Spout Controls

Control a Sender from a Receiver

spout.zeal.co

Version 1.003

September 2015

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.

This can be done using only four functions.

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

The programmer does not need to modify the Controller itself.

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 the 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 the 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 Creating 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 Opening communication

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

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

Example :

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

2.5 Opening 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 Updating 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 Closing 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 JNI dll for Processing, so can be added to an existing Spout Sender sketch. For more detail, look at the files for the example sketch.

3.1 Sender declarations

```
Spout spout; // 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. Multiple controlled senders

Senders cannot be selected by the *SpoutController* Freeframe plugin because, after the plugin is loaded by the host program, the user controls, unique to each sender, are already established. The plugin cannot subsequently make the host change them.

The solution is to create separate instances of *SpoutController* in separate folders (fortunately, Resolume examines sub-folders within the folder selected for FreeFrame plugins).

Normally the *SpoutController* plugin would have the same name and ID, and Resolume would find only one of them, but *SpoutController* has been designed so that, if a sender control file exists in the same folder, the plugin takes on the same name as the control file.

For example, if a control file with the name “Cinder Spout SDK sender.txt” exists in the same folder as the *SpoutController* plugin, Resolume will list the plugin in “Sources” as *"CINDER Spout SD"* (the first 16 characters).

(Clearly it is useful to create meaningful sender names < 16 characters !).

So, getting back to the point. Each folder you create for a unique controlled sender will contain :

1) A copy of the SpoutController plugin.
2) A control file for the sender to be controlled.

The sender control file is created by running the controlled sender once and 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 this file into the folder that is set set up for this sender instance.

For example with Resolume, all FreeFrame plugins will usually be in the plugins\vfx folder

(e.g. "C:\Program Files\Resolume Arena 4.2.1\plugins\vfx"), so :

Create a new folder for each controlled sender within the vfx folder.

"C:\Program Files\Resolume Arena 4.2.1\plugins\vfx\mySender1"
"C:\Program Files\Resolume Arena 4.2.1\plugins\vfx\mySender2"
etc..

Then, in each folder, copy SpoutController.dll and the control file for the sender to be controlled.

For example, assuming you want to control “Cinder Spout SDK sender”, you would create a folder for the sender :

"C:\Program Files\Resolume Arena 4.2.1\plugins\vfx\mySender1"

(or any other folder name meaningful name for you, it does not matter)

and you would copy into this folder :

SpoutController.dll
CINDER Spout SDK Sender.txt

Then in the “Sources” list in Resolume, the Controller will be identified with the first 16 characters of the Control file name "CINDER Spout SD".

When this instance of *SpoutController* (listed in Resolume as "CINDER Spout SD") is dragged to an empty cell, it will find the control file in it's own unique folder and establish the required controls.

By this method, multiple instances can be established by creating other folders.

Then each instance of the *SpoutController* plugin will be identified in Resolume by the name of the control file. Each instance will set up controls from their individual control files and each will only receive from the sender that matches with that control file.

For example, with the demo programs provided, you might have folders :

"C:\Program Files\Resolume Arena 4.2.1\plugins\vfx\mySender1"
SpoutController.dll
CINDER Spout SDK Sender.txt

"C:\Program Files\Resolume Arena 4.2.1\plugins\vfx\mySender2"
SpoutController.dll
Spout Sender Control.txt

Then in Resolume you will see listed in “Sources” :
“CINDER Spout SD”
“Spout Sender Con”

Each of these is unique to the respective sender and will receive from and control only that sender and no other.

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.

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. Sender examples

Examples enable testing of the Controller operation :

Openframeworks : “ofSpoutSenderControl.exe”

Cinder : “CinderSpoutSenderControl.exe”.

Processing : “SpoutSenderControl.pde”

The Openframeworks example includes the function to open the Windows controller program.

Examine the source for more detail.

6. Controllers

6.1 Freeframe plugin

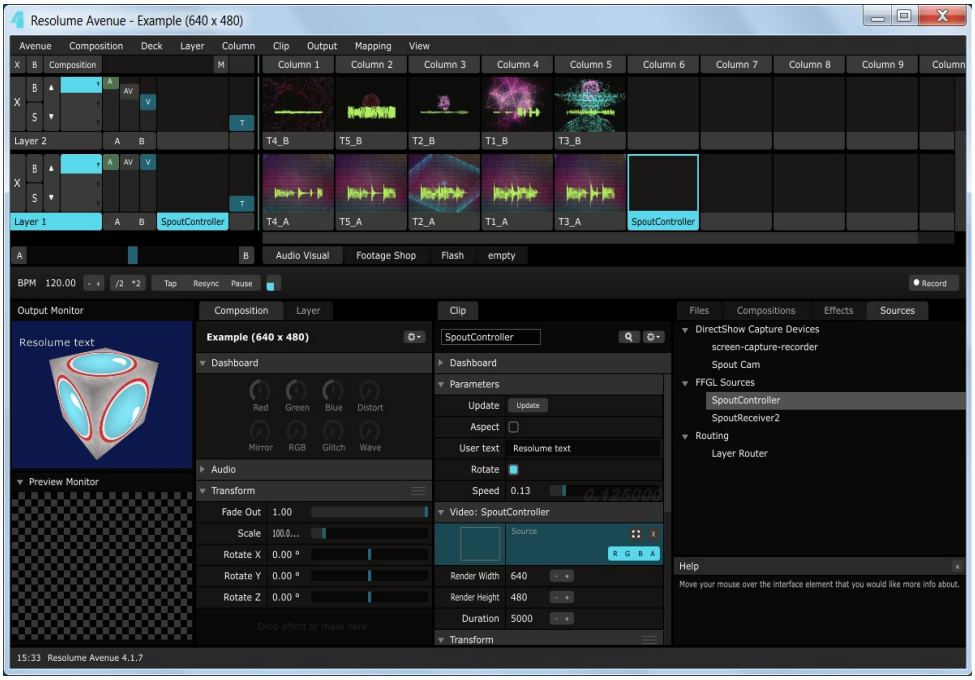
"SpoutController.dll" is a Freeframe Spout Receiver plugin that can be used to control a Spout sender.

It can be located wherever the host program can find it. For example with the Cinder sender and Resolume.

1) Start Resolume
2) Start the Cinder sender "CinderSpoutSenderControl.exe"
3) Find "SpoutController" in the list of "Sources"
4) Drag it to an empty cell

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

There is no sender selection. Only controlled senders will be detected and only the one that was last started will be found. For multiple controlled senders, see more detail above.



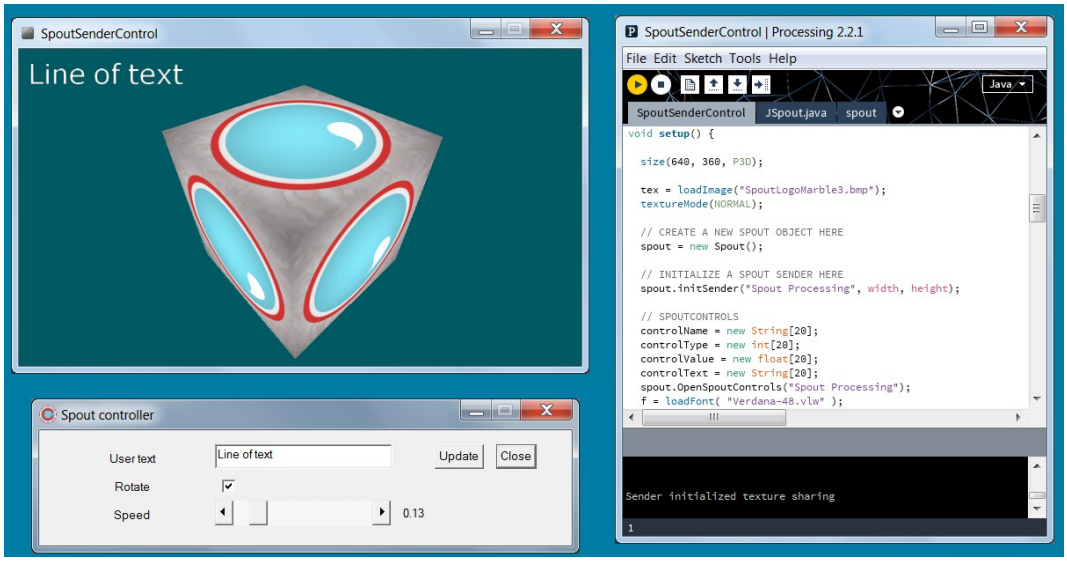
Because the *SpoutController* plugin creates sender-specific controls in the host GUI, it can be used for any C++ or Processing application without having to create a specific Freeframe plugin for each application.

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

The *SpoutController* plugin can be used in any Freeframe host including Resolume, Isadora and Magic.

6.2 Windows Spout Controller

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



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.

The controller can be minimized when not required without losing connection with the sender.

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.

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.

Normally it is not necessary to access this file. 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 “ OF Spout Sender” a control file is created :

C:\ProgramData\Spout\OF Spout Sender.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": "Controller test",
  "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

Only the functions used by a sender are described here.

```
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);
bool OpenControls (string sendername);
bool CheckControls(vector<control> &controls);
bool CloseControls();
bool OpenController();
```

8.1 OpenControls

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

The Sender name is used to provide a name of a mailslot and memory map for communication between Sender and Controller.

This function establishes that name and control file path in the registry so that the controller can find it.

Therefore the Sender should be started before the controller at least once.

8.2 OpenController

This function opens a Windows GUI program so that a user interface is immediately available.

Alternatively the Controller program can be started at any time after the sender is running or the Freeframe controller plugin can be used..

8.3 CheckControls

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

This function checks a “mailslot” for a message from the Controller which indicates that new control data is ready.

If it is 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.

8.4 CloseControls

```
bool CloseControls();
```

This function is used by a Sender to close communication with the Controller. This remove the sender name from the registry. The Controller manages the memory map creation and deletion.