Spout Controls

Control a Sender from a Receiver

spout.zeal.co

Version 1.007

Reference Manual

SpoutControls is a programmer aid that can be used to develop a controlled Spout Sender with C++ or Processing that will react to user controls specific to the application by using a Spout controller.

The Sender does not require a GUI interface that could interfere with the graphics to be shared, or require third party communication to be set up between controller and sender.

The Sender establishes the controls that the controller will use and the updates local variables when the controller has changed any controls.

Controllers are a Freeframe plugin which is itself a Spout Receiver, and a stand-alone executable.

Controllers adapt their user interface to that required by the Sender.

Example controlled Senders for Openframeworks, Cinder and Processing can be used a guide for developing others.

Credit to Eric Medine who came up with the idea.

1. How it works

1.1 Creating the controls

The Sender creates the controls that will be used by the Controller.

These can be Buttons, Checkboxes, Sliders or Text entry.

Controls are identified by name and type. The control name can be anything convenient and identifies the control. This shows up in the user interface of the Controller and is also used when checking the controls after the Controller has changed them.

The control types are as follows :

"event"	Button
"bool"	Checkbox
"float"	Slider float value
"text"	Text input

Each can be created with a default value or string, and a range with minimum and maximum can be set up for a float control.

1.2 Opening communication

The Sender opens communication with the Controller using the sender name. The Controls previously set up are written to a control file with that same name in the "ProgramData\Spout" folder.

At the same time, this name is written to the registry where the Controller can find it.

The Controller uses this name to find the corresponding control file and configures it's user interface accordingly.

Also a communication channel is established so that the Controller can send a message to the Sender when controls have changed.

The Controller creates a memory map containing the data for all the controls. When controls change, this data is updated and the Controller sends a message to the Sender that new data is ready.

1.3 Updating controls

The Sender checks for a Controller message and and receives new control data if any controls have changed. The Sender then updates local control variables for use within the application as required.

1.4 Closing communication

When the Sender closes, the communication with the Controller must also be closed.

The communication channel is closed and the sender name removed from the registry.

The control file still remains and will be used by the Controller if it is started before the Sender. More details on control files are described later.

2. How to use it

Controls are established by the Sender and the Controller adapts the user interface to them. This is described below by way of a rotating cube example for a C++ Spout sender using Openframeworks. Examine the source code for further detail.

2.1 Include required files

The required code is in two files "SpoutControls.cpp" and SpoutControls.h". Include these in your project and create a reference in the application header file.

```
#include "SpoutControls.h"
```

2.2 Create control variables

In the header file, establish variables that will be used for controls.

```
// A SpoutControls object
SpoutControls spoutcontrols;

// Vector of controls to be used
vector<control> myControls;

// Local variables to control cube rotation
float m_RotX, m_RotY;

// Control variable to chance rotation speed
float m_RotationSpeed;

// Control flag to start or stop rotation
bool m_bRotate;

// Control text to be displayed on the sender output
string m_UserText;
```

2.3 Create the controls

In the setup section of the application, the Sender creates the controls that will be used by the controller.

Types are :

"event"	Button
"bool"	Checkbox
"float"	Slider control
"text"	Text entry

Default values can be optionally included. Float controls can be set up with a range from Minimum to Maximum.

Options are :

Create a control

```
bool CreateControl(string name, string type);
```

Create a float or bool control with a default value

```
bool CreateControl(string name, string type, float value);
```

Create a text control with a default string

```
bool CreateControl(string name, string type, string text);
```

Create a float control with maximum, minimum and default values

```
bool CreateControl(string name, string type, float minimum,
                   float maximum, float value);
```

Example code :

```
spoutcontrols.CreateControl("User text", "text", "default text");
spoutcontrols.CreateControl("Rotate", "bool", 1);
spoutcontrols.CreateControl("Speed", "float", 0.0, 4.0, 0.5);
```

2.4 Open communication

The Sender opens communication with the Controller using the sender name.

```
bool OpenControls (string sendername);
```

Example :

```
spoutcontrols.OpenControls(sendername);
```

2.5 Open the controller

Optionally the Windows controller GUI program can be opened at this point unless the SpoutController freeframe plugin is to be used.

```
spoutcontrols.OpenController();
```

2.6 Update the controls

In either the Update or Draw section of the application, the Sender checks for any messages from the Controller.

If a message is received, the control vector will have been updated with new data and the local control variables can then be updated.

This is done using a single function *"CheckControls"*. If the function returns true, a local application function (UpdateControlParameters) is used to retrieve the new control values.

```
if(spoutcontrols.CheckControls(myControls))
        UpdateControlParameters();
```

Typical update function.

```
void testApp::UpdateControlParameters()
{
 for(int i=0; i<myControls.size(); i++) {
        if(myControls.at(i).name == "User text")
               m_UserText = myControls.at(i).text;
        if(myControls.at(i).name == "Speed")
               m_RotationSpeed = myControls.at(i).value;
        if(myControls.at(i).name == "Rotate")
               m_bRotate = ((int)myControls.at(i).value == 1);
 }
}
```

The control names and types match those of the controls originally created and the names are case sensitive.

For simple applications, all controls can be updated in this way, but for situations that require action only when a parameter changes, the check for change can be made here.

For example :

test a button for value 1

test text or float value for change if it activates some function

2.7 Close communication

In the Exit section of the application, the communication between Sender and Controller is closed. The Sender initializes the communication with the controller so must also close it.

```
spoutcontrols.CloseControls();
```

3. Processing

The *SpoutControls* functions are included in the Spout for Processing Library and can be added to an existing Spout Sender sketch. For more detail, look at the example "SpoutBox" sketch.

3.1 Sender declarations

```
import spout.*; // import the Spout library
Spout spout; // declare a spout object
spout = new Spout(this); // create a spout object

// Arrays of controls to be used
String[] controlName;
int[] controlType;
float[] controlValue;
String[] controlText;

String UserText = "";        // Text
boolean bRotate = true;      // Flag
float RotationSpeed = 1.0;   // Float variable
```

3.2 Sender setup

```
spout.createSpoutControl("User text”, “text”);
spout.createSpoutControl("Rotate", “bool”, 1);
spout.createSpoutControl("Speed”, “float", 0, 4, 1);
spout.openSpoutControls("Spout Processing");
```

3.3 Sender operation

In the Draw or Update function.

```
int nControls = spout.checkSpoutControls(controlName, controlType,
                                         controlValue, controlText);
```

The programmer will create an update function unique for each application. For example :

```
for(int i = 0; i < nControls; i++) {
  if(controlName[i].equals("User text"))
        UserText = controlText[i];
  if(controlName[i].equals("Speed"))
        RotationSpeed = controlValue[i];
  if(controlName[i].equals("Rotate"))
        bRotate = (boolean)(controlValue[i] == 1);
}
```

3.4 Sender exit

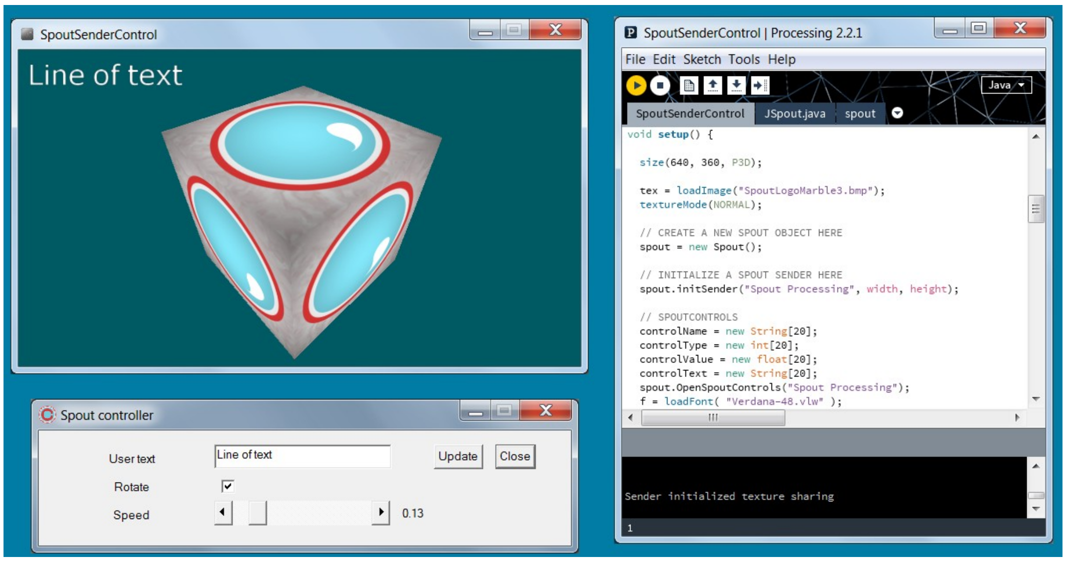
The sender opens the control communication, so must also close it.

```
spout.closeSpoutControls();
```

4. Controllers

4.1 Spout Controller

"SpoutController.exe" is a dialog program that adapts to the controls described in the control file of a sender. A Spout receiver is used to display the sender's output.



```
void setup() {

  size(640, 360, P3D);

  tex = loadImage("SpoutLogoMarble3.bmp");
  textureMode(NORMAL);

  // CREATE A NEW SPOUT OBJECT HERE
  spout = new Spout();

  // INITIALIZE A SPOUT SENDER HERE
  spout.initSender("Spout Processing", width, height);

  // SPOUTCONTROLS
  controlName = new String[20];
  controlType = new int[20];
  controlValue = new float[20];
  controlText = new String[20];
  spout.OpenSpoutControls("Spout Processing");
  f = loadFont( "Verdana-48.vlw" );
```

```
Sender initialized texture sharing
```

Controls will send information to the sender as they are changed.

Text entry will be sent by clicking the "Update" button or by hitting "Enter" or clicking anywhere in the dialog other than a control.

If the sender is closed and opened again, the controller will regain communication and the controls will work.

However, it the sender is closed and a different one opened, the controller has to be re-started.

The *SpoutController* program is not a Spout Receiver, so can be used for any application, regardless of whether it is a Spout sender, and is an easy way to add a simple user interface to Openframeworks, Cinder or Processing applications.

4.2 Freeframe plugin

"*SpoutController.dll*" is a Freeframe Spout Receiver plugin that can be used to control a Spout sender and can be used in any Freeframe host including Resolume, Isadora and Magic.

SpoutController depends on a Spout installation and will not work without it. A button leading to the Spout website for the installer download will be the only control available if Spout is not installed.

Because the *SpoutController* plugin creates sender-specific controls in the host GUI, it can be used for any C++ or Processing application without having to create a specific plugin for each application.

The application has to be a Spout sender in order to be seen in the Freeframe host, but that is very simple to do.

Plugin location

The plugin can be located wherever the host program can find it.

A separate copy of *SpoutController.dll* is required for each controlled sender application, and each application has to be in a folder of the same name as the sender.

For example with Resolume, all FreeFrame plugins will usually be in the plugins\vfx folder. So create a new folder for each controlled sender within the vfx folder :

"C:\Program Files\Resolume Arena 5.0.1\plugins\vfx\MySender" etc..

Then, put a copy of SpoutController.dll in each folder. Setting up folders in this way allows multiple instances of *SpoutController* to be loaded in Resolume and each will detect and react only to the required controlled sender.

Each folder that you create for a controlled sender must contain :

1. the controlled sender application.
2. a copy of the SpoutController plugin.
3. a control file for the sender to be controlled.

The control file

In order to generate the control file, run the controlled sender once. The control file will then be found in the "ProgramData\Spout" folder.

This is usually a hidden folder so, to access it, click the Start Button and enter "c:\programdata\spout" (without quotes) and enter.

The file will have the same name as the sender that has produced it.

Copy the file from "c:\programdata\spout" to the folder of the controlled sender.

For example, “CinderSpoutBox.exe” should be in a folder named “CinderSpoutBox” together with a control file with the name “CinderSpoutBox.txt” as well as a copy of the SpoutController plugin.

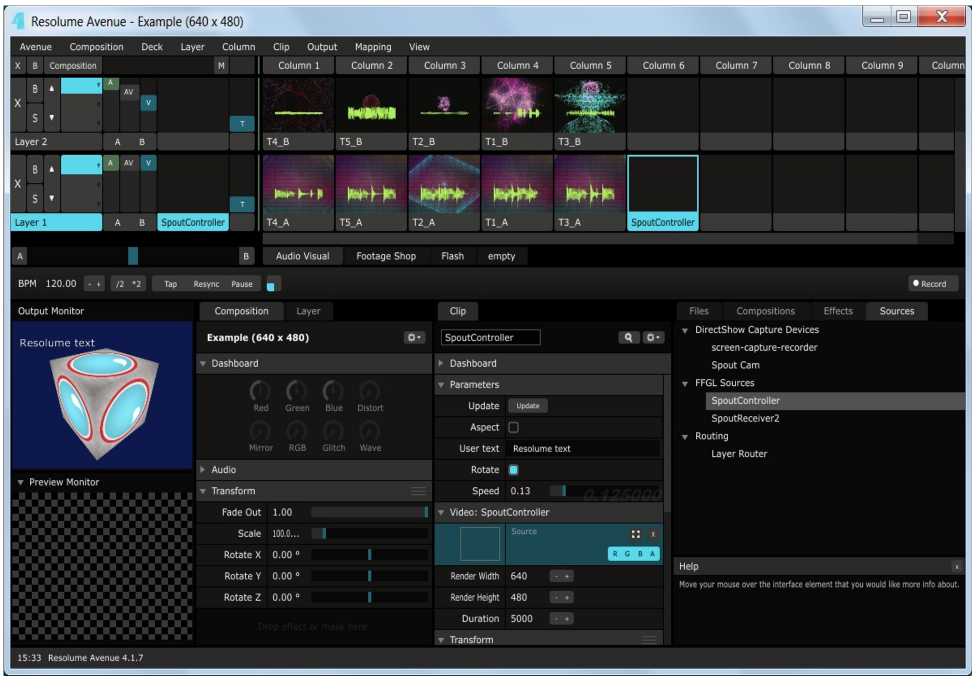
Using the plugin with Resolume

If you followed the steps above, Resolume will show the plugin in “Sources” as *"CinderSpoutBox"* (the first 16 characters).

(Clearly it is useful to create sender names less than 16 characters !).

1. Start Resolume
2. Find “CinderSpoutBox” in the list of “Sources”
3. Drag it to an empty cell

The Cinder sender will be started and the plugin controls will adapt to those established by the sender when it starts..



Multiple controlled senders

Any number of controlled senders can be set up in this way and all will be listed as sources. Each of these names is unique to the respective sender and will receive from and control only that sender and no other.

When a Composition is saved, each instance will also be saved. Then, when Resolume is re-started, each instance of *SpoutController* will again react to only one sender and will “wait” for that sender to be started.

5. Processing for Resolume

By using the *SpoutController* Freeframe plugin you can use a Processing sketch as a Freeframe source, complete with controls customized to the application.

Any Processing sketch can be converted in a matter of minutes to allow variables within the sketch to be controlled by the Freeframe user controls.

5.1 Creating a sketch plugin folder

In the Resolume plugin vfx folder, you need to create a folder with the same name as the controlled sender. For Processing there is already a folder for the sketch itself. e.g. SpoutBox.pde will be in a "SpoutBox" folder.

- Copy the entire sketch folder to the vfx plugin folder.
- Copy the *SpoutController* dll to the sketch folder.
- Run the sketch once to produce a control file.
- Copy the control file from "c:\programdata\spout" to the sketch folder.

For example you will end up with :

```
SpoutBox
    data
    SpoutBox.pde
    SpoutBox.txt
    SpoutController.dll
```

When Resolume is started, the sketch name "SpoutBox" will be seen in the list of FFGL sources, just like any other Freeframe plugin. Drag and drop it to an empty cell. Click on the cell to activate the plugin as usual and the sketch is automatically started.

Be patient while it loads because Processing sketches can take a little while to start. On slower machines this might take 10 or more seconds, so just wait for it.

You won't get the Processing PDE, but rather the output of the sketch is immediately available in Resolume just like any other plugin.

You will notice the Processing icon in the taskbar. The sketch window is there but will have been moved off-screen. You can show it at any time using the "Show sender" button. If it is accidentally closed, you can open it again with "Open sender".

And that's it. You can use the plugin controls just as you normally would. The difference is that behind the scenes you will have the processing sketch running and sending it's texture to Resolume.

Several different sketches can be set up in the same way and they all appear as FFGL sources with their individual names. Each can be used just as a typical Freeframe plugin would be.

A comprehensive tutorial has been created by Eric Medine that steps you though creation of a SpoutController Processing sketch and it's operation within Resolume from start to finish.

The example “CirclesLinesDemo” sketch is included with this distribution, so you can try it straight away.

5.2 Thumbnails

Typically when a Freeframe source plugin is dropped into a cell, a thumbnail image will be displayed of the content that will be rendered. This won't work immediately for the SpoutController plugin because it will not render anything until the cell is activated and the sketch is loaded.

Custom thumbnails can still be created by copying a bitmap image of the same name into the sketch folder. For example for the “SpoutBox” demonstration, copy an image “SpoutBox.bmp” into the folder.

The image can be any size but naturally it is more efficient if it is small because the size is reduced for the Resolume cell thumbnail.

If there is no image in the folder, a default SpoutControls thumbnail is shown.

5.3 Magic

You can used SpoutController in the same way for Magic, but in this case you also have to rename the copy of SpoutController.dll copied to the sketch folder to the same name as the sketch. e.g. in the SpoutBox folder you would rename SpoutController.dll to SpoutBox.dll, and so on for other sketch folders.

There are no thumbnails for Magic and the sketch will be started as soon as the module is placed in the Magic window.

One important thing to note is that if the Magic Window is resized by stretching, the plugin is reloaded multiple times and then many copies of the same sender are generated. To avoid this, use a fixed “Custom” resolution for the Magic Window.

Sketch windows are not closed on exit with Magic as for Resolume, so you have to close them from the taskbar. You can also leave then running if you want to use them in another session. Then they will not need to be loaded again and will appear instantly.

5.4 Isadora

Isadora will only recognise one copy of the SpoutController dll even if it has been renamed, so multiple sketches will not show up in the list of FreeframeGL effects. However a single instance can still be used.

6. Sender examples

Examples enable testing of the Controller operation :

Openframeworks : “ofSpoutBox.exe”

Cinder : “CinderSpoutBox.exe”.

Processing : “SpoutBox.pde”

Processing/Resolume : “CirclesLinesDemo” and “SpoutBox” folders

The Openframeworks and Processing examples include the function to open the Windows controller program.

Examine the source for more detail.

7. Control file details

A text file containing control information is used by the Controller to adapt it's user interface with the controls required by the Sender.

It is created by the Sender by the “OpenControls” function after the required controls have been created using “CreateControl”.

The name of the control file is the same as the Sender name.

7.1 File location

The file is created in a “ProgramData\Spout” folder. For example for the example Sender “ ofSpoutBox” a control file is created :

C:\ProgramData\Spout\ofSpoutBox.txt

The “ProgramData” folder may be hidden but can easily be accessed :

1. Press the Start Button
2. Enter "c:\programdata\spout" into the search without quotes
3. Press the enter key

7.2 File format

The control file has a JSON format described for the ISF (Interactive Shader Format) by VidVox. The principle is the same, except that only the header section is used with the description of each control.

Sections are : CREDIT, DESCRIPTION, CATEGORIES, INPUTS

CREDIT and DESCRIPTION are used if the controller supports them. For example a Freeframe plugin would use the Credits as “About” and the Description as the Plugin description in the “PluginInfo”.

The CATGORIES field is not required but can contain information about the controls that some controller programs might use.

The INPUTS field contains an array of control descriptions (JSON dictionaries). Each describes a control that will will be accessible to the user in the controller program with a structure : NAME, TYPE, DEFAULT, MAX, MIN

NAME – a string containing the name of the control. This is displayed by the controller and also used by the Sender being controlled to identify the control. Although the ISF specification requires that there be no white spaces, this is allowed. However name strings cannot be longer that 16 characters. If the entry is longer than this, the name is truncated.

TYPE – a string describing the control type : “bool”, “event”, “float” or “text”. This departs slightly from the ISF specification to match with the format of Freeframe parameters. An “event” is a button, “bool” is a checkbox, “float” is a slider and “text” is a text input field.

DEFAULT – the default value or string for the control.

MAX, MIN – Float Values are typically 0-1, but can be scaled by specifying Maximum and Minimum values to meet with the range that the Sender application requires.

For example :

```
/*{
  "CREDIT": "spout.zeal.co",
  "DESCRIPTION": "ofSpoutBox",
  "CATEGORIES": [
        "Parameter adjustment"
  ],
  "INPUTS": [
        {
              "NAME": "User text",
              "TYPE": "text",
              "DEFAULT": ""
        },
        {
              "NAME": "Rotate",
              "TYPE": "bool",
              "DEFAULT": 1
        },
        {
              "NAME": "Speed",
              "TYPE": "float",
              "MIN": 0.0,
              "MAX": 4.0,
              "DEFAULT": 0.5
        },
  ]
}*/
```

7.3 Using a control file

The controller will look for a control file in it's own folder first. This allows multiple instances of the SpoutController Freeframe plugin to be set up in separate folders as described above.

If there is no control file present in the folder containing the plugin, SpoutController will look in the registry to find the control file of the last controlled Sender that was started even if the Sender is not running at the time.

If the controller does not find a previous control file, the default control file “Spoutcontrols.txt” is used and should be in the executable or dll path of the controller.

8. SpoutControls function summary C++

8.1 CreateControl

```
bool CreateControl(string name, string type);
bool CreateControl(string name, string type, float value);
bool CreateControl(string name, string type, string text);
bool CreateControl(string name, string type,
                        float minimum, float maximum, float value);
```

Creates a control that will appear for the user.

name	The name of the control
type	The control type, float (numeric value) text (string) event (button) bool (checkbox)
minimum	Minimum control value
maximum	Maximum control value
value	Default control value
text	String for user text input

For example :

```
spoutcontrols.CreateControl("User text", "text", "default text");
spoutcontrols.CreateControl("Rotate", "bool", 1);
spoutcontrols.CreateControl("Speed", "float", 0.0, 4.0, 0.5);
```

8.2 OpenControls

```
bool OpenControls (string sendername);
```

The Sender name is used to create a memory map for communication between Sender and Controller. The name and control file path are recorded in the registry so that the controller can find it. Therefore the Sender should be started before the controller at least once.

8.3 CheckControls

```
bool CheckControls(vector<control> &controls);
```

Checks for a message from the Controller which indicates that new control data is ready. If not ready the function returns false. If it is ready, the new control data is returned in the controls vector and the function returns true. The Sender can then update all it's internal variables to the new control vector values.

The control vector has the form :

```
 struct control {
     std::string name;
     int type; // 0-checkbox, 1-button, 10-float, 100-string
     float value;
     std::string text;
 };
```

8.4 CloseControls

```
bool CloseControls();
```

Used by a Sender to close communication with the Controller. This removes the sender name from the registry. The Controller manages the memory map creation and deletion.

8.5 OpenController

Opens a dialog program as a user interface to the controlled sender. Alternatively the *SpoutController* program can be started at any time after the sender is running or the Freeframe controller plugin can be used.

9. SpoutControls for Processing

Control functions are contained within the Spout Library. They are the same as the C++ versions, except that "checkSpoutControls" uses four control arrays rather than a vector of controls.

```
boolean createSpoutControl(String name);

boolean createSpoutControl(String name, String type,
                           float value);

boolean createSpoutControl(String name, String type,
                           String text);

boolean createSpoutControl(String name, String type,
                           float minimum, float maximum,
                           float value);

boolean openSpoutControls (String sendername);

int     checkSpoutControls(String[] controlName,
                           String[] controlType,
                           float[] controlValue
                           String[] controlType);

boolean closeSpoutControls();

boolean openSpoutController();
```

For example :

```
spout.createSpoutControl("User text", "text");
spout.createSpoutControl("Rotate", "bool", 1);
spout.createSpoutControl("Speed", "float", 0, 4, 1);
```

10. Known issues

Versions of Processing after 3.0.1 do not release java correctly when the Command Line method “processing-java.exe” is used.

The result is that an instance of “java.exe” is left each time the command line is used to run a sketch.

The *SpoutController* Freeframe plugin uses this method and plugins may be started and stopped several times in a session, so to avoid the problem it is better to use Processing version 3.0.1 until this is can be resolved by the Processing developers.