Praxis LIVE

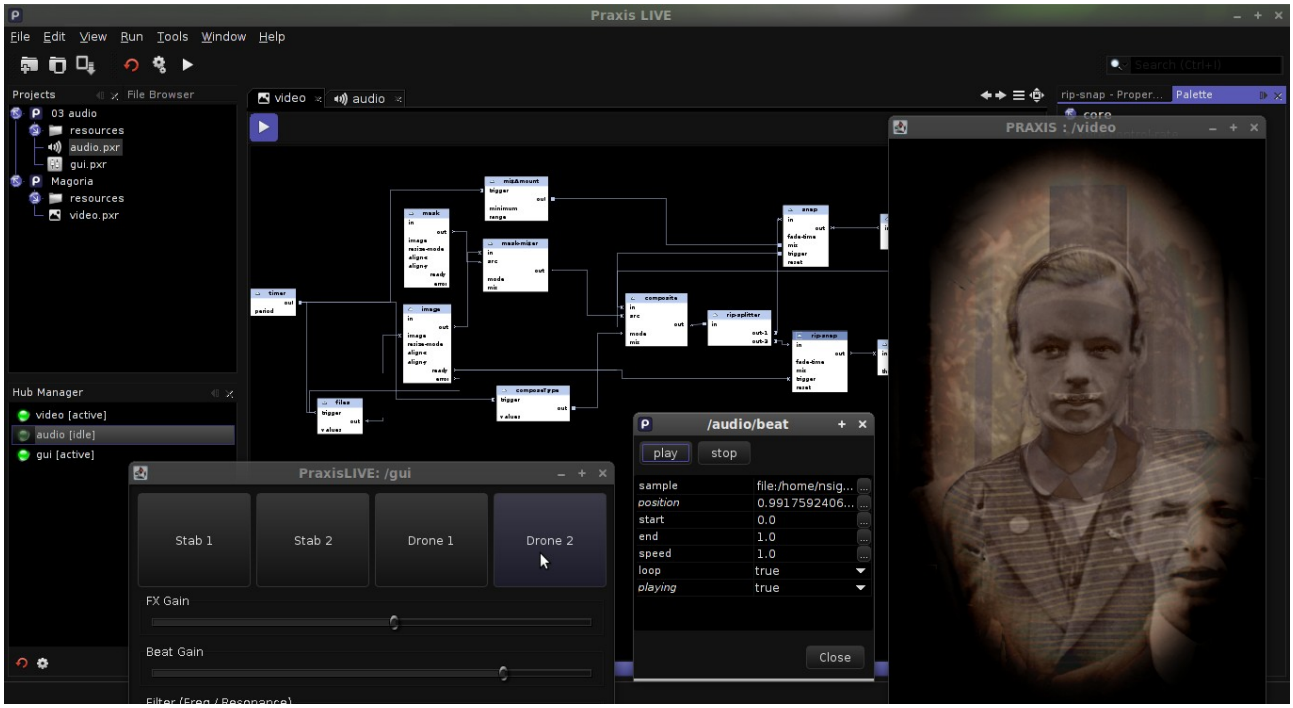
Manual (work in progress)

Neil C Smith

Contents

Introduction

Praxis LIVE is an open-source, graphical environment for rapid development of inter-media performance tools, projections and interactive spaces. It works cross-platform and is developed by UK Artist & Technologist Neil C Smith (http://neilcsmith.net).



Primary Features

- **Media neutral architecture.** Built from the ground up for working with multiple media, using an architecture based around models for distributed processing.
- **Intuitive patcher-style graphical editor.** Fast visual project building with drag-and-drop creation of components and connections.
- **Edit everything live.** Built around the central concept that everything should be editable at run time.
- **Optimized video pipeline,** including a range of pixel effects and blend modes, with software and OpenGL rendering.
- **Low latency audio** support, including JACK binding.
- **Custom GUI's and MIDI control.** Create custom controls panels bound to any parameter, or control any parameter using a MIDI controller.
- **Live coding.** For when the built-in component don't offer all you need, create custom components 'on-the-fly' using Java / Processing or GLSL.

Praxis / Praxis LIVE

Praxis is the modular framework at the core of *Praxis LIVE*. It provides the basic runtime architecture and components used within a *Praxis* project, but no UI of its own besides a simple command line player (included in the *Praxis LIVE* installation).

Praxis LIVE provides a graphical interface for creating, editing and controlling *Praxis* projects. It is the primary interface for working with the *Praxis* framework.

Keeping the two aspects distinct also allows for the core framework to be used in other ways – eg. shipping *Praxis* projects as stand-alone executables.

Architecture / Terminology

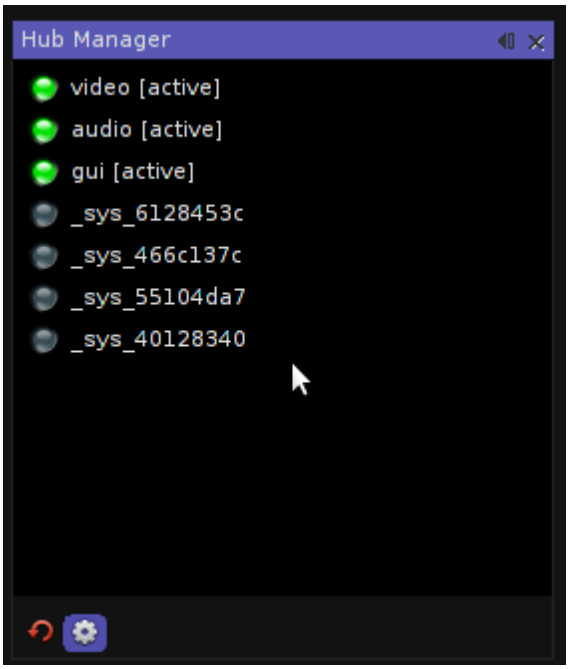
A Forest not a Tree

Praxis has an architecture influenced by message-passing models for distributed / concurrent programming (in particular the Actor Model). It is the strict adherence to this architecture that is the key strength of *Praxis*, allowing for working with multiple media without interference (or multiple pipelines of the same media such as video at different frame rates). It also provides the ability to background load resources, process data, etc. without interfering with playback.

Individual components within *Praxis* (such as a video effect, sample player, button, etc.) exist within a tree of components, which can be multiple levels deep. All components have an address, which follows a familiar slash-separated syntax (eg. `/audio/delay1`). The first level of this tree is a special type of component called a **Root**. Roots generally provide the media context for the components that exist within them, and there are currently 4 types of root available within the standard *Praxis LIVE* install – audio, video, MIDI and GUI. While you would commonly have one of each type you require, it is perfectly possible to have as many roots of the same type as your system can handle.

All roots exist within the **Root Hub**. The root hub is a container for roots, but it is not itself a component. All roots are sand-boxed from each other – there is no way for any root or component within it to directly access another root or its components. Instead, the hub acts as a router to pass messages between different roots.

Here you can see the **Hub Manager** within the *Praxis LIVE* application, which gives you a visual representation of the root hub. You will notice that there are 3 user roots running (audio, video and GUI), but that the button is toggled to also show you the system roots. This is another key part of the architecture – all of the system code is equally sand-boxed and confined to using the message-passing system.

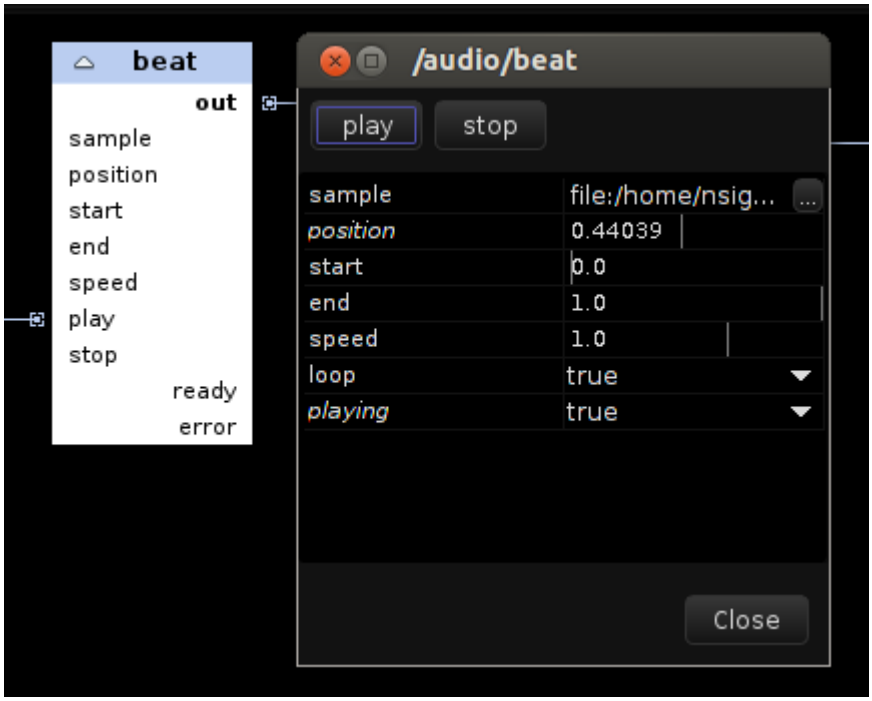


Praxis LIVE Hub Manager

Ports and Controls

Components within Praxis have two ways of communicating – **Ports** and **Controls**. Ports are used for instantaneous communication with sibling components (share the same parent). When you connect components by drawing lines in the *Praxis LIVE* graph editor, you are connecting ports together. They are a lightweight mechanism for sharing control data, video frames, audio buffers, etc. Although you are unlikely to need to know this, ports also have an address (eg. `/audio/delay!in`).

Controls are the basis of *Praxis*' message-passing system. They receive, react and respond to messages usually coming from another component within another root. All communication (*without exception*) between components in different roots ends with a control receiving a message. Controls have an address consisting of their component address and ID (eg. `/audio/delay.time`) – the dot syntax is a deliberate parallel with method calls.



Ports and Controls of an audio:sampleplayer in Praxis LIVE

Above you can see an example of an `audio:sampleplayer` component from within the *Praxis LIVE* editor. Ports can be seen on the graph component and controls within the component editor window – note that many controls will have a corresponding port, and vice versa.

When you manipulate values within the dialog, you are sending messages to controls on the component. There are three different types of controls – **actions**, **properties** and **functions**. Actions are represented by the buttons at the top of the dialog, and properties within the table. Functions are not currently reflected within the editor, but are primarily used internally.

The `ready` and `error` ports reflect the fact that samples cannot be loaded immediately without the audio breaking up. Therefore, this request is passed to one of the system roots that handles background loading of resources. Once the request is completed then a message is sent from the ready or error port depending on whether the sample could be loaded or not. Other components that load resources use the same methodology.

The Praxis LIVE Project System

The core *Praxis* system is driven by its own simple scripting language. If you're using *Praxis LIVE* you don't need to know this as the project system allows you to build projects graphically.

Praxis LIVE projects are actually directories. Each root in your project (video patch, audio patch, control panel, etc.) is stored in a separate file. These files have the `.pxr` (Praxis root) file extension.

The scripts that make up a *Praxis LIVE* project are divided into **Build Level** files and **Run Level** files. It is important to remember that everything is edited live, so only components that are actually installed in the hub can be edited. For this reason, you can build a project to enable editing without needing to run it (start video / audio playback, etc.). As you develop a project, the system tracks what files have and haven't been executed, so that you can always build and/or run a project at any time.

It is possible to work with multiple projects concurrently, as long as there is no collision between root ID's – all root components installed within the hub at any one time must have a unique name.

The main project directory also contains a `resources` directory. It is recommended that you store all media required by your project in the `resources` directory, or a sub-directory of it. This allows for projects to be completely self contained and makes distributing them easier.

If you look at a project directory in your file browser you will also see a range of configuration files that are normally hidden from view. Do not alter any of these. All files that define the project are actually valid *Praxis* scripts, allowing for projects to also be run using the *Praxis* command line or distributed as stand-alone applications. When running outside of *Praxis LIVE* there is no distinction between build level and run level files.

A more in-depth explanation of Praxis' architecture and scripting language can be read at http://praxisintermedia.wordpress.com/2012/07/26/the-influence-of-the-actor-model/

Getting Started

Praxis LIVE can be downloaded from the Download tab of the Praxis Google Code site (http://code.google.com/p/praxis/downloads/list), in either installed or non-installed form. Installation is recommended as this creates things like short-cuts and menu entries automatically. Some installers are also a smaller download as they use a different form of compression – the functionality is the same.

System Requirements

Praxis LIVE requires Java 6 or above, either Oracle's distribution or OpenJDK.

Video playback and capture require use of the GStreamer library. Linux and OSX users currently require a system installed version of GStreamer. There is a *Praxis LIVE* plugin available for Windows on the download page that will install a private version of GStreamer – this is the recommended approach.

Installation

At the time of writing installers are only available for Windows and Linux – MacOSX users will have to follow the instructions for non-installed usage.

WINDOWS

If you have an earlier version of *Praxis LIVE* already on your system then you should uninstall this before installing the new version. The uninstaller can be found in the directory in which you installed *Praxis LIVE (*you can also uninstall from the control panel). It is advisable to select the option to delete the existing user directory.

Download and run the installer. You may need to give it permission to make changes to your system. The installer will (by default) create a menu entry and desktop shortcut, and offer to start *Praxis LIVE* directly after installation.

LINUX

There are two methods of installing on Linux, a `.sh` file and an (experimental) `.deb` file. The .sh file can be run without admin permissions and will install into your user home directory.

If you are using the `.sh` installer and have an earlier version of *Praxis LIVE* already on your system then you should uninstall this before installing the new version. The uninstaller can be found in the directory in which you installed *Praxis LIVE.* It is advisable to select the option to delete the existing user directory.

Download the `.sh` installer. You may need to mark the file as executable before it will run. Run this file and it will install *Praxis LIVE* onto your system.

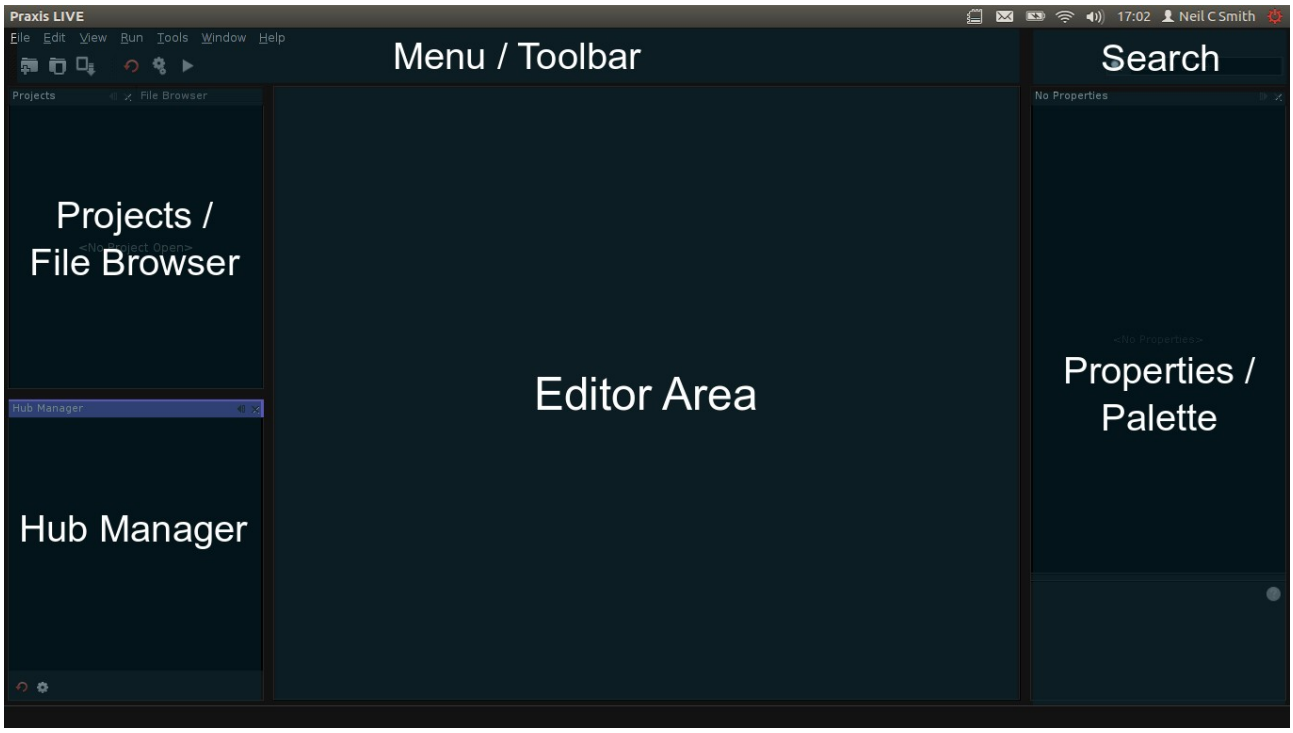
The `.deb` installer should work on any Debian / Ubuntu derived system. This method requires admin permissions, and will install like any other software on your system. The .deb file should open by default in GDebi or Ubuntu Software Centre.

Non-installed Usage

To use *Praxis LIVE* without installing it on your system, download the `.zip` distribution from the Downloads page. Unzip the archive into a suitable location. Inside the Praxis LIVE directory you will find a `bin` directory. Run either `praxis_live` (Linux / OSX) or `praxis_live.exe` (Windows) from inside the `bin` directory – you may wish to create a short-cut to make this easier.

The *Praxis LIVE* interface

The image below outlines the default window layout. *Praxis LIVE* uses a typical MDI interface, whereby components and documents being edited are encompassed within a single main window. It is possible to move components around the interface or even un-dock them into a separate window. Any changes you make will be remembered – if you want to reset to the default layout, use the `Window / Reset Windows` action in the main menu.



Default window arrangement of Praxis LIVE

Toolbar

Six common actions have buttons in the main toolbar.

New Project. This will start the new project wizard, which will allow you to create a new project.

Open Project. This will open the project browser allowing you to open a pre-existing project. The project will open in the Project window, but it will not be run.

Save File. This will save the file currently being edited. NB. It does not save the entire project.

Restart Hub. This will restart the *Praxis* hub. Any root components will be removed. A dialog will open and give you the option to save any files.

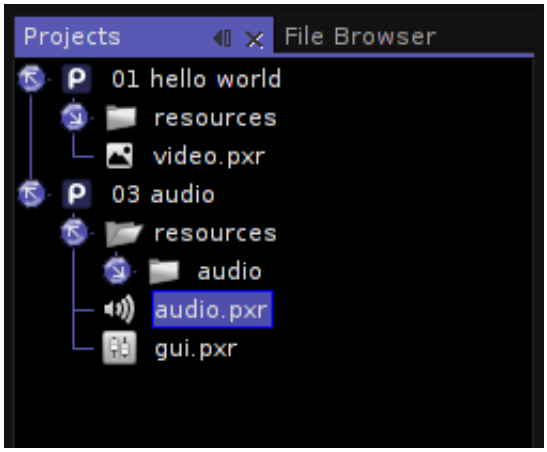
Build Project. Build the selected project. This will install any components in the hub, but not start playback.

Run Project. Run the selected project. This will build the project (if necessary) and then start any roots set to run automatically.

Projects

The `Projects` window gives you the tools for opening and managing projects. Projects are shown in a tree-like structure which shows you all the (user-editable) files that make up a *Praxis LIVE* project.

Here you can see two of the example projects opened within the project window. Note the various `.pxr` files which define different roots (video patch, audio patch, control panel).



The Projects window

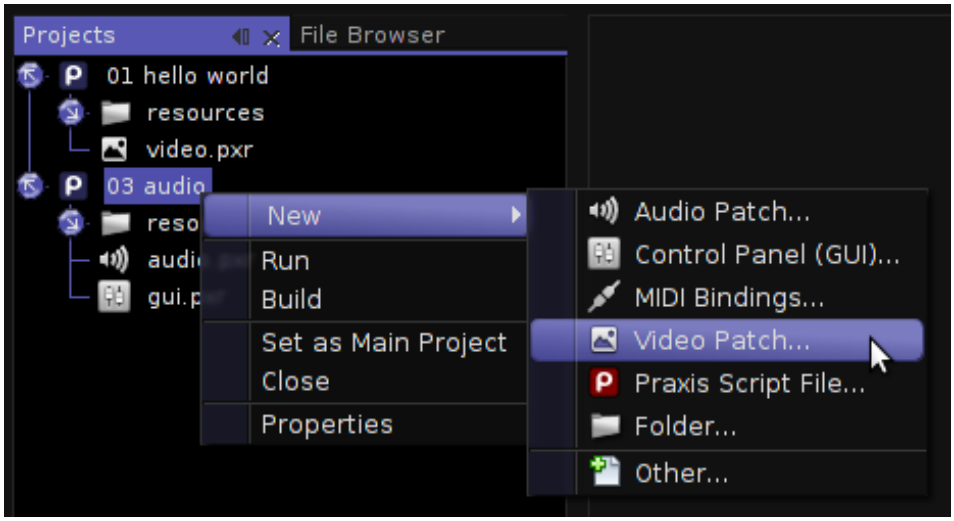
The `resources` directory provides the recommended location to store resources (images, audio files, etc.). Paths to files stored in this directory are relative, meaning that project directories can be easily distributed.

To run a project, select it and use the buttons on the toolbar, or use the pop-up menu on the main project directory (eg. `01 hello world`).

To stop and clear a running project, restart the hub – you will be given the option to save any changes. Unless you are using the ability to work with multiple projects (advanced), *get into the habit of restarting the hub before trying to run another project.*

Double-clicking a `.pxr` file will open it in an editor window (see the Editors section).

You can create a new `.pxr` file, such as an audio or video patch, from the project's pop-up menu. Selecting one of these options will open a wizard which will allow you to provide the root ID, which will also be used as the name of the file. Advanced options allow you to control whether the root will auto-start when the project is run, and whether to install the root immediately in the hub for editing.



Creating a new Video Patch

To import resources into your project, you can use the `File Browser` window to find them, then use copy and paste to bring them into your `resources` directory or a relevant sub-directory. You can also use your system file browser for this purpose. An `Import Files` wizard is in development to make managing this process easier.

The `Properties` option in the project's popup menu will open the project configuration dialog. Within this you can control which files are included within the Build and Run stages of a project. You can use this to control which roots auto-start, or to temporarily ignore certain files without deleting them.

You can also use this to fix the project configuration, in rare cases where *Praxis LIVE* is terminated unexpectedly.

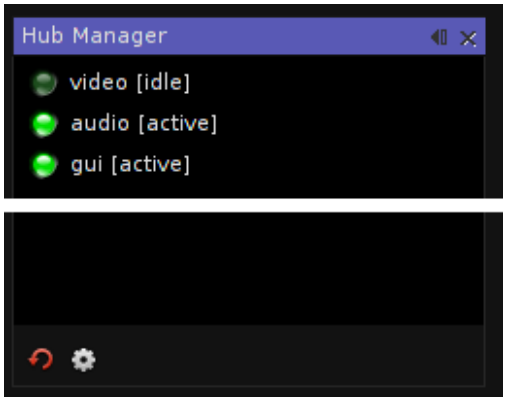
Hub Manager

The `Hub Manager` window gives you an overview of which roots are currently running within the *Praxis* hub.

The LED icon to the left of the root ID shows you whether the root is running or not. You can start or stop individual roots by double-clicking on them.

You can also remove individual roots from the hub by using the pop-up menu on the root and selecting `Delete`. You will be given the option to save the current state before the root is removed.

There are two buttons at the bottom of the window. The first duplicates the `Restart Hub` action from the main toolbar. The second allows you to toggle whether system roots are shown within the hub manager.



The Hub Manager window

System roots are normally hidden, and provide various functionality including component creation, script interpreting and background loading. One is the gateway by which *Praxis LIVE* communicates with the rest of the *Praxis* system. Notice that the ID's of the system roots change whenever the hub is restarted.

Properties / Palette

The `Properties` window allows you to view or edit the properties of the currently selected item, such as a file in the `Projects` window or a *Praxis* component in the editor.

The `Palette` window shows a range of *Praxis* components that can be added to the patch being currently edited.

There is more information on these two windows in the Editors section.

Search / Help

The primary function of the `Search` box is to allow you to search through documentation for all of the various *Praxis* components in the online help. Due to the continuous development of *Praxis*, individual components are not described within this document but within the program itself. You can also access the online help from the `Help` menu item, or by pressing `F1`.

As well as help files, the `Search` box also allows you to search for common actions and options.

You can press `CTRL-I` to focus the `Search` box without using the mouse.

Options / Settings

The `Options` window can be opened through the `Tools` menu in the main menu bar. It provides the ability to set common default settings such as the video renderer, audio sample-rate, etc. These settings can be overridden within individual root (patch) properties if required.

Settings are stored at a system level, so also apply to the command line player and to stand-alone projects (though again this behaviour can be overridden).

Editors

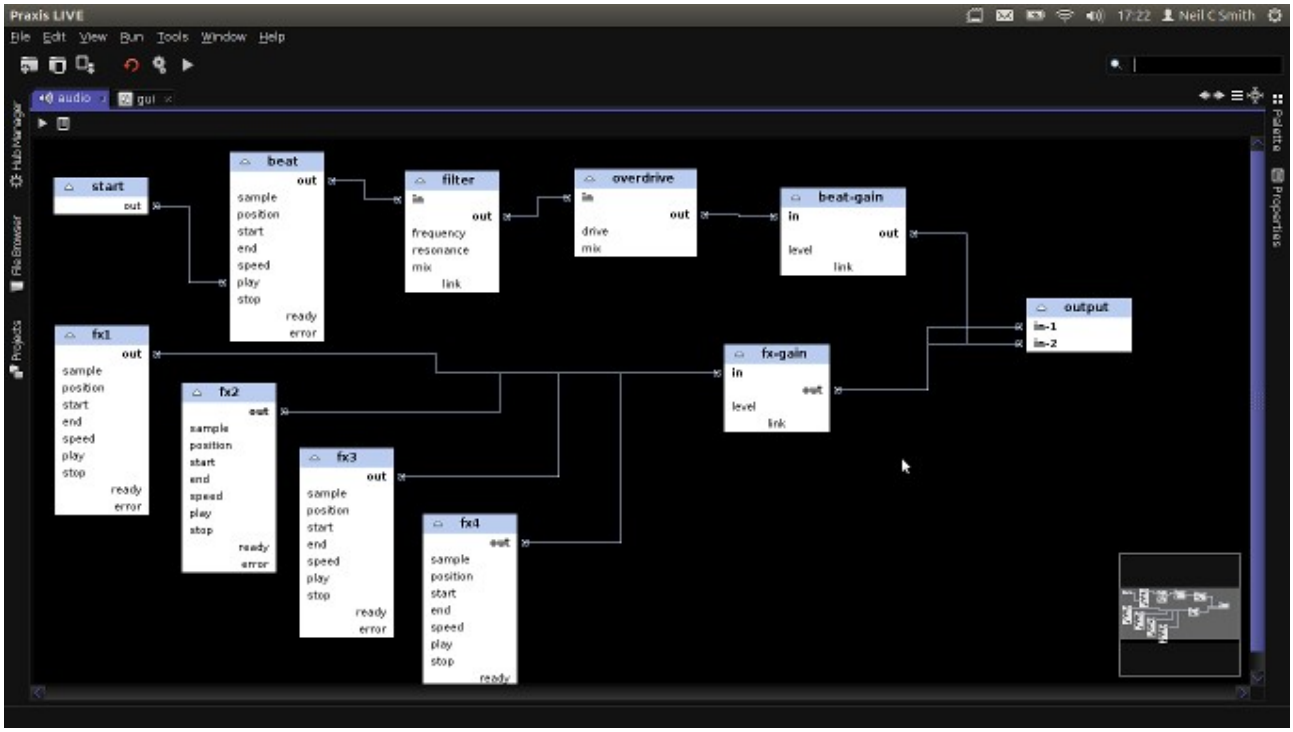
Double-clicking on a `.pxr` file will open it in an editor. The type of editor it opens in will depend on the root type (video, audio, gui, etc.). Remember that the project the file belongs to must have been built so that the root shows up in the `Hub Manager` before the file can be edited, otherwise the editor will contain a message asking you to build the project.

At the top-right of the editor region are four icons. Three of these allow you to switch between open editors. The maximise icon allows you to enlarge the editor area, minimising all the other windows to the sides and providing more workspace.

All editors have their own toolbar, which contains at least two buttons. The play button toggles whether the root is playing / active. The root properties button opens a window allowing you to set properties of the root component (such as video resolution and frame-rate). Not all root properties can be set while the root is playing.

You can save a `.pxr` file at any time when the editor is active. You will also be prompted to save whenever the hub is restarting or when a root is removed from the hub. Due to the distributed architecture, *Praxis LIVE* does not attempt to track all changes, so takes the conservative approach and always prompts you to save.

Graph Editor (audio / video)



Graph editor in maximised editor region.

Audio and video root files will open up in the graph editor. Components and port connections are represented in a graphical form that should be familiar to anyone who has used other patcher-style environments like AudioMulch, Isadora, Max, etc.

To **add components**, drag the component type you want from the `Palette` window onto the editor. The palette automatically filters to valid components depending on the root type (audio, video, etc.) You will be asked to name the component (a unique name will be suggested) – names cannot be changed once the component is created.

To **link ports**, click and drag with the mouse between the two ports. You cannot connect ports of incompatible types, and output ports can only be connected to input ports. Audio and video ports are highlighted in bold.

Components and port connections can be removed by selecting them and pressing `Delete` or selecting delete from the pop-up menu.

Multiple components can be selected by clicking and dragging a selection box around them, or by `CTRL-clicking` on each component.

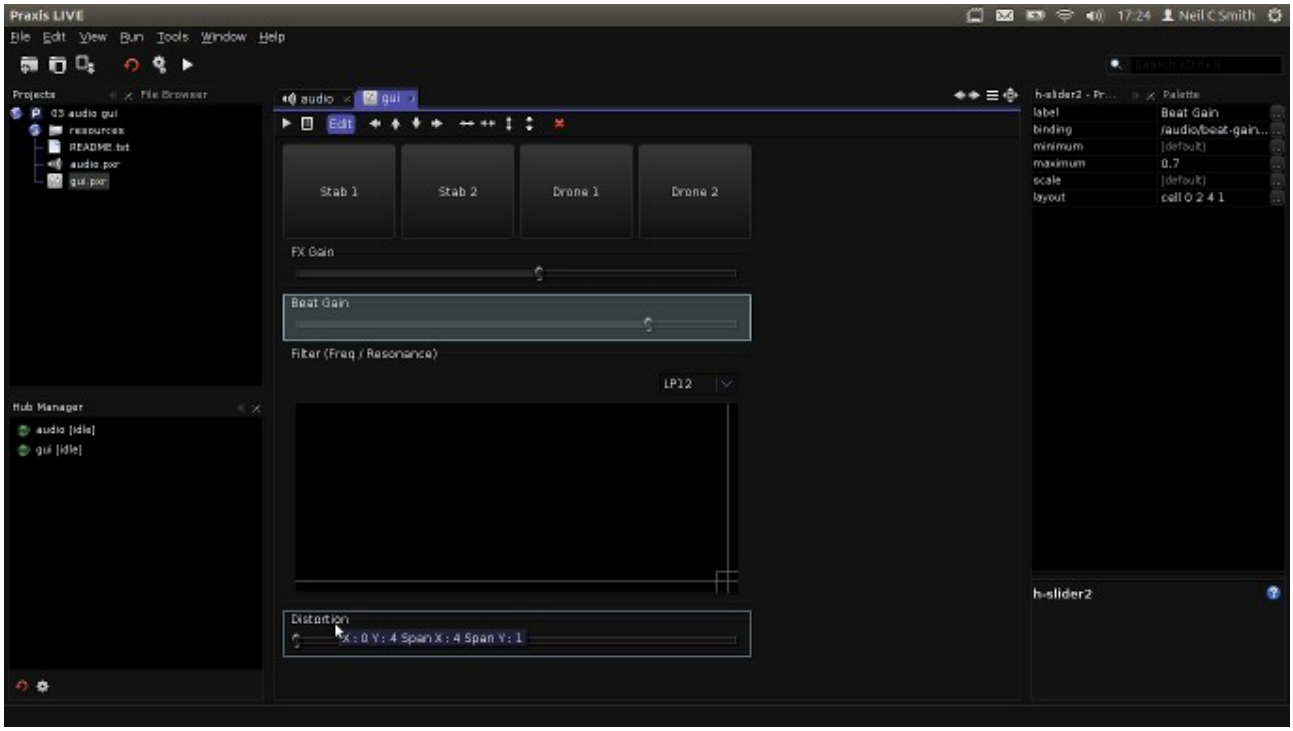
To **control or edit component properties**, double-click on the component to open the component editor window. This window includes buttons for all component actions and a table of all properties (more information in the *Component Editors* section). Component actions are also accessible through the pop-up menu on components. When selected, component properties also show up in the `Properties` window – if multiple components are selected you can use this window to control all of them together.

It is possible to **zoom in and out** of the graph editor using `CTRL-mousewheel`, and **navigate** around using the satellite view in the bottom-right of the editor. This makes it easier to spatially organise components.

While it is usually possible to add and remove components at runtime, be aware that currently inputs (audio) and outputs (video / audio) are only picked up when the root is started, and only one of each of these type of components may be used in any patch.

For example, if you add an audio input while audio is playing, you will not hear anything until you stop and restart the audio playback.

GUI (Control Panel) Editor



Audio GUI example open in the GUI editor

When you open a GUI / control panel `.pxr` file it will open in the GUI editor. If the GUI was open (running), the external window will be closed and the panel "hijacked" into the editor. The control panel is still fully functional when open in the editor.

To make changes to the control panel you need to enable the edit overlay. Toggle the `Edit` button in the editor toolbar (or use `CTRL-E`). Component boundaries will now highlight as you hover over them and you can select component by clicking on them. Switch off the edit overlay (using the same button / keys) to re-enable use of the control panel.

Add components to the control panel by dragging them from the palette onto the control panel (the edit overlay must be enabled). You can insert components by dropping them on top of existing components. Unlike the graph editor, the GUI editor gives all components an automatic name.

Edit component properties by double-clicking on them to open the component editor window, or selecting them and using the `Properties` window. The GUI editor does not currently support multiple select.

Components are organised in a grid / table. You can **move selected components** around the panel by using the arrow buttons in the editor toolbar or the arrow keys on the keyboard. It is possible for components to span multiple rows or columns of the grid, using the buttons in the toolbar or holding `CTRL` while using the arrow keys.

To **bind a GUI component** to another *Praxis* component, use the `.binding` property and enter the address of the control you want to bind to. GUI components will attempt to configure themselves from the binding – eg. a slider with no minimum / maximum value overrides will use the minimum / maximum values of the bound control. When bound to a property control, GUI components will automatically sync.

Remove components by selecting them and using the `Delete` key or the delete button on the editor toolbar.

The GUI editor uses a layout manager called MigLayout (http://www.miglayout.com/). The `.layout` property of components is a MigLayout constraint string. For advanced usage, you can use other parameters defined here - http://www.migcalendar.com/miglayout/mavensite/docs/cheatsheet.html Do not remove the cell constraints!

A better GUI for setting layout constraints is in development.

MIDI Bindings Editor

MIDI `.pxr` files will open in the MIDI bindings editor. This is a simple editor that allows you to connect MIDI controllers to any *Praxis* control.

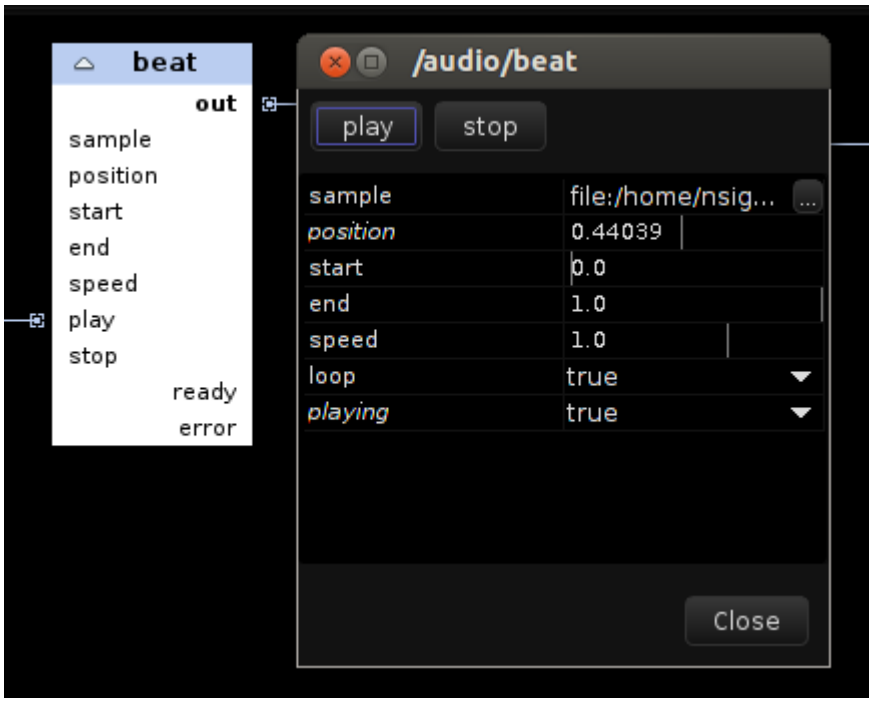
To **add a binding**, use the `Add` button in the editor toolbar.

Bindings are arranged in a table. You can **edit bindings** by double-clicking on them to open the component editor, or selecting them and using the `Properties` window. Some properties are also editable directly within the bindings table.

Unlike GUI components, MIDI bindings do not set up their minimum and maximum values automatically. By default they send a value from 0 to 1.

To **delete a binding**, select it and press `Delete` or use the delete button in the editor toolbar.

Component Editors / Properties



Component editor of an audio:sampleplayer in Praxis LIVE

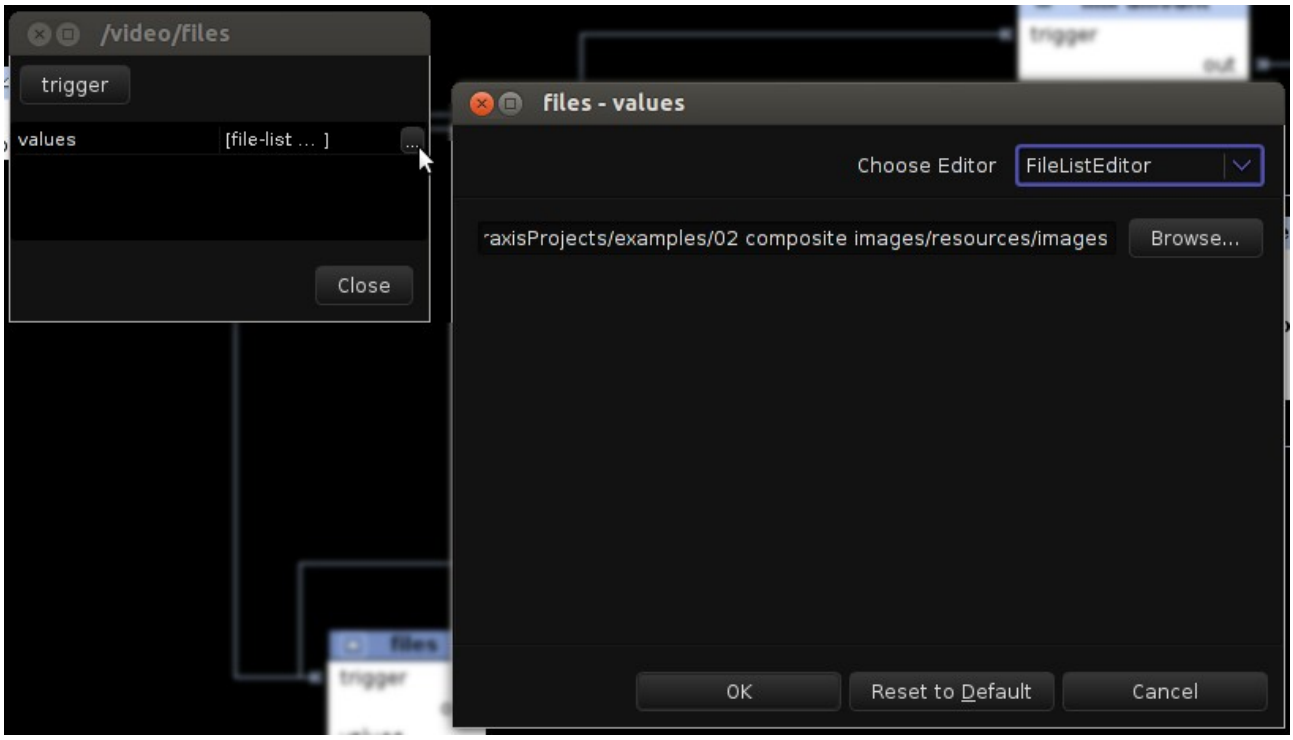
The above image shows the *Component Editor* window opened by double-clicking on the */audio/beat* component – an `audio:sampleplayer`.

At the top of the editor are buttons for action controls defined by the component – in this case for playing and stopping the player.

Below that is the property table. This is the same table that will show in the `Properties` window when you select the */audio/beat* component. You can change individual values by clicking on the value cell with the mouse and entering the new value. You can also use the arrow keys on the keyboard to move between different values – click `SPACE` to highlight and edit a value. Type the new value and press `ENTER` to change – if you change your mind you can also use `ESCAPE` to revert.

Numeric values that have a defined range (eg. `.speed`) also have a vertical line that represents the current value. You can also change these values by clicking and dragging the mouse within the value cell, just like you would use a slider. A short click still works to highlight the cell to edit by text.

Properties marked in italics (eg. `.playing`) are transient. This means that although you can control these properties, the value is not saved when you save the file. Transient is used for cases where it is unlikely you want to save the current value, maybe because it is continuously changing. In this case, should you want the sample-player to start playing immediately, you should create a `core:start-trigger` component and attach it to the `play` port.



Extended editors - File List editor

Some properties have the ability to open an extended editor – this is represented by the ellipsis button at the right of the value cell. Clicking on this button will open the extended editor window. Other properties will open the extended editor whenever you try and edit them.

Depending on the type of the property, there may be more than one extended editor available. A select box will appear at the top right of the extended editor window to allow you to choose which extended editor to use. Alternative extended editors usually reflect functions available within the *Praxis* scripting language. In the example above, the *File List Editor* is being used to provide the `.values` array of a `core:array:random` component. This editor works in the same way as the `file-list` function, providing a list of all the files in a directory. You might create an *Images* directory inside the resources of your project, and use this method to drive a random slide-show in your project (see example `02 composite images`).

Advanced Usage

Live Coding (Java / Processing)

The problem with higher-level patcher environments like *Praxis* is that they can't always provide the features you need. Therefore, *Praxis* makes it possible to define custom components using live-compiled Java code. Code can be edited and compiled as your project is running so that you can see / hear exactly what effect your changes are having.

Use the `core:code:custom` component to work with control signals in both audio and video patches.

Use the `video:code:composite` component to work with video. The API for custom video coding is based on Processing for ease of transition, although rewritten on top of *Praxis*' renderer.

Remember, you don't have to put all of your code into one component in a single, large 'sketch'. Consider breaking out different aspects into multiple components for ease of development.

Better editing and error reporting is in development. Currently, if you make an error in your code, it will revert to the previous value – make changes little and often!

See http://code.google.com/p/praxis/wiki/LiveCodeAPI for details of what functions are available.

Live Coding (GLSL)

When using the OpenGL rendering pipeline, you can live code custom fragment shaders in GLSL using the `video:opengl:filter` component.

The component provides 8 properties (u1-u8) which can be accessed within your fragment shader by using `uniform float u1`, etc. These uniform properties are always a value between 0 and 1.

There is minimal error reporting. If you make an error in your GLSL code, it is likely it just won't work. If you aren't using the OpenGL renderer (see `Tools - Options - Video`) then the filter will have no effect.

Praxis Script files

It is possible to create Praxis Script files (`.pxs`). These can be added manually to the Build or Run stages of your project.

Stand-alone Projects

It is possible to distribute *Praxis LIVE* projects as stand-alone executables. More information is online at http://code.google.com/p/praxis/wiki/StandAloneProjects

Make sure at least one root in your project has the `.exit-on-stop` property set to `true` or the framework will continue running in the background even if all windows have been closed!

CLI Player

The `praxis` player is shipped in the same directory as the launcher for *Praxis LIVE*. This can be used to launch a *Praxis LIVE* project from the command line. The player takes a single default argument – a directory that is a valid *Praxis LIVE* project or an individual script file (you can point it at the `.pxp` file in a project too).

Make sure at least one root in your project has the `.exit-on-stop` property set to `true`, or you run the command within an open terminal, otherwise you will not be able to stop the framework even if all windows have been closed.

Terminal

The `Terminal` window available in the `Tools` menu is *not* an OS terminal. It provides the ability to interact directly with the *Praxis* hub and components. You can execute short *Praxis* scripts directly.

For example, try running the `03 audio gui` example and open the terminal. Type `/audio/filter.frequency` into the terminal, and click `Run`. The filter component will respond with its frequency setting. Now try typing `/audio/filter.frequency 2000` and notice the sound change, and the control panel sync to the new value.

Contact / Support

Support is available through *Praxis'* Google Code site - http://code.google.com/p/praxis/ Here you will find links to the discussion group, issue tracker, Google+ page and Twitter feed.

You can also email praxis@neilcsmith.net