NOVA Server

Introduction

The NOVA Server is a simple Java application that controls the NOVA hardware by directly sending ethernet frames to the hardware via `jnetpcap`. In addition to controlling the NOVA hardware (composed of 5x5x10 voxel NOVA modules), the NOVA Server provides a simple Web interface to control the content playback. Content playback can be controlled as well through a proprietary protocol for Apps running on mobile devices. The HTTP server runs on port 80 whereas the proprietary server runs on port 55555. The server is invoked by calling the `main()` method on `ch.digisyn.nova.NOVAControl` class and passing a path to a configuration file (see “Configuration” below) as its first argument.

Configuration

The server is configured by a configuration file which is the only argument passed to the server’s `main()` method. The following keys are supported by the server:

- **brightness:** The initial brightness value in the range [0..1]
- **nova:** The ethernet interface name to use for sending content and control packets to the NOVA (e.g. “eth0”)
- **sync:** The ethernet interface name to use for sending sync packets to the NOVA (e.g. “eth0”)
- **http:** The IP address the internal web server binds to
- **tcp:** The IP address the NOVA server binds to. See “Protocol” below for a description of the NOVA server protocol.
- **addr_<X>_<Y>:** The address (as configured by jumpers on the NOVA board) of the module at module location (X,Y)
- **content:** A comma separated list of the names of content classes to load at startup from the `ch.digisyn.nova.content` package. See “Writing a Content Extension” below for writing a custom content extension.
- **duration:** The number of seconds a content class should run if it does not control its running time by itself (e.g. a movie will stop by itself after the last frame showed).

A sample configuration may look as follows:

```
brightness=0.6
nova=eth1
sync=eth1
http=127.0.0.1
tcp=127.0.0.1

addr_0_0 = 1
addr_0_1 = 5
addr_0_2 = 9
addr_0_3 = 13
addr_1_0 = 2
addr_1_1 = 6
addr_1_2 = 10
addr_1_3 = 14
addr_2_0 = 3
addr_2_1 = 7
addr_2_2 = 11
addr_2_3 = 15
addr_3_0 = 4
addr_3_1 = 8
addr_3_2 = 12
addr_3_3 = 16

content=Colorcube,Random,Movie,Sweep
duration=300
```

Hardware Address Configuration

The modules derive their MAC as well as IP address from a jumpers on the board. The jumpers encode the least significant byte of the MAC address and also the last 8 bit of the IP address in the 192.168.1.0/24 subnet. Traditionally, modules are in the range 192.168.1.1-192.168.1.100 and the server has the address 192.168.1.130.

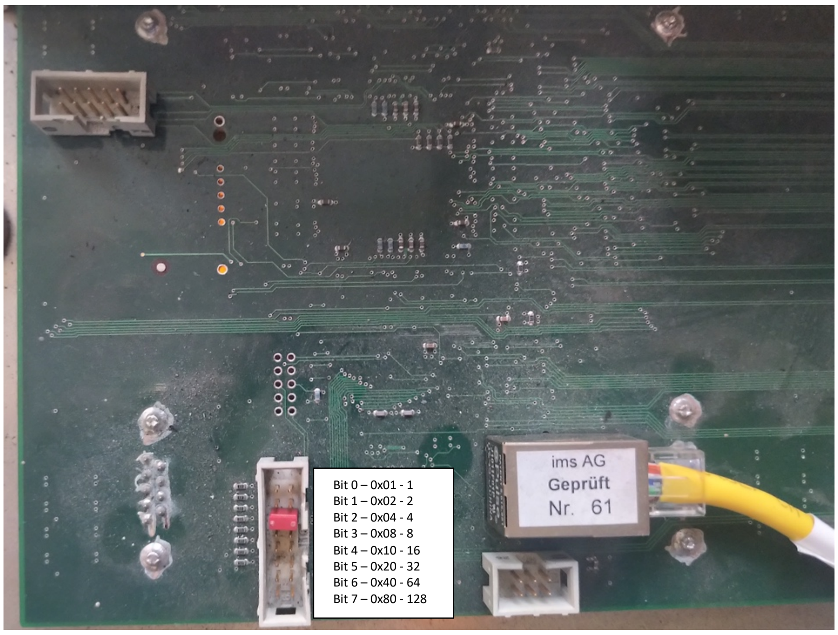

Figure 1 - NOVA module set to address 4

Protocol

The proprietary protocol is stream based and currently runs over TCP port 55555. When a client first connects to the server it receives a comma separated (",") list of available content from the server encoded as UTF-8. From then on, the server listens for the following commands from the client:

- **lum*LL***: Set luminance (brightness). *LL* is the brightness value [0..1] mapped to [0..255] encoded as hexadecimal number [00..FF].
- **vel*VV***: Set velocity (playback speed). *VV* is the speed value [0..1] mapped to [0..255] encoded as hexadecimal number [00..FF].
- **col*RRGGBB***: Set color factors. *RR*, *GG*, and *BB* are the red, green, and blue components [0..1] mapped to [0..255] encoded as hexadecimal number [00..FF] which are multiplied with the current content frame RGB values in order to yield the final output color.
- **anim*<animation name>***: Select animation by name. The name must be exactly the same UTF-8 sequence as received from the server.

Each command is executed immediately by the server after the last byte of the command is read from the stream. They client must flush the stream at the end of each command in order to make sure that the command gets transmitted over the stream.

Writing a Content Extension

The server loads content extensions as configured from a predefined package. Adding classes to this package makes them available to the server. A content extension must inherit from the `ch.digisyn.nova.content.Content` class and implement at least a constructor and the `fillFrame()` method. Below is the implementation of the content class:

```
package ch.digisyn.nova.content;

import java.util.Collections;
import java.util.List;

/**
 * Content base class for NOVA Server content.
 *
 * @author sschubiger
 */
public abstract class Content {
	/* name of content. */
	protected final String name;
	/* X-dimension in the horizontal plane of the NOVA hardware, always a multiple of 5. */
	protected final int    dimI;
	/* Y-dimension in the horizontal plane of the NOVA hardware, always a multiple of 5. */
	protected final int    dimJ;
	/* Z-dimension (vertical) of the NOVA hardware, always 10. */
	protected final int    dimK;
	/* number of frames to run. */
	protected final int    numFrames;
	/* number current number of frames. */
	protected       int    frames;

	/**
	 * Creates a content instance.
	 *
	 * @param name The name of the content.
	 * @param dimI The X-dimension.
	 * @param dimJ The Y-dimension.
	 * @param dimK The Z-dimension.
	 * @param numFrames The number of frames to run.
	 */
	protected Content(String name, int dimI, int dimJ, int dimK, int numFrames) {
		this.name      = name;
		this.dimI      = dimI;
		this.dimJ      = dimJ;
		this.dimK      = dimK;
		this.numFrames = numFrames <= 0 ? Integer.MAX_VALUE : numFrames;
	}

	/**
	 * Utility function to set a RGB values of a voxel at position (i,j,k).
	 * @param rgbFrame The voxel frame to operate on.
	 * @param i The X-position.
	 * @param j The Y-position.
	 * @param k The Z-position.
	 * @param r The red value.
	 * @param g The green value.
	 * @param b The blue value.
	 */
	protected void setVoxel(float[] rgbFrame, int i, int j, int k, float r, float g, float b) {
		final int idx = 3 * (k + (dimK * (i + j * dimI)));
		rgbFrame[idx+0] = r;
		rgbFrame[idx+1] = g;
		rgbFrame[idx+2] = b;
	}

	/**
	 * Called when a content is activated.
	 */
	public void start() {frames = numFrames;}
	/**
	 * Called when a content is deactivated.
	 */
	public void stop() {}

	/**
	 * Request a voxel frame to be filled. Must complete in less than 40ms in
	 * order to keep up with the
	 * 25 Hz frame rate of the display.
	 *
	 * @param rgbFrame The voxel frame to operate on.
	 * @param timeInSec The relative animation time starting from 0.
	 * @return True if this content has more frames avilable or fals if the server
	  * should switch to the next content.
	 */
```

```
        public abstract boolean fillFrame(float[] rgbFrame, double timeInSec);

        /**
         * Returns the name of the content.
         */
        @Override
        public String toString() {
                return name;
        }

        /**
         * Called by the server to get a list of content instances.
         *
         * @return The list of content instances. Usually just this instance.
         */
        public List<Content> getContents() {
                return Collections.singletonList(this);
        }
}
```

An actual implementation of a procedural content is as below:

```
package ch.digisyn.nova.content;

/**
 * Top-down sweep content.
 *
 * @author sschubiger
 */
public class Sweep extends Content {
        /**
         * Creates a content instance.
         *
         * @param name The name of the content.
         * @param dimI The X-dimension.
         * @param dimJ The Y-dimension.
         * @param dimK The Z-dimension.
         * @param numFrames The number of frames to run.
         */
        public Sweep(int dimI, int dimJ, int dimK, int numFrames) {
                super("Sweep", dimI, dimJ, dimK, numFrames);
        }

        /**
         * Fill the frame.
         */
        @Override
        public boolean fillFrame(float[] rgbFrame, double timeInSec) {
                final double dimK_1 = dimK - 1;
                // loop over all voxels
                for(int k = 0; k < dimK; k++) {
                        double dk    = k / dimK_1;
                        // compute value based on time from a sine curve
                        float  v     = (float)Math.abs(Math.sin(dk * Math.PI + timeInSec));
                        for(int i = 0; i < dimI; i++)
                                for(int j = 0; j < dimJ; j++)
                                        setVoxel(rgbFrame, i, j, k, v, v, v);
                }
                // return true as long as we are allowed to playback frames
                return --frames > 0;
        }

}
```

Please be aware that `fillFrame()` must complete in 40ms otherwise a frame underrun will occur. If the content is too complex to render in real time, it can be written to a flat file of RGB voxels and played back with the `ch.digisyn.nova.content.Movie` class.