Scratch Connector

Manual

Author: Stamati Crook
Email: stamati.crook@redware.com
Date: 29 July 2009
Doc: scratchconnector-manual-0.11.doc

redware research ltd
Brighton Media Centre. 15-17 Middle Street. Brighton BN1 1AL. England.
web www.redware.com

1. Contents

2. Redware Research Limited

Redware Research Limited is a small software company specialising in software applications for business located on the south coast of England.

Our **Scratch Learning Program** aims to help teachers introduce Scratch into the classroom as part of the teaching curriculum. Please contact us if you would like more information on how to introduce Scratch into your school.

Stamati Crook
Scratch Learning Program
Redware Research Limited
Brighton Media Centre (306)
15-17 Middle Street
Brighton BN1 1AL
England

Telephone: +44 (0) 203 179 9444
Web: www.redware.com
Email: stamati.crook@redware.com

3. Introduction

This is the first release of the Scratch Connector for connecting two copies of Scratch together over the internet. It is pretty basic and not properly tested.

We hope to have a server version ready in the next few weeks. The server version will allow more than two copies of Scratch to be connected and raises many issues discussed in the “conventions” section of this document.

Thank you to **chalkmarrow** and **magnie** for your comments and help so far on the Scratch Connections wiki.

The software is free for non-commercial use and is part of the Scratch Learning Program from Redware Research Limited. Please look at http://www.redware.com/scratch for some tutorials and videos on learning Scratch.

If you are looking at this software in the period before we release our server version, please email us at sync@redware.com and tell us how you got on.

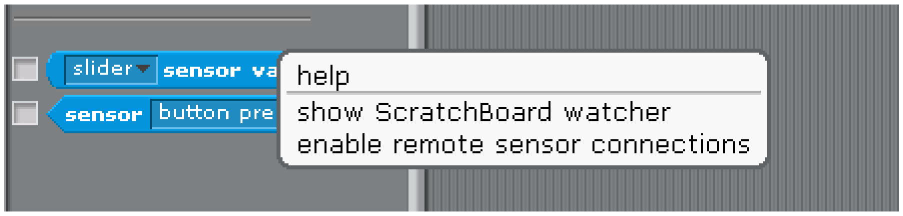
4. Scratch Connector

You can connect two or more Scratch applications together in the classroom or over the internet to make multi-player games or simply communicate between applications (this feature is available in Scratch version 1.3 onwards).

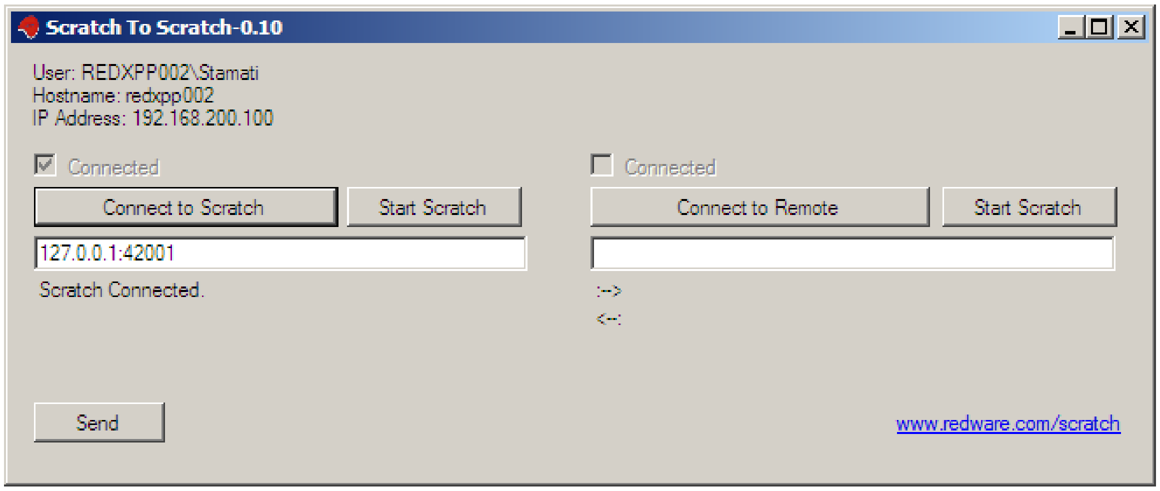
Be aware that Scratch communicates using a port that may be blocked on your firewall (42001) so connecting over the internet may be difficult. Connecting copies of Scratch in the same class on the same network should not usually be a problem.

A Scratch application communicates each time a message is **broadcast** or a change is made to a **global variable.** Any Scratch applications connected and listening will receive and react to the broadcast message. Global variables are translated into **remote sensor values** which can be used in your application to position a sprite in the right place for a multi-player game for example. See below for how to create the sensor variables when programming your application for the first time.

You need to **enable remote sensor connections** in your Scratch application the first time you want to connect by right clicking on a sensor in the sensor blocks area. The application will remember this setting subsequently (in version 1.4 onwards).



You can now connect to Scratch with the Scratch Connector program and send and receive messages. Bring up the application and press the **Connect to Scratch** button to connect to the copy of Scratch on your local machine.



User: REDXPP002\Stamati
Hostname: redxpp002
IP Address: 192.168.200.100

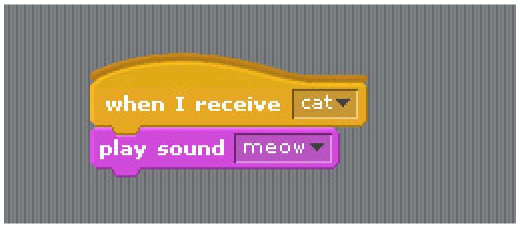
[✓] Connected
Connect to Scratch | Start Scratch
127.0.0.1:42001
Scratch Connected.

[ ] Connected
Connect to Remote | Start Scratch
:-->
<--:

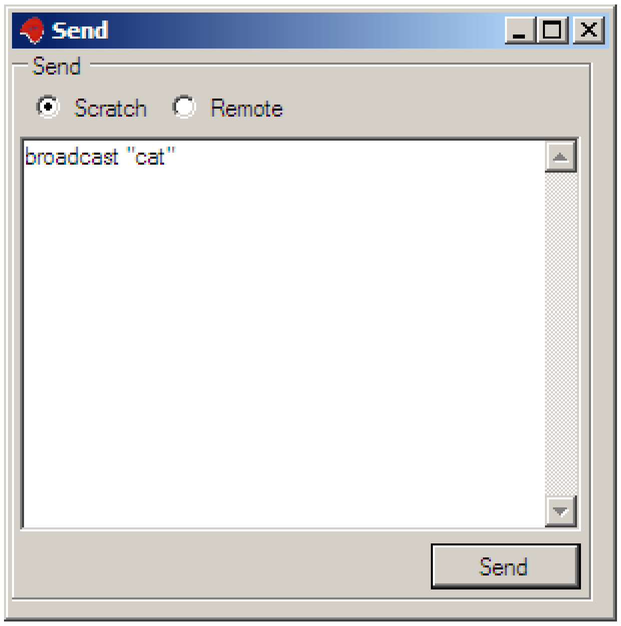
Send

www.redware.com/scratch

Create a simple broadcast event in your application so you can test. The example below makes a cat noise when the broadcast message 'cat' is received.

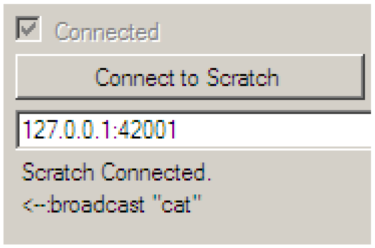


Press the **send** button on the Scratch Connector to bring up the Send form. Now type in **broadcast "cat"** in the send area of the application and press the **send** button. As you press the button you should hear the application making a meow sound if everything is working correctly.



broadcast "cat"

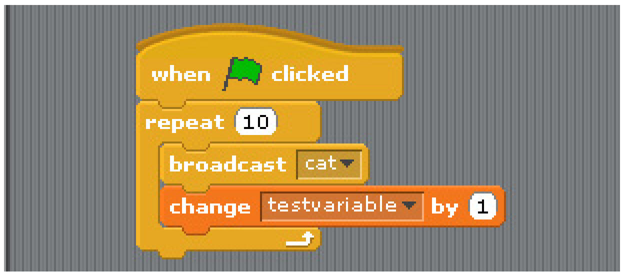
Notice how the message area for the on the main Scratch Connector page shows that the message has been sent to the application. The two message lines here show you the messages passing back and forth between the Scratch Application and the Connector.



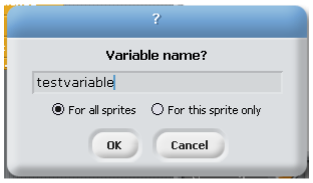
127.0.0.1:42001

```
Scratch Connected.
<--:broadcast "cat"
```

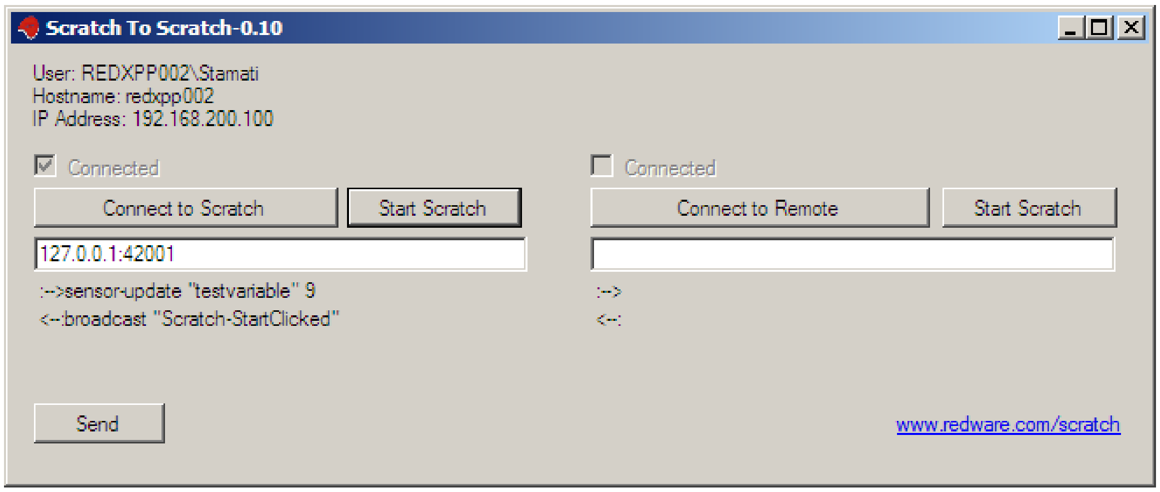
Now write a simple **repeat** loop to broadcast and update a global variable.


```
when ⚑ clicked
repeat 10
    broadcast cat
    change testvariable by 1
```

Remember that a **global variable** must have the **for all sprites** setting specified.



Press the **Start Scratch** button to start the Scratch application (or press the green start of your application) and notice how the messages are updated in the Connector as your Scratch application communicates:

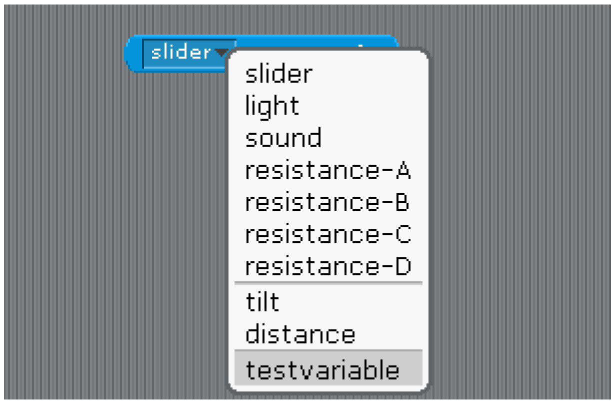


```
User: REDXPP002\Stamati
Hostname: redxpp002
IP Address: 192.168.200.100

127.0.0.1:42001
:-->sensor-update "testvariable" 9
<--:broadcast "Scratch-StartClicked"
```

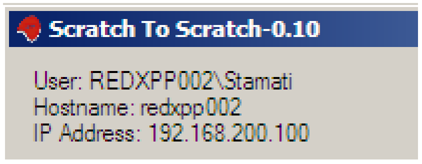
One problem that you will encounter when creating your application is that you cannot make a **remote sensor variable** to use in your programming blocks within Scratch. You have to send a **sensor-update** message to the application in order to create the sensor value initially. Send the following message to your application to create a remote sensor (also called **testvariable**) so you can use it in a programming block.

```
sensor-update "testvariable" 1
```



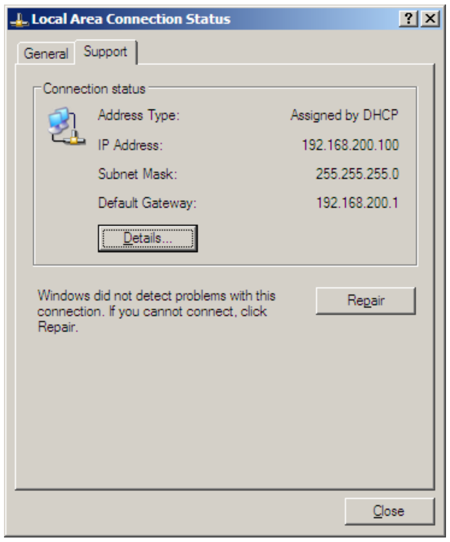
You now have a global variable and a remote sensor variable with the same name in your application. You can change the values in the global variable in your local application and these will update the remote sensor of the same name in the remote application. See the next section for some programming tips.

Now it is time to connect up a two player game. Load up the game on your machine and get a friend to do so on a second machine. Get your friend to tell you the IP address of his machine which they can find out by running the Scratch Connector and looking at the top left of the main form:


Scratch To Scratch-0.10

User: REDXPP002\Stamati
Hostname: redxpp002
IP Address: 192.168.200.100

Another way to do this is to go to the network connections section of your **control panel** and right click on the **local area connection** and bring up the **status** window. You should be able to see the IP address.


Local Area Connection Status

General | Support

Connection status

Address Type: Assigned by DHCP
IP Address: 192.168.200.100
Subnet Mask: 255.255.255.0
Default Gateway: 192.168.200.1

Details...

Windows did not detect problems with this connection. If you cannot connect, click Repair.

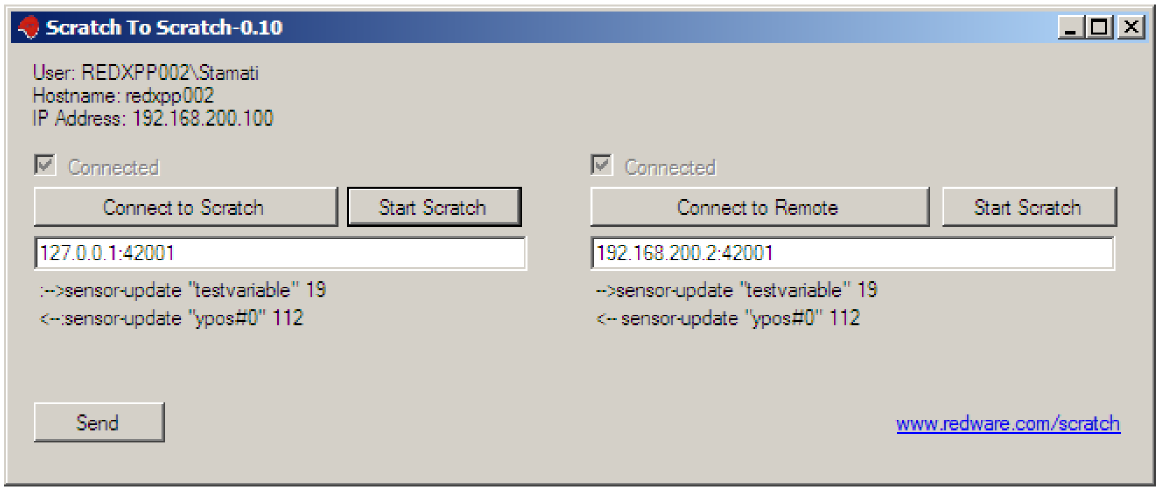
Repair

Close

Your friend will need to tell you his IP address and you can type it in on the right hand side of the **Scratch to Scratch** Connection form. Remember that you need to add a colon and 42001 for the port. The above example needs the following entry:
192.168.200.100:42001

Make sure that your two player Scratch application is running on both machines. Use our **2PLAYER.SB** example if you like which is designed so you can run the same application on both machines.

Now press **Connect to Scratch** to connect to the local machine and **Connect to Remote** to connect to the remote machine. If both connect OK, press the **Start Scratch** for both the applications to start the application on each machine. Notice the messages updating for each connection. You should each be able to move your player around with the mouse and make it move on your opponent's machine.



User: REDXPP002\Stamati
Hostname: redxpp002
IP Address: 192.168.200.100

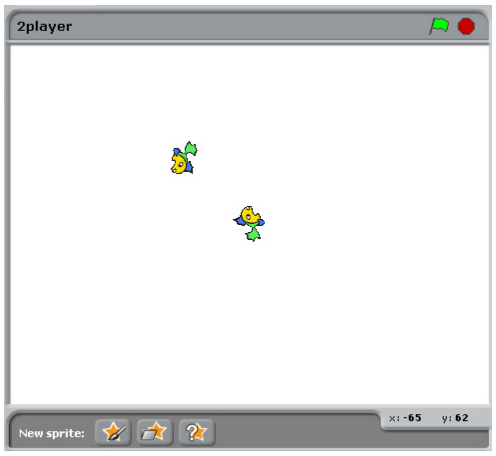
127.0.0.1:42001

```
:-->sensor-update "testvariable" 19
<--:sensor-update "ypos#0" 112
```

192.168.200.2:42001

```
-->sensor-update "testvariable" 19
<-- sensor-update "ypos#0" 112
```

This is the first test version of this application – improvements on their way. We have found that you get problems with the local Scratch application if there is nothing connected at the other end. If your application stalls, you may have to end the program with **Control-Alt-Delete** and terminate Scratch using the Windows Task Manager. Make sure you have someone who can help you if you need to do this and save any work before you run this program.



5. Scratch Protocol

Scratch always connects on port 42001 and communicates each time a **global variable** is changed or a **broadcast** takes place. You can send a message back into Scratch to do one of the following:

- Start the Application
- Send a broadcast message
- Update the value of a remote sensor

You cannot update the remote sensor values inside your own Scratch application but you can create and global variables with the same name as the remote sensor and these will be sent as messages and will update the remote sensor values in the remote copy of Scratch.

Start the Application

You can start (or restart) the Scratch application:

```
broadcast "Scratch-StartClicked"
```

As soon as you start Scratch, the value of each global variable is sent out as a **sensor-update** message.

Messages sent by Scratch

Messages are sent by Scratch each time a broadcast occurs or a global variable changes its value:

```
sensor-update "globalvariablename" 0.3830383
broadcast "broadcastname"
```

Messages received by Scratch

You can send similar messages back into Scratch:

```
broadcast "broadcastname"
sensor-update "numbersensorname" 0.1
sensor-udpate "textsensorname" "textvalue"
```

The sensor-update message creates the remote sensor within your Scratch project if it does not already exist. You need to send these messages manually into your project when you are writing the application for the first time to be able to use the sensor values in your Scratch programming blocks.

It seems you can omit quotation marks:

```
broadcast broadcastname
sensor-update numbersensorname 0.1
sensor-udpate textsensorname textvalue
```

You can combine multiple sensor updates:

```
sensor-update "var01" 0.1 "var02" "textvalue"
```

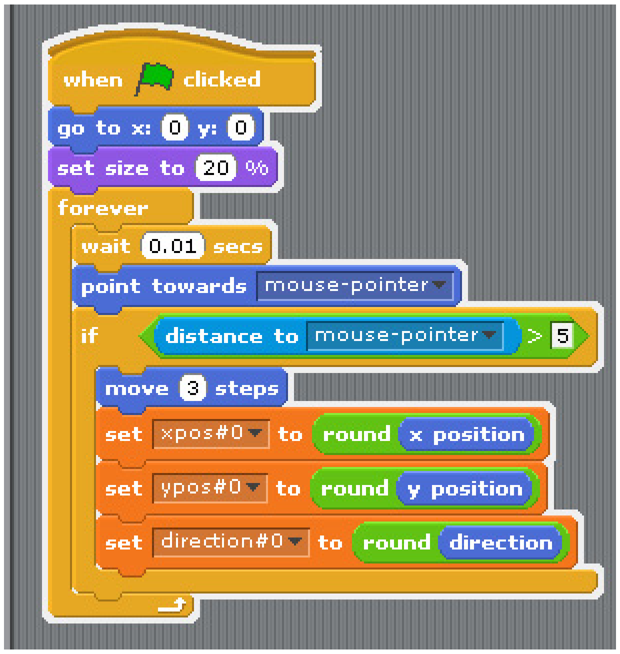
6. Programming Scratch

Scratch communicates to remote copies of Scratch each time a broadcast occurs or the value is changed on a global variable. You can connect many Scratch client programs up together and you may want to follow some kind of convention so that you can run the same program on each machine.

2 Player Game

A two player game will contain two sprites with one sprite controlled by each player. We will call the main player **player#0** and it will update three global variables periodically with its position and direction:

- xpos#0
- ypos#0
- direction#0.

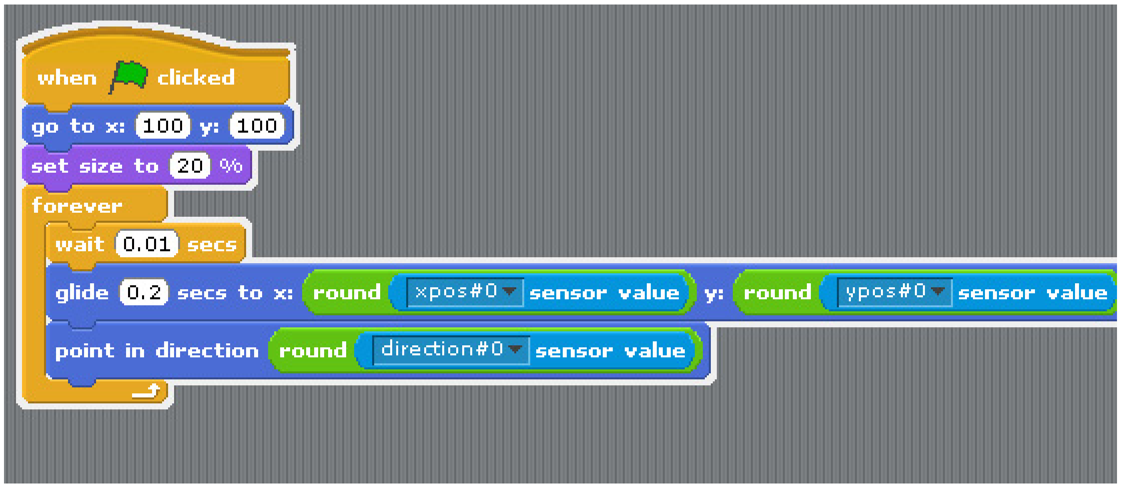


```
when [flag] clicked
go to x: 0 y: 0
set size to 20 %
forever
    wait 0.01 secs
    point towards mouse-pointer
    if distance to mouse-pointer > 5
        move 3 steps
        set xpos#0 to round x position
        set ypos#0 to round y position
        set direction#0 to round direction
```

Changes to the values of these global variables will cause the **sensor-update** messages to be sent to the remote Scratch program. Note that we have used the **round** operation because otherwise the messages keep being sent even if the sprite does not move.

The remote application running for the second player will receive the message and automatically set the remote sensor values to the values corresponding to the global variables of the first player's application.

The second sprite representing the remote player can be controlled with the following logic:



```
when clicked
go to x: 100 y: 100
set size to 20 %
forever
    wait 0.01 secs
    glide 0.2 secs to x: round xpos#0 sensor value y: round ypos#0 sensor value
    point in direction round direction#0 sensor value
```

7. Conventions

This section is still under investigation and has some ideas to resolve issues for programming multi-player Scratch applications. Please send an email to stamati.crook@redware.com if you have any thoughts here (or look on the Scratch Connections wiki). We hope to adopt these conventions in our Server version.

You will come across some programming issues especially if you are try to create an application with more than two players. For example, you may want to broadcast when one player touches the other. The broadcast will be sent over to the remote machine which may have already reacted to the event using a local broadcast.

Some ideas of what is to come are mentioned here. I have used some ideas from the **Snyff** product from **Chalkmarrow**.

> to indicate direction of a Broadcast

Broadcast messages are sent from Scratch whenever a broadcast is made. Use the **>** character in your programming blocks to indicate that a broadcast was created in the local copy of Scratch and the **<** character to indicate that a broadcast message originated in a remote copy of Scratch. The server coverts broadcast names which begin with > to < when sending on to the remote copy.

@ prefix for Server messages and local broadcasts

Prefix a broadcast name with **@** if you do not want it sent on to the remote applications.

This prefix can also be used for server broadcasts and special server remote sensor values. For example, the server might broadcast **@connected** each time a remote player is connected to the application.

#0 suffix for global variables for the current player

Global variables will automatically be passed on to the remote Scratch applications. This convention uses global variables ending in a **#0** suffix for the current player (e.g. **xpos#0**, **ypos#0**, **direction#0**). These variables will be automatically translated into the remote sensor values according to the player number that you are on the remote machine. For example, if you the third player connected then your global variable **xpos#0** is translated automatically by the server to remote sensor **xpos#3** on the remote application.

@servermessage remote sensor value

A general remote sensor value might be used to simplify communication from the server in conjunction with the broadcasts. For example, the player name might be sent in the **@servermessage** remote sensor value just before the **@connected** broadcast message is sent. The Scratch application can then respond and store the value without having to rely on many different sensor values.

@global variables

A special list of global variables might be used to control the server. To set up a new session for a game for example might require the following application blocks:

- @game = “multipong”
- @maxplayers = 4

^ prefix to communicate with sensor boards

This convention is used with the Catenary software from **Chalkmarrow** that communicates with the arduino board. Commands designed to communicate with hardware directly (and not another copy of Scratch) might be prefixed with the **^** character to send messages to the hardware such as **^reset** and so on.