

Select by name

This action, similar to selecting a channel by name, selects a peer in the current channel by their name. it takes one parameter, which is the name of the peer you want to select.

Select by ID

This action, similar to selecting a peer by name, selects a peer in the current channel by their ID. It takes one parameter, which is the ID of the peer you want to select.

Select the channel master

This action selects the channel master of the current channel. Some channels may not have a channel master, in which case the channel master cannot be selected. Also, it is possible that you are the channel master, and because you cannot select yourself, this also means you cannot select the channel master. This action takes no parameters.

Loop

This action loops through all the peers in the currently selected channel. This action takes no parameters.

Loop (with loop name)

This action is the same as loop peers in channel, but it allows you to specify a loop name. It takes one string parameter, which is the name of the loop. The name can be anything at all, including a blank string "", and may contain anything at all, including new lines.

Send

Text

To server

This action allows you to send text to the server. It takes two parameters, the first being the subchannel to send the message on, and the second being the text itself to send. Only customized servers will have any reaction to this, as by default a normal server just ignores received messages.

To channel

This action allows you to send text to the currently selected channel. It takes two parameters, the first being the subchannel to send the message on, and the second being the text itself to send. The server, if it chooses, will then relay the message to the other peers in the channel. You will not receive your own message.

To peer

This action allows you to send text to the currently selected peer in the currently selected channel. It takes two parameters, the first being the subchannel to send the message on, and the second being the text itself to send. The server, if it chooses, will send the message to the peer. The server may change the To and From peers for the message if it is a custom server.

Number

To server

This action allows you to send a number to the server. It takes two parameters, the first being the subchannel to send the message on, and the second being the number itself to send. Only customized servers will have any reaction to this, as by default a normal server just ignores received messages.

To channel

This action allows you to send a number to the currently selected channel. It takes two parameters, the first being the subchannel to send the message on, and the second being the number itself to send. The server, if it chooses, will then relay the message to the other peers in the channel. You will not receive your own message.

To peer

This action allows you to send a number to the currently selected peer in the currently selected channel. It takes two parameters, the first being the subchannel to send the message on, and the second being the number itself to send. The server, if it chooses, will send the message to the peer. The server may change the To and From peers for the message if it is a custom server.

Binary

To server

This action allows you to send a binary message to the server. It takes one parameter: the subchannel to send the message on. If you have Automatically clear binary checked, then the binary will be cleared after sending the message. Otherwise, the binary will stay for the next message. Only customized servers will have any reaction to this, as by default a normal server just ignores received messages.

To channel

This action allows you to send a binary message to the currently selected channel. It takes one parameter: the subchannel to send the message on. If you have Automatically clear binary checked, then the binary will be cleared after sending the message. Otherwise, the binary will stay for the next message. The server, if it chooses, will then relay the message to the other peers in the channel. You will not receive your own message.

To peer

This action allows you to send a binary message to the currently selected peer in the currently selected channel. It takes one parameter: the subchannel to send the message on. If you have Automatically clear binary checked, then the binary will be cleared after sending the message. Otherwise, the binary will stay for the next message. The server, if it chooses, will send the message to the peer. The server may change the To and From peers for the message if it is a custom server.

Blast

Text

To server

This action allows you to blast text to the server. It takes two parameters, the first being the subchannel to blast the message on, and the second being the text itself to blast. Only customized servers will have any reaction to this, as by default a normal server just ignores received messages.

To channel

This action allows you to blast text to the currently selected channel. It takes two parameters, the first being the subchannel to blast the message on, and the second being the text itself to blast. The server, if it chooses, will then relay the message to the other peers in the channel. You will not receive your own message.

To peer

This action allows you to blast text to the currently selected peer in the currently selected channel. It takes two parameters, the first being the subchannel to blast the message on, and the second being the text itself to blast. The server, if it chooses, will blast the message to the peer. The server may change the To and From peers for the message if it is a custom server.

Number

To server

This action allows you to blast a number to the server. It takes two parameters, the first being the subchannel to blast the message on, and the second being the number itself to blast. Only customized servers will have any reaction to this, as by default a normal server just ignores received messages.

To channel

This action allows you to blast a number to the currently selected channel. It takes two parameters, the first being the subchannel to blast the message on, and the second being the number itself to blast. The server, if it chooses, will then relay the message to the other peers in the channel. You will not receive your own message.

To peer

This action allows you to blast a number to the currently selected peer in the currently selected channel. It takes two parameters, the first being the subchannel to blast the message on, and the second being the number itself to blast. The server, if it chooses, will blast the message to the peer. The server may change the To and From peers for the message if it is a custom server.

Binary

To server

This action allows you to blast a binary message to the server. It takes one parameter: the subchannel to blast the message on. If you have Automatically clear binary checked, then the binary will be cleared after blasting the message. Otherwise, the binary will stay for the next message. Only customized servers will have any reaction to this, as by default a normal server just ignores received messages.

To channel

This action allows you to blast a binary message to the currently selected channel. It takes one parameter: the subchannel to blast the message on. If you have Automatically clear binary checked, then the binary will be cleared after blasting the message. Otherwise, the binary will stay for the next message. The server, if it chooses, will then relay the message to the other peers in the channel. You will not receive your own message.

To peer

This action allows you to blast a binary message to the currently selected peer in the currently selected channel. It takes one parameter: the subchannel to blast the message on. If you have Automatically clear binary checked, then the binary will be cleared after blasting the message. Otherwise, the binary will stay for the next message. The server, if it chooses, will blast the message to the peer. The server may change the To and From peers for the message if it is a custom server.

Binary to send

Add byte

ASCII character

This action adds one byte to the end of the current binary message. It takes one sting parameter, which is the character you want to add. You should only put one character in the string, eg "c". If you want to add multiple characters at once, use one of the Add string actions instead.

Integer value

This action adds one byte to the end of the current binary message. It takes one parameter: the integral value of the byte you want to add. It does not matter if it is signed or unsigned, as the binary representations are the same. Remember that the range of a byte is -128 to 127 or 0 to 255, depending on how it is interpreted by the receiver.

Add short

This action adds two bytes to the end of the current binary message. It takes one parameter: the integral value of the short you want to add. It does not matter if it is signed or unsigned, as the binary representations are the same. Remember that the range of a short is -32768 to 32767 or 0 to 65535, depending on how it is interpreted by the receiver.

Add integer

This action adds four bytes to the end of the current binary message. It takes one parameter: the integral value of the integer you want to add. It does not matter if it is signed or unsigned, as the binary representations are the same - additionally, MMF2 can only handle signed integers at most. Remember that the range of an integer is -2147483648 to 2147483647.

Add float

This action adds four bytes to the end of the current binary message. It takes one parameter: the floating point value of the float you want to add. MMF2 can handle double precision floats, which are eight bytes, but can only give extensions four bytes at most, which is why Lacewing only lets you send floats and not doubles. Remember that the range of a float is 3.4e-38 to 3.4e38.

Add string

Without null terminator

This action allows you to simply add a plain string to the end of the current binary data. The number of bytes added is literally the length of the string in characters. It takes one parameter: the string to add.

With null terminator

This action allows you to add a null-terminated string to the end of the current binary data. The number of bytes added is the length of the string in characters plus one for the null character (byte value 0). It takes one parameter: the string to add.

Add binary

This action is for advanced usage; it takes two parameters: the location of binary in memory, and how much of it you want to add. The number of bytes added is literally the second parameter. This is useful in conjunction with objects such as the Binary object and/or the Memory object.

Add file

This action allows you to add an entire file to the end of the current binary data. It takes one parameter, which is the file to add. You can choose to use an expression to evaluate the file path. You shouldn't send large files all at once - even though you can send up to 2147483647 bytes (approx. 2 gig), it will most likely freeze the application or game for a long duration. If you intend to send large files, you should break the file up into smaller chunks and send them individually.

Compress (ZLIB)

This action compresses the binary using ZLIB compression routines. You should find a balance between faster send/blast time with smaller binary messages and faster message generation/interpreting time when not compressing/decompressing the binary message.

Clear

This action simply clears the binary to send and sets its size to 0. If you have Automatically clear binary checked, then this action will occur whenever you send/blast a binary message. Otherwise, it is up to you to decide when

to clear a binary message.

Resize

This action will resize the binary to send to whatever number of bytes you specify via the only parameter. This is useful in conjunction with the Binary to send address expression, as you can load binary into the existing memory though another object (such as Binary Quickload).

Received binary

Save to a file

This action allows you to save a portion of the received binary to a file, overwriting the file if it already exists. It takes three parameters, the first being the position in the received binary to be the start of the file, the second being how many bytes after that you want to save to the file, and the third being a string expression for the path to the file to save to.

Append to a file

This action allows you to save a portion of the received binary to the end of a file, creating the file if it does not exist. It takes three parameters, the first being the position in the received binary to be the start of the file, the second being how many bytes after that you want to save to the file, and the third being a string expression for the path to the file to save to.

Uncompress (ZLIB)

This action simply decompresses the received binary using ZLIB compression routines.

Move cursor

This action moves the cursor in the received binary for cursor related expressions. It takes one parameter: the new position for the cursor.

Conditions

On error

I This condition is triggered when an error occurs. You can use the Error string expression to get the error in string format. There are many things that can cause an error, but there has been no official public list.

Connection

On connect

I This condition is triggered as soon as connection is successful. You can use the Get welcome message expression to get the server's MOTD.

On connection denied

I This condition is triggered as soon as connection to a server has been denied. You can use the Deny reason expression to get the reason.

Is connected

N This condition is used to test if you are connected to a server or not.

On disconnect

This condition is triggered as soon as you have disconnected from a server, be it by your own events or the server terminating its connection with you.

Channel listing

On list received

This condition is triggered as soon as the channel list has been fully received. You can then start looping through with either the nameless or named loop actions.

On loop

This condition is triggered for each channel in the channel list. You can use the Get name and Get peer count expressions to get information about the channel.

On loop finished

This condition is triggered after the last channel in the channel list have been listed (via triggering the On loop condition). This is only for convenience, as it would have the same effect as putting events after the Loop listed channels action.

[With loop name] On loop

This condition is triggered for each channel in the channel list. You can use the Get name and Get peer count expressions to get information about the channel. This condition takes one string parameter, which is the name of the loop.

[With loop name] On loop finished

This condition is triggered after the last channel in the channel list have been listed (via triggering the On loop condition). This is only for convenience, as it would have the same effect as putting events after the Loop listed channels action. This condition takes one string parameter, which is the name of the loop.

Channel

On join

This condition is triggered as soon as you join a channel. You can start a peer loop on the channel to see what peers are already there.

On join denied

This condition is triggered as soon as joining a channel has been denied. You can use the Deny reason expression to get the reason. However, much like with the On leave condition, you cannot get the name of the channel.

On leave

This condition is triggered as soon as you leave a channel, be it by your own events or from the server kicking you from that channel. You cannot get the name of the channel you left, unfortunately, making this condition the arch nemesis of people in multiple channels at once. The only way to find out which channel you have left is to loop through all the channels you are in and see which is missing.

On leave deined

This condition is triggered as soon as connection to a server has been denied. You can use the Deny reason

expression to get the reason. Note that the default server behavior will never deny you leaving a channel, only customized servers may do this - thus, the only time you should worry about this condition is if you are designing the server with this behavior or if you know the server will do this.

On joined channel loop

This condition is triggered for each channel you are in when you use the Loop channels action. It changes the selected channel, so you can use the Channel name expression to get the name of the channel.

On joined channel loop finished

This condition is triggered when the joined channel loop is finished, after all channels have been iterated.

[With loop name] On joined channel loop

This condition is triggered for each channel you are in when you use the Loop channels action that uses a loop name. It changes the selected channel, so you can use the Channel name expression to get the name of the channel. It takes one string parameter, which is the name of the loop.

[With loop name] On joined channel loop finished

This condition is triggered when the joined channel loop is finished, after all channels have been iterated. It takes one string parameter, which is the name of the loop.

Peers

On peer connect

This condition is triggered when a peer joins any channel you are in. It automatically selects the channel and peer for you so you can get the channel name and peer name/ID.

On peer disconnect

This condition is triggered when a peer leaves any channel you are currently in. The channel and peer are selected for you so you can get the channel name and peer name/ID.

On peer changed name

This condition is triggered when a peer changes their name while they are in any channel that you are also in. The channel and peer are selected for you so you can get the channel name and the peer ID/old name/new name.

On loop

This condition is triggered for each peer in a channel when you use peer loops. The channel and peer are selected for you so you can get the channel name and peer name/ID.

On loop finished

This condition is triggered when all the peers in a channel have been iterated. The channel is selected for you so you can get the channel name. This is useful for adding yourself to the end of a user list.

[With loop name] On loop

This condition is triggered for each peer in a channel when you use peer loops. The channel and peer are selected for you so you can get the channel name and peer name/ID. It takes one parameter, which is the name of the loop.

[With loop name] On loop finished

This condition is triggered when all the peers in a channel have been iterated. The channel is selected for you so you can get the channel name. This is useful for adding yourself to the end of a user list. It takes one parameter, which is

the name of the loop.

Selected peer is the channel master

N This condition is true if the currently selected peer in the currently selected channel is the channel master.

You are the channel master

N This condition is true if you are the channel master of the currently selected channel.

Name

On name set

I This condition is triggered when your name is first set after previously not having a name.

On name deined

I This condition is triggered when your newly requested name is denied, either when you are first setting your name or when you are changing it. You can use the Deny reason expression to get the reason.

On name changed

I This condition is triggered when you have successfully changed your name, but not when you previously did not have a name.

Client has a name

N This condition is true when you have a name.

Message

Sent

On text message from server

To me

I This condition is triggered when the server sends a text message to you. It can happen when you aren't even in any channels. It takes one parameter, which is the subchannel to filter this event for.

To channel

I This condition is triggered when the server sends a text message to a channel that you are currently in. The channel is selected for you so that you can get the channel name. It takes one parameter, which is the subchannel to filter this event for.

On number message from server

To me

I This condition is triggered when the server sends a number message to you. It can happen when you aren't even

in any channels. It takes one parameter, which is the subchannel to filter this event for.

To channel

I This condition is triggered when the server sends a number message to a channel that you are currently in. The channel is selected for you so that you can get the channel name. It takes one parameter, which is the subchannel to filter this event for.

On binary message from server

To me

I This condition is triggered when the server sends a binary message to you. It can happen when you aren't even in any channels. It takes one parameter, which is the subchannel to filter this event for.

To channel

I This condition is triggered when the server sends a binary message to a channel that you are currently in. The channel is selected for you so that you can get the channel name. It takes one parameter, which is the subchannel to filter this event for.

On any message from server

To me

I This condition is triggered when the server sends any message to you. It can happen when you aren't even in any channels. It takes one parameter, which is the subchannel to filter this event for.

To channel

I This condition is triggered when the server sends any message to a channel that you are currently in. The channel is selected for you so that you can get the channel name. It takes one parameter, which is the subchannel to filter this event for.

On text message from channel

I This condition is triggered when a peer sends a text message to a channel you are currently in. The channel and peer are selected for you so you can get the channel name and peer name/ID. It takes one parameter, which is the subchannel to filter this event for.

On number message from channel

I This condition is triggered when a peer sends a number message to a channel you are currently in. The channel and peer are selected for you so you can get the channel name and peer name/ID. It takes one parameter, which is the subchannel to filter this event for.

On binary message from channel

I This condition is triggered when a peer sends a binary message to a channel you are currently in. The channel and peer are selected for you so you can get the channel name and peer name/ID. It takes one parameter, which is the subchannel to filter this event for.

On any message from channel

I This condition is triggered when a peer sends any message to a channel you are currently in. The channel and peer are selected for you so you can get the channel name and peer name/ID. It takes one parameter, which is the subchannel to filter this event for.

On text message from peer

I This condition is triggered when a peer sends a text message to you in a channel you are currently in. The channel and peer are selected for you so you can get the channel name and peer name/ID. It takes one parameter, which is the subchannel to filter this event for.

On number message from peer

I This condition is triggered when a peer sends a number message to you in a channel you are currently in. The channel and peer are selected for you so you can get the channel name and peer name/ID. It takes one parameter, which is the subchannel to filter this event for.

On binary message from peer

I This condition is triggered when a peer sends a binary message to you in a channel you are currently in. The channel and peer are selected for you so you can get the channel name and peer name/ID. It takes one parameter, which is the subchannel to filter this event for.

On any message from peer

I This condition is triggered when a peer sends any message to you in a channel you are currently in. The channel and peer are selected for you so you can get the channel name and peer name/ID. It takes one parameter, which is the subchannel to filter this event for.

Blasted

On text message from server

To me

I This condition is triggered when the server blasts a text message to you. It can happen when you aren't even in any channels. It takes one parameter, which is the subchannel to filter this event for.

To channel

I This condition is triggered when the server blasts a text message to a channel that you are currently in. The channel is selected for you so that you can get the channel name. It takes one parameter, which is the subchannel to filter this event for.

On number message from server

To me

I This condition is triggered when the server blasts a number message to you. It can happen when you aren't even in any channels. It takes one parameter, which is the subchannel to filter this event for.

To channel

I This condition is triggered when the server blasts a number message to a channel that you are currently in. The channel is selected for you so that you can get the channel name. It takes one parameter, which is the subchannel to filter this event for.

On binary message from server

To me

This condition is triggered when the server blasts a binary message to you. It can happen when you aren't even in any channels. It takes one parameter, which is the subchannel to filter this event for.

To channel

This condition is triggered when the server blasts a binary message to a channel that you are currently in. The channel is selected for you so that you can get the channel name. It takes one parameter, which is the subchannel to filter this event for.

On any message from server

To me

This condition is triggered when the server blasts any message to you. It can happen when you aren't even in any channels. It takes one parameter, which is the subchannel to filter this event for.

To channel

This condition is triggered when the server blasts any message to a channel that you are currently in. The channel is selected for you so that you can get the channel name. It takes one parameter, which is the subchannel to filter this event for.

On text message from channel

This condition is triggered when a peer blasts a text message to a channel you are currently in. The channel and peer are selected for you so you can get the channel name and peer name/ID. It takes one parameter, which is the subchannel to filter this event for.

On number message from channel

This condition is triggered when a peer blasts a number message to a channel you are currently in. The channel and peer are selected for you so you can get the channel name and peer name/ID. It takes one parameter, which is the subchannel to filter this event for.

On binary message from channel

This condition is triggered when a peer blasts a binary message to a channel you are currently in. The channel and peer are selected for you so you can get the channel name and peer name/ID. It takes one parameter, which is the subchannel to filter this event for.

On any message from channel

This condition is triggered when a peer blasts any message to a channel you are currently in. The channel and peer are selected for you so you can get the channel name and peer name/ID. It takes one parameter, which is the subchannel to filter this event for.

On text message from peer

This condition is triggered when a peer blasts a text message to you in a channel you are currently in. The channel and peer are selected for you so you can get the channel name and peer name/ID. It takes one parameter, which is the subchannel to filter this event for.

On number message from peer

This condition is triggered when a peer blasts a number message to you in a channel you are currently in. The channel and peer are selected for you so you can get the channel name and peer name/ID. It takes one parameter, which is the subchannel to filter this event for.

On binary message from peer

I This condition is triggered when a peer blasts a binary message to you in a channel you are currently in. The channel and peer are selected for you so you can get the channel name and peer name/ID. It takes one parameter, which is the subchannel to filter this event for.

On any message from peer

I This condition is triggered when a peer blasts any message to you in a channel you are currently in. The channel and peer are selected for you so you can get the channel name and peer name/ID. It takes one parameter, which is the subchannel to filter this event for.

Expressions

Error string (for on error)

S This expression returns the error message for the On error condition.

Lacewing version string

S This expression returns the version string of Lacewing. For example, this help file was created for the version of Lacewing "liblacewing #20 (Windows/x86)".

Binary to send size

N This expression returns the size in bytes of the binary to send.

Binary to send address

N This expression returns the memory address of the binary to send. This is commonly used on conjunction with objects such as the Memory object and Binary Quickload.

Deny reason (for on [..] denied)

S This expression returns the reason for denying an action when one of the On [..] denied conditions triggers.

Get welcome message (for on connect)

S This expression returns the welcome message (similar to MOTD) for the On connect condition.

Connection

Host IP address

S This expression returns the IP address of the server you are currently connected to. This will always be in the format "###.###.###.###".

Host port

N This expression returns the port that you are connected through to the server.

Channel listing

Get name

S This expression returns the name of the current channel in the channel list loop.

Get peer count

N This expression returns the number of peers in the channel being listed in the channel list loop. This number includes yourself if you are in that channel, thus there should never be a channel with 0 peers.

Received

Get text

S This expression returns the received text message.

Get number

N This expression returns the received number message.

Get binary size

N This expression returns the size in bytes of the received binary message.

Get binary memory address

N This expression returns the memory address of the received binary. This is often used in conjunction with the Memory object.

Get binary data

Byte

ASCII character

S This expression returns a string containing the single character represented by the indicated byte. It takes one parameter, which is the location of the byte in the binary message, with a 0-based index.

Integer value

Unsigned

N This expression returns the unsigned representation of the single byte at the specified location in the binary message. It takes one parameter, which is the location of the byte in the binary message, with a 0-based index. The value returned will be in the range 0 to 255.

Signed

N This expression returns the signed representation of the single byte at the specified location in the binary message. It takes one parameter, which is the location of the byte in the binary message, with a 0-based index. The value returned will be in the range -128 to 127.

Short

Unsigned

N This expression returns the unsigned representation of the two-byte short at the specified location in the binary message. It takes one parameter, which is the location of the short in the binary message, with a 0-based index. The value returned will be in the range 0 to 65535.

Signed

N This expression returns the signed representation of the two-byte short at the specified location in the binary message. It takes one parameter, which is the location of the short in the binary message, with a 0-based index. The value returned will be in the range -32768 to 32767.

Integer

Unsigned

N This expression returns the unsigned representation of the four-byte integer at the specified location in the binary message. It takes one parameter, which is the location of the integer in the binary message, with a 0-based index. The value returned would be in the range 0 to 4294967295, but MMF2 will truncate this value to be within the range -2147483648 to 2147483647, making this expression the same as getting the Signed integer.

Signed

N This expression returns the signed representation of the four-byte integer at the specified location in the binary message. It takes one parameter, which is the location of the integer in the binary message, with a 0-based index. The value returned will be in the range -2147483648 to 2147483647.

Float

N This expression returns the floating-point representation of the four-byte float at the specified location in the binary message. It takes one parameter, which is the location of the float in the binary message, with a 0-based index. The value returned will be in the range -3.4e38 to 3.4e38.

Text

With size

S This expression returns the string at the specified location in the binary message. It takes two parameters, the first being the location of the string in the binary message, with a 0-based index, and the second being the size in bytes/characters of the string to read.

Null terminated

S This expression returns the string at the specified location in the binary message. It takes one parameter, which is the location of the string in the binary message, with a 0-based index. The string will be read until a null terminator character is found in the binary message (byte value 0).

Get binary data (with cursor)

Byte

ASCII character

S This expression returns a string containing the single character represented by the byte at the cursor.

Integer value

Unsigned

N This expression returns the unsigned representation of the single byte at the cursor. The value returned will be in the range 0 to 255.

Signed

N This expression returns the signed representation of the single byte at the cursor. The value returned will be in the range -128 to 127.

Short

Unsigned

N This expression returns the unsigned representation of the two-byte short at the cursor. The value returned will be in the range 0 to 65535.

Signed

N This expression returns the signed representation of the two-byte short at the cursor. The value returned will be in the range -32768 to 32767.

Integer

Unsigned

N This expression returns the unsigned representation of the four-byte integer at the cursor. The value returned would be in the range 0 to 4294967295, but MMF2 will truncate this value to be within the range -2147483648 to 2147483647, making this expression the same as getting the Signed integer.

Signed

N This expression returns the signed representation of the four-byte integer at the cursor. The value returned will be in the range -2147483648 to 2147483647.

Float

N This expression returns the floating-point representation of the four-byte float at the cursor. The value returned will be in the range -3.4e38 to 3.4e38.

Text

With size

S This expression returns the string at the cursor. It takes one parameter, which is the size in bytes/characters of the string to read.

Null terminated

S This expression returns the string at the cursor. The string will be read until a null terminator character is found in

the binary message (byte value 0).

Get subchannel

N This expression returns the subchannel that the message was received on.

Self

Name

S This expression returns your name, and should always be used rather than keeping a separate record.

ID

N This expression returns your unique ID on the server.

Number of channels

N This expression returns the number of channels you are currently in.

Previous name (for on name changed)

S This expression returns your previous name for the On name changed condition.

Selected channel

Name

S This expression returns the name of the currently selected channel.

Number of peers

N This expression returns the number peers (excluding yourself) that are in the currently selected channel.

Selected peer

Name

S This expression returns the name of the currently selected peer in the currently selected channel.

ID

N This expression returns the unique ID of the currently selected peer in the currently selected channel on the server.

Previous name (for on peer name changed)

S This expression returns the previous name of a peer for the On peer changed name condition.

Properties

Version

This property shows you the version of the liblacewing library that this Lacewing Relay Client extension was

compiled with. Refer to the build number when asked which version of Lacewing you use. For example, this help file was made for Lacewing Build #20. This can be retrieved at runtime with the Lacewing version string expression.

Automatically clear binary

This property allows you to control whether the binary to send is automatically cleared after sending/blasting a binary message. It cannot be changed at runtime.

Global

This property is used on conjunction with the Global identifier property. It determines whether Lacewing will stay active across frames and even sub applications. This property cannot be changed at runtime. Note: if this is checked and you have a connection and go to a frame where there is not a matching global Lacewing Relay Client, the connection will remain active. Returning to a frame with a matching LRC will allow you to once again control the connection.

Global identifier

This property is used in conjunction with the Global property. This is a unique global identifier to separate/conjoin separate global Lacewing Relay Client extensions from/with each other within the same application. it can be anything at all, even a sentence with spaces, or it can be blank. As long as it is the same as another LRC extension somewhere else in the application, they will be considered the same object.

Lacewing Relay Server

The **Lacewing Relay Server** is an object which allows you to host and somewhat customize the behavior of a Lacewing Relay Server. Relay Clients connect to Relay Servers, and with this object you may control server-side behavior. This object should be used only if public servers do not provide the functionality needed (such as being for specific games, having custom behavior (like simulating games to prevent cheating), or for responding to moderation commands for chat programs). The Relay Server sees peers as its Clients.

Actions

Relay Server

Host

This action attempts to begin hosting the Relay Server. It takes one parameter, which is the port to host on. This may trigger an error, for instance, when there is already a server (of any kind) hosting on the specified port, be it TCP, UDP, or both.

Stop hosting

This action stops hosting if the server is hosting at all. It generates an error if the server is not hosting.

Flash Player Policy Server

Host

This action attempts to begin hosting the Relay Server. It takes one parameter, which is the Flash Policy XML file to host - it will always host on port 843, whether you want it to or not. This may trigger an error, for instance, when there is already a server (of any kind) hosting on port 843, or if the file can't be used.

Stop hosting

This stops hosting a policy file, generating an error if no policy file was being hosted in the first place.

Set welcome message

This action allows you to set the welcome message that clients receive when they connect to the server. The only parameter is the welcome message string. The string can be blank, have new lines, etc.

Enable conditions

On message to channel

This action allows the "On message to channel" condition to be triggered so that you may have more fine grained control over the behavior of the server. Normally, for efficiency purposes, this condition is disabled from being triggered. This action takes no parameters.

On message to peer

This action allows the "On message to peer" condition to be triggered so that you may have more fine grained control over the behavior of the server. Normally, for efficiency purposes, this condition is disabled from being triggered. This action takes no parameters.

On interactive condition

Deny (for on [..] request)

This action allows you to give the generic 'request denied'. It does not allow you to set the deny reason message.

Change name (for name set/change request)

This action allows you to change the name of a client from their requested name. This allows you to give them an entirely different name from what they requested, though it is not recommended to do so. Its only parameter is the new name.

Change channel name (for channel join request)

This action allows you to change the name of a channel from the requested name as it is being created. This allows you to give the channel an entirely different name from what was requested, though it is not recommended to do so. Its only parameter is the new name.

Drop message (for on message to channel/peer)

This action is the 'silent' version of the Deny action - it simply refuses to send any message at all, but leaves no trace (eg there is no error message or notice sent to the client that send the message or the client(s) that would have received the message).

Channel

Close channel

Closes the currently selected channel. All peers within are forced to leave the channel. Any local channel data stored with the channel is lost.

Select the channel master

Sets the currently selected peer as the channel master of the currently selected channel. This can be used to create behavior where if the channel master leaves a new one is picked. There is no way to make a channel not have a channel master if it has one, however, without completely disconnecting the channel master from the server.

Loop clients

Starts a client loop of all the clients in the currently selected channel. There is no version of this action that uses a loop name, despite there being conditions for such an action.

Select by name

Selects a channel by the given name parameter.

Loop all channels

Starts a channel loop of all channels on the server.

Loop all channels (with loop name)

Starts a channel loop of all channels on the server with the given loop name.

Set local channel data

This action allows you to set server-side information about a channel. For the current channel, it associates a string value with a key. The first parameter is the key to use, and the second is the new string value to store with that key. This is very useful for storing information such as game session information, channel state information, etc., but all the data is lost when the channel closes.

Client

Disconnect

This action forcefully disconnects a client from the server, causing them to leave all channels they are in. Any local client data stored with the client is lost.

Loop client channels

This action starts a loop of all the channels the client is currently in.

Loop client channels (with loop name)

This action starts a loop of all the channels the client is currently in using the given loop name.

Select by name

Selects a client on the server by the given name parameter. Because there can be more than one client with the same name on a server, this action should be avoided when possible.

Select by ID

Selects a client by the given ID parameter. All clients are guaranteed to have a unique ID.

Select sender (for "on message to peer")

This action allows you to modify the 'from' peer of a peer message (not a channel message) by setting the currently selected peer as the peer who sent the message.

Select receiver (for "on message to peer")

This action allows you to modify the 'to' peer of a peer message (not a channel message) by setting the currently selected peer as the peer to receive the message.

Loop all clients

This action starts a loop of all the clients on the server.

Loop all clients (with loop name)

This action starts a loop of all the clients on the server with the given loop name.

Set local client data

This action allows you to set server-side information about a client. For the current client, it associates a string value with a key. The first parameter is the key to use, and the second is the new string value to store with that key. This is very useful for storing information such as game session information, client state information, etc., but all the data is lost when the client disconnects.

Send

Text

To client

This action sends text to the currently selected client. It takes two parameters: the subchannel to send on, and the text to send.

To channel

This action sends text to the currently selected channel. It takes two parameters: the subchannel to send on, and the text to send.

Number

To client

This action sends a number to the currently selected client. It takes two parameters: the subchannel to send on, and the number to send.

To channel

This action sends a number to the currently selected channel. It takes two parameters: the subchannel to send on, and the number to send.

Stack

Fun fact: this submenu should actually be named "Binary". "Stack" is the previous name used for this feature.

To client

This action sends a binary message to the currently selected client. It takes one parameter: the subchannel to send on.

To channel

This action sends a binary message to the currently selected channel. It takes one parameter: the subchannel to send on.

Blast

Text

To client

This action blasts text to the currently selected client. It takes two parameters: the subchannel to blast on, and the text to blast.

To channel

This action blasts text to the currently selected channel. It takes two parameters: the subchannel to blast on, and the text to blast.

Number

To client

This action blasts a number to the currently selected client. It takes two parameters: the subchannel to blast on, and the number to blast.

To channel

This action blasts a number to the currently selected channel. It takes two parameters: the subchannel to blast on, and the number to blast.

Stack

Fun fact: this submenu should actually be named "Binary". "Stack" is the previous name used for this feature.

To client

This action blasts a binary message to the currently selected client. It takes one parameter: the subchannel to blast on.

To channel

This action blasts a binary message to the currently selected channel. It takes one parameter: the subchannel to blast on.

Binary to send

Add byte

ASCII character

This action adds one byte to the end of the current binary message. It takes one sting parameter, which is the character you want to add. You should only put one character in the string, eg "c". If you want to add multiple characters at once, use one of the Add string actions instead.

Integer value

This action adds one byte to the end of the current binary message. It takes one parameter: the integral value of the byte you want to add. It does not matter if it is signed or unsigned, as the binary representations are the same. Remember that the range of a byte is -128 to 127 or 0 to 255, depending on how it is interpreted by the receiver.

Add short

This action adds two bytes to the end of the current binary message. It takes one parameter: the integral value of the short you want to add. It does not matter if it is signed or unsigned, as the binary representations are the same. Remember that the range of a short is -32768 to 32767 or 0 to 65535, depending on how it is interpreted by the receiver.

Add integer

This action adds four bytes to the end of the current binary message. It takes one parameter: the integral value of the integer you want to add. It does not matter if it is signed or unsigned, as the binary representations are the same - additionally, MMF2 can only handle signed integers at most. Remember that the range of an integer is -2147483648 to 2147483647.

Add float

This action adds four bytes to the end of the current binary message. It takes one parameter: the floating point value of the float you want to add. MMF2 can handle double precision floats, which are eight bytes, but can only give extensions four bytes at most, which is why Lacewing only lets you send floats and not doubles. Remember that the range of a float is 3.4e-38 to 3.4e38.

Add string

Without null terminator

This action allows you to simply add a plain string to the end of the current binary data. The number of bytes added is literally the length of the string in characters. It takes one parameter: the string to add.

With null terminator

This action allows you to add a null-terminated string to the end of the current binary data. The number of bytes added is the length of the string in characters plus one for the null character (byte value 0). It takes one parameter: the string to add.

Add binary

This action is for advanced usage; it takes two parameters: the location of binary in memory, and how much of it you want to add. The number of bytes added is literally the second parameter. This is useful in conjunction with objects such as the Binary object and/or the Memory object.

Add file

This action allows you to add an entire file to the end of the current binary data. It takes one parameter, which is the file to add. You can choose to use an expression to evaluate the file path. Do not send large files, the most a binary message can hold is 65535 bytes. If you intend to send large files, you should break the file up into smaller chunks and send them individually.

Resize

This action will resize the binary to send to whatever number of bytes you specify via the only parameter. This is useful in conjunction with the Binary to send address expression, as you can load binary into the existing memory though another object (such as Binary Quickload).

Compress (ZLIB)

This action compresses the binary using ZLIB compression routines. You should find a balance between faster send/blast time with smaller binary messages and faster message generation/interpreting time when not compressing/decompressing the binary message.

Clear

This action simply clears the binary to send and sets its size to 0. If you have Automatically clear binary checked, then this action will occur whenever you send/blast a binary message. Otherwise, it is up to you to decide when to clear a binary message.

Received binary

Save to a file

This action allows you to save a portion of the received binary to a file, overwriting the file if it already exists. It takes three parameters, the first being the position in the received binary to be the start of the file, the second being how many bytes after that you want to save to the file, and the third being a string expression for the path to the file to save to.

Append to a file

This action allows you to save a portion of the received binary to the end of a file, creating the file if it does not exist. It takes three parameters, the first being the position in the received binary to be the start of the file, the second being how many bytes after that you want to save to the file, and the third being a string expression for the path to the file to save to.

Uncompress (ZLIB)

This action simply decompresses the received binary using ZLIB compression routines.

Move cursor

This action moves the cursor in the received binary for cursor related expressions. It takes one parameter: the new position for the cursor.

Conditions

On Error

I This condition is triggered whenever an error occurs.

Lacewing server is hosting

N This condition is true if the server is hosting at all.

Flash Player policy server is hosting

N This condition is true if the flash policy server is hosting at all.

Connection

On connect request

I This condition is triggered when a client requests connection to the server. The client will be selected for you.

On disconnect

I This condition is triggered when a client disconnects from the server. The client will be selected for you.

Channel

On join request

I This condition is triggered when a client requests to join a channel. The client will be selected for you, but the channel will not as it has yet to be created.

On leave request

I This condition is triggered when a client requests to leave a channel they are in. Both the client and channel will be selected for you. It would be evil to deny them from leaving the channel, though it is an action you can do during this event.

On all channels loop

I This condition is triggered for each channel in a channel loop of all channels on the server. The channel will be selected for you.

On all channels loop finished

I This condition will be triggered after all channels have been looped through. It currently serves no purpose, though it may in the future.

On client channels loop

I This condition is triggered for each client in a channel. The client and channel will be selected for you.

On client channels loop finished

I This condition is triggered when the loop of all the clients in a channel is finished.

[With loop name] On all channels loop

I This condition is triggered for each channel in a channel loop of all channels on the server. The channel will be selected for you. It takes one parameter, which is the name of the loop.

[With loop name] On all channels loop finished

I This condition will be triggered after all channels have been looped through. It currently serves no purpose, though it may in the future. It takes one parameter, which is the name of the loop.

[With loop name] On client channels loop

I This condition is never triggered because there is no corresponding action to trigger it.

[With loop name] On client channels loop finished

I This condition is never triggered because there is no corresponding action to trigger it.

Channel is hidden from the channel list

N This condition is true if the currently selected channel is marked as hidden from the channel list. This property can only be set by the client that creates the channel.

Channel is set to close automatically

N This condition is true if the currently selected channel is marked to close when the channel master leaves. This property can only be set by the client that creates the channel.

Client

Client is the channel master

N This condition is true if the currently selected client is the channel master of the currently selected channel.

On name set request

I This condition is triggered when a client requests to set or change their name.

On all clients loop

I This condition is triggered for each client on the server. The client is selected for you, but the selected channel is not affected as clients can be in more than one channel.

On all clients loop finished

I This condition is triggered when the loop of all clients has completed.

On channel clients loop

I Despite the name being an obvious copy and paste from the Channel submenu, this condition is triggered for each channel a client is in. Both the client and channel are selected for you.

On channel clients loop finished

I Despite the name being an obvious copy and paste from the Channel submenu, this condition is triggered when the loop of all channels a client is in has completed.

[With loop name] On all clients loop

I This condition is triggered for each client on the server. The client is selected for you, but the selected channel is not affected as clients can be in more than one channel. Its only parameter is the name of the loop.

[With loop name] On all clients loop finished

I This condition is triggered when the loop of all clients has completed. Its only parameter is the name of the loop.

[With loop name] On channel clients loop

I Despite the name being an obvious copy and paste from the Channel submenu, this condition is triggered for each channel a client is in. Both the client and channel are selected for you. Its only parameter is the name of the loop.

[With loop name] On channel clients loop finished

I Despite the name being an obvious copy and paste from the Channel submenu, this condition is triggered when the loop of all channels a client is in has completed. Its only parameter is the name of the loop.

Message

Sent

On text message to server

This condition is triggered when a client has sent a text message to the server. The client is selected for you. It takes one parameter, which is the subchannel to filter this condition for.

On number message to server

This condition is triggered when a client has sent a number message to the server. The client is selected for you. It takes one parameter, which is the subchannel to filter this condition for.

On binary message to server

This condition is triggered when a client has sent a binary message to the server. The client is selected for you. It takes one parameter, which is the subchannel to filter this condition for.

On any message to server

This condition is triggered when a client has sent a message to the server. The client is selected for you. It takes one parameter, which is the subchannel to filter this condition for.

On text message to channel

This condition is triggered when a client has sent a text message to a channel. The client and channel are selected for you. It takes one parameter, which is the subchannel to filter this condition for.

On number message to channel

This condition is triggered when a client has sent a number message to a channel. The client and channel are selected for you. It takes one parameter, which is the subchannel to filter this condition for.

On binary message to channel

This condition is triggered when a client has sent a binary message to a channel. The client and channel are selected for you. It takes one parameter, which is the subchannel to filter this condition for.

On any message to channel

This condition is triggered when a client has sent a message to a channel. The client and channel are selected for you. It takes one parameter, which is the subchannel to filter this condition for.

On text message to peer

This condition is triggered when one client has sent a text message to another client. The channel is selected for you, but there is an obvious conceptual error as to which client is selected. It takes one parameter, which is the subchannel to filter this condition for.

On number message to peer

This condition is triggered when one client has sent a number message to another client. The channel is selected for you, but there is an obvious conceptual error as to which client is selected. It takes one parameter, which is the subchannel to filter this condition for.

On binary message to peer

This condition is triggered when one client has sent a binary message to another client. The channel is selected for you, but there is an obvious conceptual error as to which client is selected. It takes one parameter, which is the subchannel to filter this condition for.

On any message to peer

This condition is triggered when one client has sent a message to another client. The channel is selected for you, but there is an obvious conceptual error as to which client is selected. It takes one parameter, which is the subchannel to filter this condition for.

Blasted

On text message to server

This condition is triggered when a client has blasted a text message to the server. The client is selected for you. It takes one parameter, which is the subchannel to filter this condition for.

On number message to server

This condition is triggered when a client has blasted a number message to the server. The client is selected for you. It takes one parameter, which is the subchannel to filter this condition for.

On binary message to server

This condition is triggered when a client has blasted a binary message to the server. The client is selected for you. It takes one parameter, which is the subchannel to filter this condition for.

On any message to server

This condition is triggered when a client has blasted a message to the server. The client is selected for you. It takes one parameter, which is the subchannel to filter this condition for.

On text message to channel

This condition is triggered when a client has blasted a text message to a channel. The client and channel are selected for you. It takes one parameter, which is the subchannel to filter this condition for.

On number message to channel

This condition is triggered when a client has blasted a number message to a channel. The client and channel are selected for you. It takes one parameter, which is the subchannel to filter this condition for.

On binary message to channel

This condition is triggered when a client has blasted a binary message to a channel. The client and channel are selected for you. It takes one parameter, which is the subchannel to filter this condition for.

On any message to channel

This condition is triggered when a client has blasted a message to a channel. The client and channel are selected for you. It takes one parameter, which is the subchannel to filter this condition for.

On text message to peer

This condition is triggered when one client has blasted a text message to another client. The channel is selected for you, but there is an obvious conceptual error as to which client is selected. It takes one parameter, which is the subchannel to filter this condition for.

On number message to peer

This condition is triggered when one client has blasted a number message to another client. The channel is selected for you, but there is an obvious conceptual error as to which client is selected. It takes one parameter, which is the subchannel to filter this condition for.

On binary message to peer

I This condition is triggered when one client has blasted a binary message to another client. The channel is selected for you, but there is an obvious conceptual error as to which client is selected. It takes one parameter, which is the subchannel to filter this condition for.

On any message to peer

I This condition is triggered when one client has blasted a message to another client. The channel is selected for you, but there is an obvious conceptual error as to which client is selected. It takes one parameter, which is the subchannel to filter this condition for.

Expressions

Error string (for on error)

S This expression returns the error string from the most recently triggered error condition.

Lacewing version string

S This expression returns the version string for the version of Lacewing that this server extension wraps.

Port

N This expression returns the port that the server is hosting on. This is not for the flash policy server, which always hosts on port 843.

Binary to send size

N This expression returns the size in bytes of the binary to send.

Binary to send address

N This expression returns the address of the binary to send. This can be used in conjunction with extensions like Binary Quickload and the Memory Object for quickly filling the binary to send with data.

Requested name (for name set/change request)

S This expression returns the requested name for a client (as the client's name has not been set at the point of triggering the On name set request condition).

Requested channel name (for channel join request)

S This expression returns the requested name of the channel that a client wants to join. Even if the channel exists, the channel is not selected because the client is not actually in the channel at the point of triggering the On join request condition.

Channel

Name

S This expression returns the name of the currently selected channel.

Client count

N This expression returns the number of clients in the currently selected channel.

Get local channel data

S This expression retrieves the value string associated with the given key string for the currently selected channel.

Number of channels on the server

N This expression returns the number of channels currently on the server.

Client

Name

S This expression returns the name of the currently selected client.

ID

N This expression returns the ID of the currently selected client.

IP address

S This expression returns the IP address of the currently selected client as a string.

Connection time

? This expression has been removed from Lacewing, but the menu name has not. Bug Jamie about it.

Channel count

N This expression returns the number of channels the currently selected client is in.

Get local client data

S This expression retrieves the value string associated with the given key string for the currently selected channel.

Get client protocol implementation

? This expression has been removed from Lacewing, but the menu name has not. Bug Jamie about it.

Received

Get text

S This expression returns the received message as text.

Get number

N This expression returns the received message as a number.

Get binary size

N This expression returns the size of the received binary data.

Get binary memory address

N This expression returns the memory address of the received binary data.

Get binary data

Byte

ASCII character

S This expression returns a string containing the single character represented by the indicated byte. It takes one parameter, which is the location of the byte in the binary message, with a 0-based index.

Integer value

Unsigned

N This expression returns the unsigned representation of the single byte at the specified location in the binary message. It takes one parameter, which is the location of the byte in the binary message, with a 0-based index. The value returned will be in the range 0 to 255.

Signed

N This expression returns the signed representation of the single byte at the specified location in the binary message. It takes one parameter, which is the location of the byte in the binary message, with a 0-based index. The value returned will be in the range -128 to 127.

Short

Unsigned

N This expression returns the unsigned representation of the two-byte short at the specified location in the binary message. It takes one parameter, which is the location of the short in the binary message, with a 0-based index. The value returned will be in the range 0 to 65535.

Signed

N This expression returns the signed representation of the two-byte short at the specified location in the binary message. It takes one parameter, which is the location of the short in the binary message, with a 0-based index. The value returned will be in the range -32768 to 32767.

Integer

Unsigned

N This expression returns the unsigned representation of the four-byte integer at the specified location in the binary message. It takes one parameter, which is the location of the integer in the binary message, with a 0-based index. The value returned would be in the range 0 to 4294967295, but MMF2 will truncate this value to be within the range -2147483648 to 2147483647, making this expression the same as getting the Signed integer.

Signed

N This expression returns the signed representation of the four-byte integer at the specified location in the binary message. It takes one parameter, which is the location of the integer in the binary message, with a 0-based index. The value returned will be in the range -2147483648 to 2147483647.

Float

N This expression returns the floating-point representation of the four-byte float at the specified location in the binary

message. It takes one parameter, which is the location of the float in the binary message, with a 0-based index. The value returned will be in the range -3.4e38 to 3.4e38.

Text

With size

S This expression returns the string at the specified location in the binary message. It takes two parameters, the first being the location of the string in the binary message, with a 0-based index, and the second being the size in bytes/characters of the string to read.

Null terminated

S This expression returns the string at the specified location in the binary message. It takes one parameter, which is the location of the string in the binary message, with a 0-based index. The string will be read until a null terminator character is found in the binary message (byte value 0).

Get binary data (with cursor)

Byte

ASCII character

S This expression returns a string containing the single character represented by the byte at the cursor.

Integer value

Unsigned

N This expression returns the unsigned representation of the single byte at the cursor. The value returned will be in the range 0 to 255.

Signed

N This expression returns the signed representation of the single byte at the cursor. The value returned will be in the range -128 to 127.

Short

Unsigned

N This expression returns the unsigned representation of the two-byte short at the cursor. The value returned will be in the range 0 to 65535.

Signed

N This expression returns the signed representation of the two-byte short at the cursor. The value returned will be in the range -32768 to 32767.

Integer

Unsigned

N This expression returns the unsigned representation of the four-byte integer at the cursor. The value returned would be in the range 0 to 4294967295, but MMF2 will truncate this value to be within the range -2147483648 to 2147483647, making this expression the same as getting the Signed integer.

Signed

N This expression returns the signed representation of the four-byte integer at the cursor. The value returned will be in the range -2147483648 to 2147483647.

Float

N This expression returns the floating-point representation of the four-byte float at the cursor. The value returned will be in the range -3.4e38 to 3.4e38.

Text

With size

S This expression returns the string at the cursor. It takes one parameter, which is the size in bytes/characters of the string to read.

Null terminated

S This expression returns the string at the cursor. The string will be read until a null terminator character is found in the binary message (byte value 0).

Get subchannel

N This expression returns the subchannel that the message was received on.

Properties

Version

This property displays the version string of the Lacewing library that this server extension wraps.

Automatically clear binary

If this property is ticked, the binary to send will be cleared after sending/blasting the binary message.

Global

If ticked, this server object will be global across frames and sub-apps to other server objects that have the same global identifier.

Global identifier

This property is the global identifier to be used to determine whether two server objects share the same state or not. It can be blank.

Lacewing Webserver

The **Lacewing Webserver** is an object which allows you to host a completely customizable Lacewing Webserver, hosing either an HTTP server, and HTTPS server, or both, and handling GET, POST, and HEAD requests from HTTP clients (such as web browsers). You must generate response messages yourself in the same event as the On [..] request conditions, though it may be broken up into several On [..] request events. The header is generated for you, with you being able to change some things (such as MIME type and/or Location: header items). You as the programmer must design it to behave in any way you see fit, be it actually for hosting a website or for acting as a miniature remote control panel.

Actions

HTTP server

Host

Attempts to begin hosting a normal HTTP server on the given port.

Stop hosting

Attempts to stop hosting the HTTP server.

HTTPS server

Host

Attempts to host an HTTPS server on the given port.

Stop hosting

Attempts to stop hosting the HTTPS server.

Load certificate from system

Loads a certificate for the HTTPS server from the given store, common name, and location.

On request

Set MIME type

Sets the MIME type in the header from the given string. This is useful if you want to force a different MIME type than Lacewing would guess.

Set MIME type (with charset)

Sets the MIME type from the given string and also the charset from the given string in the header.

Set cookie

Sets a cookie by name to the specified string value. Useful for storing long-term session data, but not as tasty.

Caching

Disable response caching

Prevents web browsers from caching the response for faster page loading. This is useful if you want the page to be downloaded from the server every time.

Set as unmodified

Tells web browsers that the page or file being requested has not changed from before.

Set last modified

Sets the Last Modified time from the given Unix time value.

Add to response

String

Adds the given string parameter to the end of the response.

File

Guess MIME type

Adds the given file to the end of the response, changing the MIME type to Lacewing's best guess at what the file is.

Leave the MIME type alone

Adds the given file to the end of the response while not affecting the MIME type in any way.

Binary

Set big endian

Sets the binary mode to Big Endian.

Set little endian (default)

Sets the binary mode to Little Endian.

Byte

Adds the given one byte byte to the end of the response.

Short

Adds the given two byte short to the end of the response.

Integer

Adds the given four byte integer to the end of the response.

From memory address

Adds data from the given memory address for the given number of bytes.

Redirect to another URL

Adjusts the header so that the browser will go to the given URL after the request is finished. You can still add information to the request, but most browsers will not show any of it as they will have already gone to the new URL.

Conditions

On error

I This condition is triggered upon an error occurring, such as not being able to host the server because there is already another server hosting on the same port.

HTTP server is hosting

N This condition is true if the HTTP server is hosting at all.

HTTPS server is hosting

N This condition is true if the HTTPS server is hosting at all.

HTTPS certificate is loaded

N This condition is true if an HTTPS certificate is loaded.

Connection

On disconnect

I This condition is triggered when a client disconnects from the web server.

Connection is HTTPS

N This condition is true if the request currently being processed is through the HTTPS server rather than the regular HTTP server.

Request

On GET request

I This condition is triggered when a connected client sends a GET request.

On POST request

I This condition is triggered when a connected client sends a POST request.

On HEAD request

I This condition is triggered when a connected client sends a HEAD request (where they only want the header).

Expressions

Error string (for on error)

S This expression returns the error string from the last triggered error.

Lacewing version string

S This expression returns the version string of the Lacewing library this Webserver object wraps.

Get connection IP address

S This expression returns the IP address of the client whose request is currently being processed.

Get local connection data

S This expression returns the data associated with the given key for the current whose request is currently being processed. It is useless, however, because there is no action to set the local connection data in the first place.

Statistics

Bytes sent

N This expression returns the number of bytes that have been sent throughout the lifetime of the server.

Bytes received

N This expression returns the number of bytes that have been received throughout the lifetime of the server.

Request

Get URL

S This expression returns the requested URL, without the hostname or protocol.

Get header

S This expression returns the header item with the given name.

Get cookie

S This expression returns the string value of a cookie with the given name.

Get last modified time

N This expression returns the last modified time in Unix units for the given name.

Get GET item

S This expression returns the GET item with the given name.

Get POST item

S This expression returns the POST item with the given name.

Get hostname

S This expression returns the hostname for which the client is requesting data. This is useful if the same server is used to host multiple sites with different hostnames.

Guess MIME type for filename

S This expression *would* return the MIME type for the given filename, but it can't actually be given a filename due to a programming error. It accepts a nameless number parameter rather than the expected filename string parameter, making it useless.

Properties

Version

This property displays the version string of the Lacewing library that this Webserver object wraps.

Global

If ticked, this allows Webserver objects with the same global identifier to share state across frames and sub applications.

Global identifier

This is the global identifier used to determine which Webserver objects share state with each other. It can be blank.