



Peppy User Manual

Version 0.16.0

Rob McMullen
robm@users.sourceforge.net

February 24, 2011

CONTENTS

1	Introduction	1
1.1	Target Audience	1
1.2	Cross Platform	1
1.3	GPL Licensed	2
2	Concepts	3
2.1	User Interface	3
2.2	Files as URLs	5
2.3	Network File Systems	5
2.4	Documents and Views	6
3	Up and Running with Peppy	7
3.1	Command Line Use	7
3.2	Single Peppy Process	8
3.3	Using the Basic Menu Functions	8
4	Customization and Preferences	9
4.1	Using the Preferences Dialog	9
4.2	Changing Text Styles	10
4.3	Configuration Files	11
4.4	Internationalization (i18n)	11
4.5	Key Bindings	11
5	Text Editing Basics *	15
5.1	Fundamental Mode	15
5.2	View Settings	16
5.3	Cut, Copy, and Paste	16
5.4	Undo and Redo	16
5.5	Find and Replace	16
5.6	Running Scripts	16
6	Macros *	17
6.1	Hierarchical Macros	17
6.2	Recording Macros	17
6.3	Playback of Macros	18
6.4	Macro Minor Mode	18
7	Projects	19
7.1	Peppy's Concept of Projects	19
7.2	Setting Up Projects	19

7.3	Source Code Control	20
7.4	Templates	20
7.5	Building	21
7.6	Running	21
7.7	CTAGS Support	21
8	Search Mode	23
8.1	The Search Window	23
8.2	Search Types	25
9	Work in Progress	29
10	Python Major Mode	31
10.1	Introducing Python Mode	31
10.2	Running Python Scripts	31
11	HexEdit Major Mode	33
11.1	The Editing Window	33
12	Hyperspectral Image Major Mode	35
12.1	What is Hyperspectral Imagery?	35
12.2	Hyperspectral Images in Peppy	36
12.3	Starting HSI Mode	36
12.4	Changing View Parameters	36
12.5	Using Profile Plots	36

INTRODUCTION

Peppy is an editor in the general sense – even though it aims to be an exemplary text editor, it allows you to edit lots of different types of data, not *just* text.

It is designed in the spirit of Emacs and its concept of “major modes” – editing modes that are tuned to edit a type of file using a specific user interface. Some major modes are very specific and only allow editing of a certain type of file, while others are general and can be used to edit different types of files.

Multiple views of the same document are permitted, and the views may be different major modes. A peppy window is organized into tabs, like the Mozilla Firefox browser, and unlike Integrated Development Environments (IDEs), multiple main windows can appear on the screen.

1.1 Target Audience

Peppy is targeted at software developers and people who edit source code for a living. It is also designed to be extensible: if you program in the Python language (and are so inclined), you can help improve the editor with new functionality that you’d like to see in the editor.

1.1.1 Editing Source Code

Source code editing is a major part of peppy. The goal is to provide a framework where new major modes can be added easily, and language support beyond simple syntax highlighting can be shared among similar languages. For example, there is an auto indenting class based on regular expressions that can be used to provide automatic indenting to many different languages simply by changing the regex. Many python based editors focus on editing python code and hard-code too many aspects of the editor based on that goal. Peppy strives to be general and easy to extend.

1.1.2 Other Editing Modes

The main difference between peppy and all the other python editors is the concept of the major mode. All other editors assume that you are going to be editing text and make no provision for a different type of user interface. Peppy provides many user interfaces depending on the type of file, and even provides major modes that can edit more than one type of file, like HexEdit major mode.

1.2 Cross Platform

The ability to have the same user interface on multiple platforms is also a goal of peppy. Thanks to the wxPython toolkit, this is possible on the major classes of platforms in existence today: unix/linux, Windows, and Mac OS X.

wxPython also provides a native look and feel, unlike Emacs which although can be compiled on all platforms forces its own look on the platform.

In addition, because it is free, peppy can be taken with you to any computer. Unlike proprietary editors like Wing IDE or TextMate that are either locked to one computer by a license key or being platform specific, you are free to use peppy wherever you want.

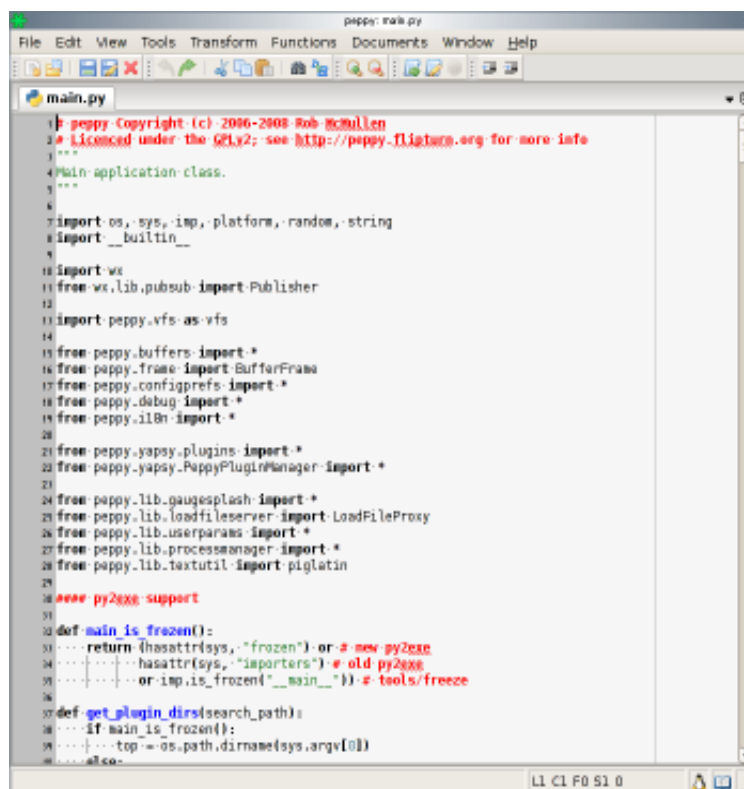
1.3 GPL Licensed

The main application code is licensed under the GPLv3, but peppy is actually a combination of many licenses. Most code is licensed under the GPLv2 or the wxPython license, but because one component of the code (the virtual file system) is under the GPLv3, the application as a whole must be licensed under the GPLv3. Parts of the code designed to be used as libraries are licensed under the wxPython license so that they may be freely used with other wxPython projects.

CONCEPTS

Peppy provides a fairly standard looking graphical user interface that should be familiar if you've used a text editor before. However, there are several unique features that may not be obvious, and these are described here.

2.1 User Interface



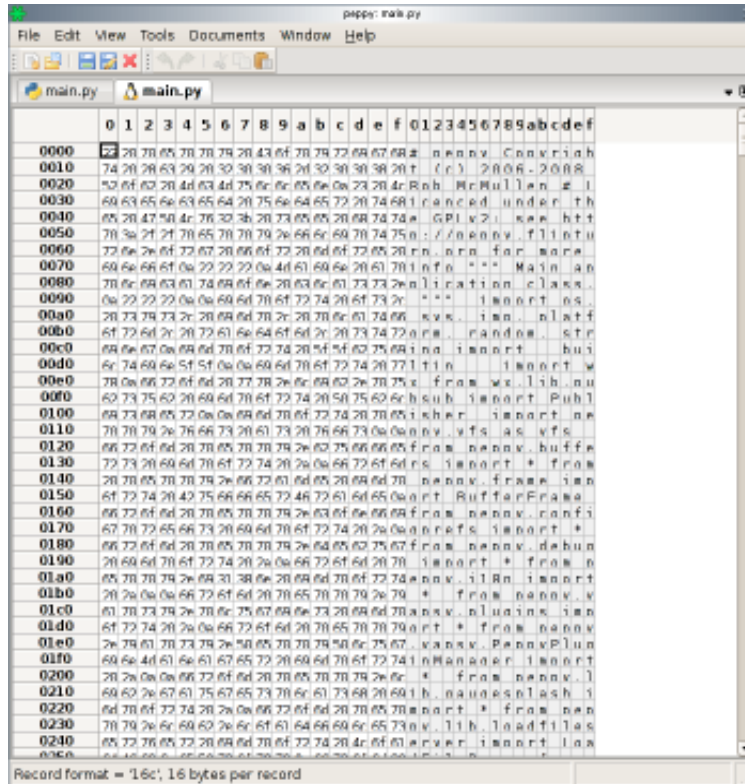
2.1.1 Top Level Windows

Peppy can display multiple top level windows, each of which can display an arbitrary number of tabs. Each tab contains exactly one major mode, which is to say that each tab contains one view of a document.

2.1.2 Major Modes

A major mode is a specific type of user interface window for a particular type of file. For example, the Python Major Mode in the image above shows a typical text editor window with line numbers on the left side, a cursor for typing, and scroll bars for moving back and forward in the document.

Other major modes like HexEdit:



provide a different type of user interface. Some types of major modes are specific to a type of file and some are more general and can be used to edit many types of files.

2.1.3 Menu Bar

The menu bar of peppy is dynamic, and is customized depending on the major mode. Switching major modes by changing tabs or loading a new file causes the menu bar to be modified so that it only displays items relevant to the current major mode. This prevents cluttering the user interface with a bunch of grayed out items that don't apply to what you're editing at the moment.

2.1.4 Tool Bar

The tool bar is likewise dynamic, and shows only those tool bar items that are appropriate to the current major mode. The tool bar may also be turned off if you don't like a tool bar or if you want a little extra vertical space for the major mode.

2.2 Files as URLs

All files in peppy are treated as being referenced by a URL, even local files. This abstraction makes it easy to add support for new URL schemes to load files, and for the most part, it makes no difference what scheme has been used to load a file.

Local files can be specified as simple pathnames, like `/Users/rob/file.txt`, or as full URLs like `file:///Users/rob/file.txt`. Note that URLs always use forward slashes even on windows. A windows path `C:\Program Files\peppy\peppy.exe` is equivalent to the URL `file://C:/Program Files/peppy/peppy.exe`

2.2.1 Automatic Recognition of File Type

Another unique aspect of peppy is the lengths to which it goes to identify a file. Because most text editors assume that the file that you're loading is a text file, they don't spend much time trying to figure out what type of file it really is. They just look at the file extension and assume that it correctly identifies the text within it.

Peppy does take into account the filename and extension when identifying a file, but it doesn't *just* do that – it also provides several hooks in the file loading process to examine the URL or the contents of the file to determine what type of file it is. This set of heuristics allows peppy to correctly determine the major mode to use even if the file is incorrectly labeled, or in cases where the same file extension is used for different types of data.

2.3 Network File Systems

Peppy uses the virtual file system from `itools` to provide the framework to support networked file loading. It provides the means to load files based on the *protocol* (also called the *scheme*) of the URL. For example, the protocol of the URL `http://peppy.flipturn.org` is `http`, and the protocol of `file://C:/Program Files/peppy/peppy.exe` is `file`.

2.3.1 HTTP

Read-only support is provided for files using the `http` protocol, so any file that is visible to a normal web browser can be loaded by peppy. Obviously, due to the read-only nature of normal `http` servers, you will have to save the file using some other protocol.

2.3.2 WebDAV

The 0.13.0 release added experimental support for the `WebDAV` protocol, which is a distributed filesystem based on web servers.

This is an *experimental* addition to peppy, and a work in progress. I have tested it quite a bit, but this is the first networked filesystem that peppy supports for both reading and writing, and there may still be issues to resolve.

Also note that the current implementation of WebDAV will lock the GUI until the operation completes, so if a WebDAV server freezes in the middle of a transfer, you're stuck. Multithreaded operation of the networked file systems is planned, the goal being to provide an opportunity to cancel an operation if it is taking too long.

WebDAV files and directories are accessed using URLs like `webdav://www.webdavserver.com/path/to/file`, where peppy will prompt you for authentication information and remember the authentication for the duration of your editing session. Optionally, a less secure method is also supported where you embed the authentication information directly into the URL itself, like: `webdav://user:pass@www.webdavserver.com/path/to/file`

2.3.3 SFTP

The 0.14.1 release added experimental support for the [SSH File Transfer Protocol \(SFTP\)](#), which is a method to access a remote filesystem using the SSH protocol to provide encryption and security.

This is currently an *experimental* addition to peppy, but is expected to be a stable feature in the next major release of peppy.

SFTP files and directories are accessed using URLs like `sftp://some.server.com/path/to/file`, where peppy will prompt you for authentication information and remember the authentication for the duration of your editing session. A username may also be specified, like `sftp://username@some.server.com/path/to/file`, in which case peppy will prompt you for the password by popping up a dialog box. Also supported, but not recommended, is to include the password in the URL like `sftp://username:passwd@some.server.com/path/to/file`, but this will result in the storage of the plain text password in history files and other places. Definitely not recommended.

2.3.4 Other Network File Systems

Support for other network protocols may be added in the future. Under consideration are the old insecure FTP protocol, as well as the Files transferred over Shell (FISH) protocol.

2.3.5 Special File Systems

There are also some built-in schemes, like **about:** that used for read only documentation, **mem:** used for an in-memory temporary file system, **tar:** used for read only access to files contained within tar files, and more esoteric schemes like **aptus:** which is used in the fractal renderer.

2.4 Documents and Views

A URL uniquely identifies a file on some file system, and peppy uses the URL as the identifier of a loaded document. Only one copy of a document exists in peppy, but it can have many different views in the user interface. And, even if no more views exist of the document, it is still kept in memory by peppy until you explicitly delete it from memory.

Opened files appear in the *Documents* menu, and a particular document can be opened in any peppy window by selecting it from the menu. A new tab containing a view of the document will appear, using its default major mode. Deleting the tab only causes the tab to go away; it doesn't delete the document. Only when closing the document will the document be removed from memory.

UP AND RUNNING WITH PEPPY

Once peppy is installed, starting the application depends how you installed the code.

Installed using easy_install: (RECOMMENDED) If you used `easy_install` to install the latest version from the Python Package Index, a python script named `peppy` will be installed in the system binary directory, so simply typing `peppy` on the command line will start the program.

Downloaded from source: If you downloaded `peppy` but did not install the source, you run the application with the command line `python peppy.py` from within the `peppy` directory.

Installed from the downloaded source: If you ran `python setup.py install` on the downloaded source, a python script named `peppy` will be installed in the system binary directory.

Installed a binary distribution: This is only applicable to Windows and Mac OS X. The install process will leave an icon somewhere, and you can just click on that.

3.1 Command Line Use

When you start `peppy` from the command line, you can specify options and URLs to load. For example, running from a source distribution, you might use a command line like this:

```
python peppy.py README /tmp/file.txt http://www.flipturn.org/index.html
```

which tells `peppy` to open three URLs: the local file `README` in the current directory, the file `/tmp/file.txt`, and the `index.html` file from `http://www.flipturn.org`.

3.1.1 Options

There are many command line options available, the most useful of which are outlined below. You can see the full list by running `peppy` with the `--help` argument.

- d:** Send all debug printing to the console
- t:** Run in test mode – allow multiple `peppy` processes (see the next section) and send all debug printing to the console
- v:** Run in verbose mode, which turns on **a lot** of debugging output. Use in combination with `-d` or `-t` to send the output to the console

3.2 Single Peppy Process

Normally, peppy only allows a single process per user so that subsequent file open requests (by starting peppy from the command line with an argument) will actually be opened by the already running process. This can be disabled by preference or by the `-t` option as described above.

3.3 Using the Basic Menu Functions

The GUI should be similar to typical windowed applications on your platform. The layout of the menu attempts to follow the [Apple style guidelines for the menu bar](#) even though peppy works with all platforms and not just Mac OS X. I found their guidelines useful and logical, and so applied them to my work.

There are seven menu titles that will always appear in the menu bar, and additional titles depending on the major mode. The seven are: File, Edit, View, Tools, Documents, Window, and Help.

The File menu contains the common file load, file save, and related options. The Edit menu holds commands related to editing and selecting text or data, and also is where you'll find the [preferences](#). View holds menu items that affect how the major mode displays its contents, but nothing that actually alters the data – just how the data is viewed. The Tools menu holds standalone commands that either start a process or don't change the contents of the document. The Documents menu contains a list of currently open documents, but note that some documents may not have an active view. Window holds a list of top level peppy windows, allowing you to switch back and forth between them. And finally, Help contains menu items related to documentation.

3.3.1 Opening Files

Peppy provides several commands to open files, from the traditional GUI file dialog using the File -> Open -> Open File menu command, to the more emacs- like File -> Open -> Open File using Minibuffer command (bound to C-x C-f in emacs keybinding mode).

3.3.2 Editing Files

Once a file has been opened, a new tab will appear in the window that shows the GUI that is used to edit that type of file. There are specific types of editing modes, called [major modes](#), for different types of files. For instance, plain text files are editing using the Fundamental major mode, while python source files are edited using the Python major mode. Both these major modes are similar in that they use a text editing component (called the StyledTextCtrl in wxPython, which is based on the [Scintilla](#) source code editing component). However, unlike most editors, text files are not the only thing that can be edited. There are major modes for editing [binary files](#), and even [hyperspectral images](#).

3.3.3 Saving Files

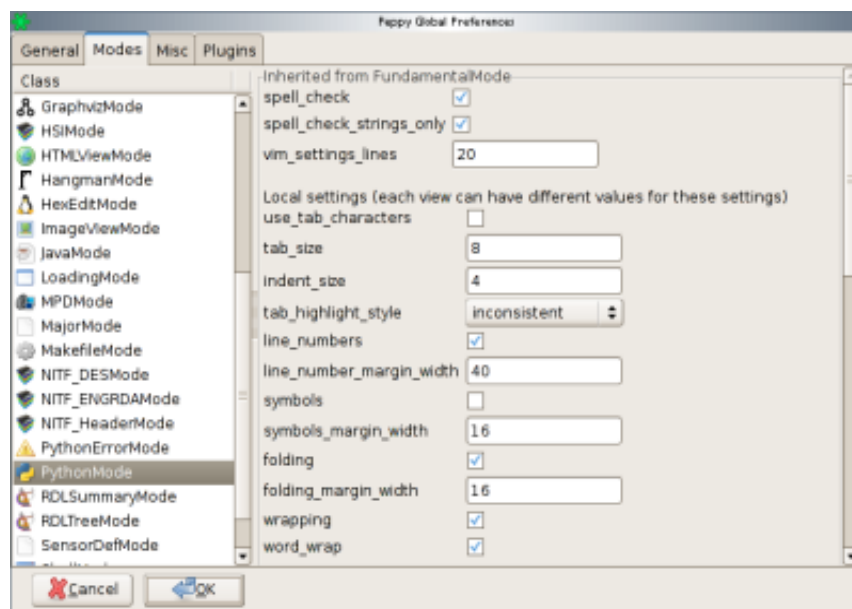
After editing the file, it must be saved before the changes can be made permanent. Like opening files, there are several ways to save the file. You can use the File -> Save to save the file if you want to overwrite your changes, or File -> Save As to pull up a traditional file save dialog to save it to a new file. Some major modes provide other ways to save the file in the File -> Export menu.

CUSTOMIZATION AND PREFERENCES

There are two ways to change preferences in peppy. First is to use the Edit -> Preferences menu (or Peppy -> Preferences menu if you're on Mac OS X). Most settings are available here, although there are a few that can't be set through the GUI. Those settings that can't must be modified by hand in the configuration directory.

4.1 Using the Preferences Dialog

Most of the common configuration options are available through the preferences dialog.

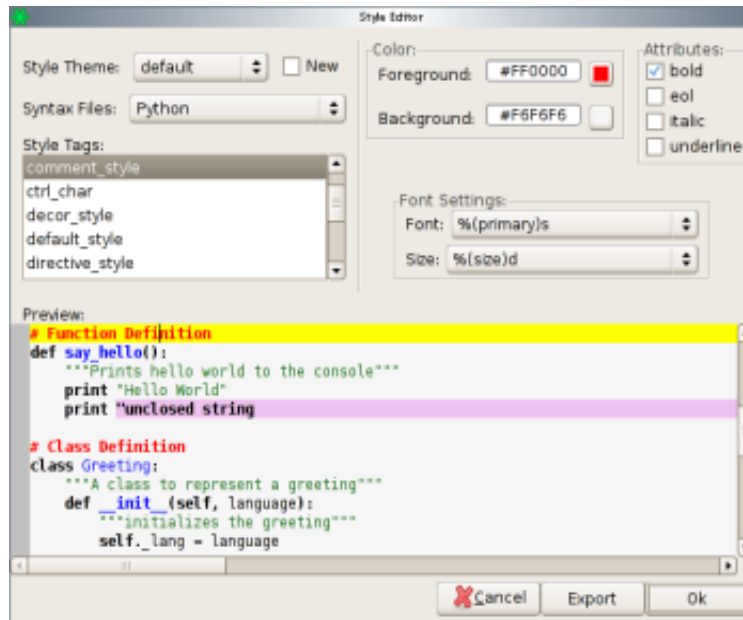


The settings are grouped into four different tabs: General, Modes, Misc, and Plugins, where General settings apply to global or application level configuration objects, Modes apply to the various major modes, Misc contains a bunch of stuff that didn't have any other tab to be placed, and Plugins unsurprisingly contains options for plugin objects. Clicking any item on in the left hand list will bring up the specific settings for that item.

Note that any changes to settings are only only applied after clicking OK to the dialog; they are not applied immediately. Note that you may change tabs at any time before hitting OK; your settings aren't lost when investigating settings of other items.

4.2 Changing Text Styles

Font colors and styles are changed using the Edit -> Text Styles dialog. This associates particular syntax elements of the text file with styles and colors specified here. If you're familiar with Editra, you'll recognize the text style dialog:

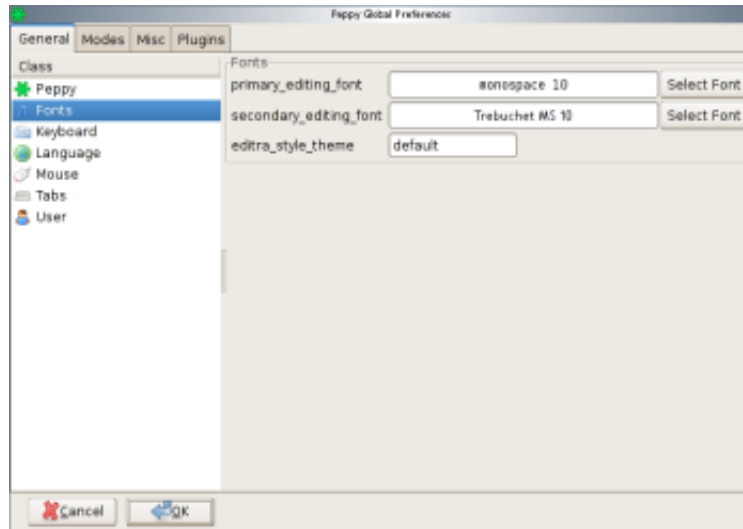


Peppy uses the same styling system as Editra, and indeed can use the same style sheets as created by Editra when placed in the proper configuration directory of peppy. (See below for more information on configuration files.)

The convenient characteristic of the Editra styling system is syntax elements are styled using the same font/color combination regardless of language. So, for example, comments in Python will appear the same as comments in C++. This leads to a much quicker setup time when configuring peppy to display syntax elements in the manner that you like.

4.2.1 Changing Fonts

The only text styling item not set through the text style dialog is the specific font used to display the text. The wxPython StyledTextCtrl uses a single font for the main text, and can optionally use a separate font for the line numbers. These fonts are set in the Preferences dialog: in the General tab:



choose the Fonts item and select the primary editing font (for the main text) and the secondary editing font (for the line numbers).

4.3 Configuration Files

Peppy maintains a configuration directory on a per-user basis. Where it is located depends on which platform you are using: (if you're familiar with wxWindows, it's the directory that's returned from the `wx.StandardPaths.GetUserDataDir()` method)

- unix/linux: `$HOME/.peppy`
- windows: `C:/Documents and Settings/username/Application Data/Peppy`
- mac: `~/Library/Application Support/Peppy`

There are a variety of files in that directory, used to store various bits of data for persistence between runs of peppy. The user configuration file is `peppy.cfg`, and the settings saved by the application go in `preferences.cfg`. Don't use `preferences.cfg` for your user settings, as they are overwritten when settings are saved after exiting the main application. Put all your hand-edited changes in `peppy.cfg` instead.

The preferences are stored in [Windows INI](#) format.

4.4 Internationalization (i18n)

Several national languages are available thanks to the members of Launchpad who have been kind enough to translate my English into other languages. In the preferences dialog, click on the General tab and choose the Languages item. You can select from any of the languages in the list, and the user interface will change immediately. No need to restart the application!

4.5 Key Bindings

There are three styles of key binding supported by peppy: windows style, Mac style, and emacs style. These are controlled by preference settings in the Preferences menu, in the General tab under the Keyboard item. This setting controls the default key bindings for all actions; actions may still be overridden manually.

4.5.1 Configuring Key Bindings

User specified key bindings should be stored in the `peppy.cfg` file in the user's preferences directory.

In the `peppy.cfg` file, the section `KeyboardConf` controls user overrides to the default keybindings.

An example of the section might be:

```
[KeyboardConf]
key_bindings = "emacs"
NextWord = C-N
PreviousWord = C-P
Paste = C-V
```

Breaking it down by parts:

[KeyboardConf] This is the section header that denotes a keyboard configuration block

key_bindings = "emacs" The `key_bindings` variable controls the default keyboard configuration. If this variable is not present, the default configuration is automatically determined based on the platform. To set this variable, it should use one of the following values: "emacs", "win", or "mac".

NextWord = C-n The remaining entries in the section are keybindings. The format of each line is `ActionName = keybinding`, where `ActionName` is the class name of the action you wish to rebind. Currently, the easiest way to discover these name is to run `peppy` with the `--show-key-bindings` argument, which will display all of the current keybindings to the console in a format usable for the `KeyboardConf` section of the preferences. You can also use the menu item `Help -> Show Key Bindings`, or when all else fails you can look at the source code. (A GUI to change the keybindings is ticket #53 on the roadmap. Feel free to add suggestions there.)

To specify keystrokes, consult the section below.

4.5.2 Describing Keystrokes

To modify the key bindings, Key bindings are specified using a special shorthand showing the modifiers of the key and the key itself. Modifiers are:

- **S** - Shift
- **C** - Control on win/unix, Command on mac
- **M** - Meta or Alt on win/unix, Option on mac

Keys are then specified using a string listing the modifier characters separated by the `-` character, and then the unshifted character itself. Unmodified keys are also possible – if the keystroke doesn't contain a `-` character it is used as a literal character. For example:

- **x** - the letter X
- **s** - the letter S
- **S-s** - shift S
- **C-a** - control A
- **S-/** - shift slash (note that you can't specify a `?` directly – keys are recognized by their unshifted state in `wx`)
- **M-C-q** - alt control Q

Multiple keystrokes are separated by spaces:

- **C-x C-s** - control X followed by control S

- **C-x 5 2** - control X followed by the number 5 followed by the number 2

Special characters are given by their text equivalent:

- **TAB** - the tab key
- **S-UP** - shift up arrow

Here's a list of special characters (note that if you're familiar with wxPython, this is really just the *WXK_* name with the *WXK_* prefix removed):

```
BACK TAB RETURN ESCAPE SPACE DELETE START LBUTTON RBUTTON CANCEL MBUTTON
CLEAR PAUSE CAPITAL PRIOR NEXT END HOME LEFT UP RIGHT DOWN SELECT
PRINT EXECUTE SNAPSHOT INSERT HELP NUMPAD0 NUMPAD1 NUMPAD2 NUMPAD3
NUMPAD4 NUMPAD5 NUMPAD6 NUMPAD7 NUMPAD8 NUMPAD9 MULTIPLY ADD SEPARA-
TOR SUBTRACT DECIMAL DIVIDE F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15 F16
F17 F18 F19 F20 F21 F22 F23 F24 NUMLOCK SCROLL PAGEUP PAGEDOWN NUMPAD_SPACE
NUMPAD_TAB NUMPAD_ENTER NUMPAD_F1 NUMPAD_F2 NUMPAD_F3 NUMPAD_F4
NUMPAD_HOME NUMPAD_LEFT NUMPAD_UP NUMPAD_RIGHT NUMPAD_DOWN
NUMPAD_PRIOR NUMPAD_PAGEUP NUMPAD_NEXT NUMPAD_PAGEDOWN
NUMPAD_END NUMPAD_BEGIN NUMPAD_INSERT NUMPAD_DELETE NUMPAD_EQUAL
NUMPAD_MULTIPLY NUMPAD_ADD NUMPAD_SEPARATOR NUMPAD_SUBTRACT
NUMPAD_DECIMAL NUMPAD_DIVIDE
```


TEXT EDITING BASICS *

5.1 Fundamental Mode

Fundamental mode is the basis for all source code editing modes in peppy. All of the text editing functions that are common to more than one mode are generally placed in Fundamental mode. If you're familiar with object oriented programming concepts, Fundamental mode is the *superclass* of the other text editing modes.

5.1.1 Hierarchies of Major Modes

There are different hierarchies of major modes in peppy, but the primary hierarchical tree starts from Fundamental mode. Major modes that are used to edit text files descend from Fundamental mode, and therefore are part of this hierarchy.

Hierarchies are important because many of the commands available in Peppy are based on the major mode of the current file. Actions are only presented to you in the menubar and toolbar if they're appropriate to the major mode. For instance, when editing text files, it doesn't make sense to include actions used to edit images.

The major mode hierarchy starting with Fundamental mode includes Python mode, C++ mode, Bash mode, and all the other modes used to edit source code. Fundamental mode is the more general major mode, so all actions available for Fundamental mode are available to Python mode, for instance. However, Python mode actions are not available to Fundamental mode.

5.1.2 Scintilla and the Styled Text Control

Peppy uses the wxPython component called the StyledTextCtrl for its user interface, which is based on the source code editing component called Scintilla. Scintilla and the StyledTextCtrl supply the highlighting, syntax coloring, code folding, line numbers, word wrapping, and many more built-in features. Additionally, due to the cross-platform nature of both wxPython and Scintilla, peppy operates in very similarly across the three major classes of operating systems supported by wxPython: unix-like systems, Windows, and Mac OS X.

5.2 View Settings

5.3 Cut, Copy, and Paste

5.3.1 Middle Mouse Paste

5.4 Undo and Redo

5.5 Find and Replace

5.5.1 Wildcard / Shell Style

5.5.2 Regular Expressions

5.6 Running Scripts

5.6.1 Filters

MACROS *

Macros are a set of actions that are recorded for later playback. They are used as a convenient shortcut to perform repetitive tasks.

6.1 Hierarchical Macros

Macros are associated with major modes, and because Peppy uses a hierarchical system of major modes (see *Hierarchies of Major Modes*), macros are displayed only if they are appropriate for the current major mode.

Note that macros for more general major modes will be available. For example, when editing python source files using Python mode, Fundamental mode macros will also be available. However when editing plain text files using Fundamental mode, you will not be able to use Python mode macros. This keeps the user interface much less cluttered.

6.2 Recording Macros

Start recording a macro using the Tools -> Macros -> Start Recording menu item, or the equivalent keystroke. For windows keybindings, the default keystroke is Ctrl+Shift+9. For emacs, the standard emacs binding of 'C-x (' is available.

Once recording, every action that makes a direct change to the contents of the document will be recorded. For example, characters will be recorded as you type, and keystrokes that have special meaning (like using TAB to reindent the line or RETURN to end the current line and autoindent the next line) will be recorded such that on playback they will apply their function.

Actions will be recorded until you stop recording using the Tools -> Macros -> Stop Recording menu item (or its equivalent keystroke: Ctrl+Shift+0 or 'C-x)' with emacs keybindings).

Changing tabs will abort macro recording.

By default, macros are saved using a name derived from the characters that you typed. Macros can be renamed to give them more meaning to you; see the section *Macro Minor Mode* below.

You can also bind a macro to a keystroke. After finishing the macro recording, use the Tools -> Macros -> Add Keybinding for Last Macro menu item. You'll be prompted to enter a sequence of keystrokes followed by the RETURN character that ends the sequence. The keystrokes will then be bound to that macro so whenever you are in the current major mode, that sequence of keys will automatically trigger the macro.

Macros in Peppy are automatically saved so they'll be available the next time you use the program.

6.3 Playback of Macros

In addition to macros being triggered by the keystroke set using the Add Keybinding for Last Macro menu item, macros can be played back by last macro defined, by name, or by selecting the macro from a list.

Tools -> Macros -> Play Last Macro will replay the last macro recorded, assuming the macro is valid for the current major mode. If not, nothing happens.

Tools -> Macros -> Execute Macro brings up a minibuffer that allows you to use tab completion to type in the name of the macro you'd like to replay.

Macros can also be executed by selecting them from a list: see the section below for more information.

6.4 Macro Minor Mode

In the major mode sidebar on the right hand side of the text, you'll see a springtab named "Macros". This is the Macro Minor Mode – it contains the user interface for managing the macros that work with the current major mode.

Clicking on the Macros button will pop out the window that contains the hierarchy of major modes applicable to the current major mode and the macros defined for each of those major modes:

[pic here of macro minor mode]

Right clicking on a macro will bring up a context menu showing various options, like Edit, Rename, Delete, and an option to provide a new keybinding.

6.4.1 Editing Macros

Macros are stored as snippets of Python code, and they can be edited to provide even more functionality than is available by simply recording keystrokes. This is a bit of an advanced topic, so some care is needed to make sure that the macro will function properly if you add custom python code to the macro.

Regardless of the major mode for which the macro was recorded, macros are Python code – therefore the normal Python mode is used to make changes. For example, even if the macro is for C++ mode, you edit the macro itself using Python mode.

You must save your changes by selecting File -> Save in order for the updated macro to take effect.

PROJECTS

7.1 Peppy's Concept of Projects

Peppy uses a very simple definition of projects based on directories. Any directory can be considered as the top level of a project, including directories residing in other projects. Peppy determines membership in a project by starting at the source file and marching upwards through the path until it finds a special marker that indicates the directory is a project. By default, this marker is a special directory at the root of the project directory called `‘.peppy-project’`.

Features available in projects include source code control operations like viewing the status of files and committing changes, template files used to generate boilerplate text for newly created files, and menu and toolbar items to control the build process and to run the application.

7.2 Setting Up Projects

You can either create a new project from within peppy, or tell peppy about an existing project.

7.2.1 New Projects

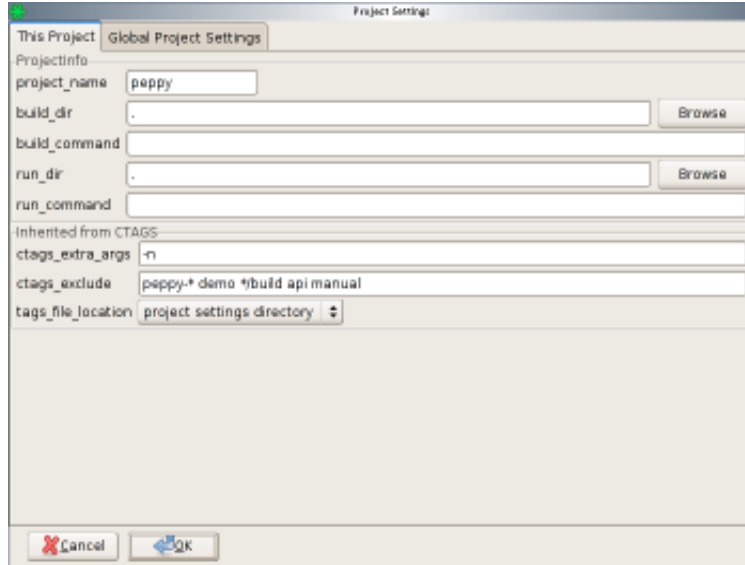
Choose the File -> New -> Project... menu item to create a new project directory. Use this when starting a project from scratch.

A dialog box will appear allowing you to choose where to place the new directory, and after selecting that will show the project configuration dialog allowing you to set some project specific options.

7.2.2 Existing Projects

If you have an existing project in a directory, you can tell peppy about it by using the File -> New -> Project From Existing Code... menu item. This also prompts for a directory, but this time it is looking for you to enter an existing directory. After entering the directory, it continues on to the project configuration dialog.

7.2.3 Project Configuration Dialog



This dialog contains two tabs, one showing the configuration information for this project and the other containing global settings that apply to all projects.

7.3 Source Code Control

Most projects will be using some form of source code management system to keep track of changes. (If your project doesn't use source control, you should consider it strongly!) Peppy currently supports Git, SVN, Bzr, and CVS systems.

Peppy displays some source control information in the Projects springtab, showing the current status of each file in a project.

7.4 Templates

Templates are boilerplate code that will be inserted when creating a new file of a specific type. There are two classes of templates, global templates that don't depend on a project, and project templates that apply only to the specific project that you're using. Global templates are stored in the peppy configuration directory, while project templates are stored within the project directory.

Note that project templates override global templates, so creating a file within a project directory will use the project template instead of the global one.

7.4.1 Creating Templates

Templates are associated with major modes. Creating a template begins with deciding what major mode to use. After creating some source code in that major mode, you can either save it as a global template or as a project template using the Project -> Templates menu item. Note that the template will only be associated with that major mode and not subclasses of the major mode. So, for instance, a template for FundamentalMode will not be used when creating a new PythonMode file.

7.5 Building

A build command can be associated with the project using the project configuration dialog. Only after entering a command with the Project -> Build... menu item (and associated toolbar icon) be active.

You can specify the build directory using the project settings dialog. This directory can be an absolute directory, or can be relative to the project directory.

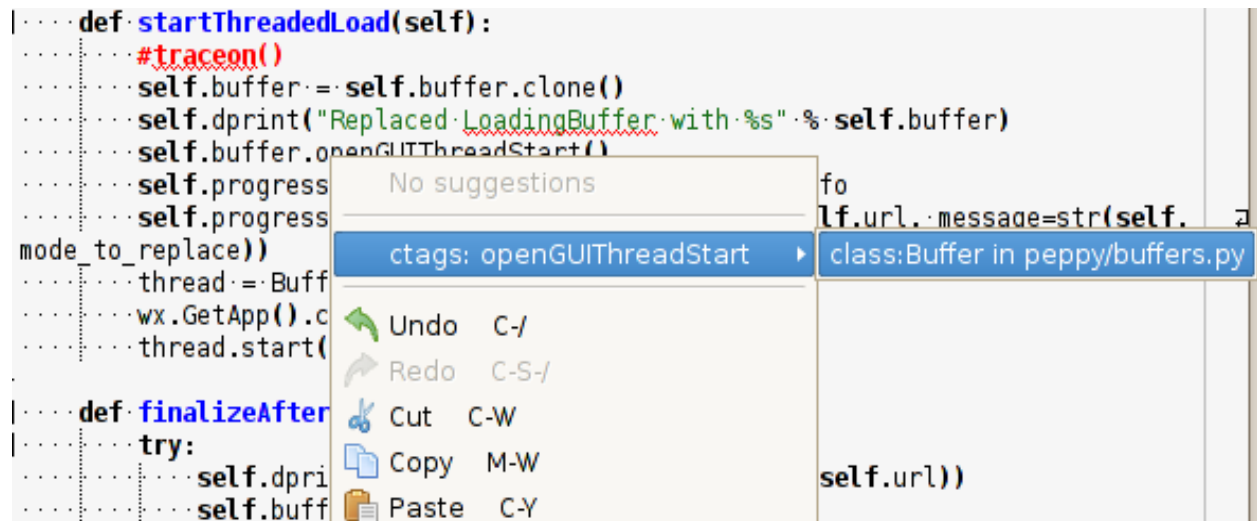
In general, this only applies to projects using compiled languages like C, C++, or java. Any messages output by the build process will be displayed in an output sidebar.

7.6 Running

You can also associate an executable with the project that will allow you to test the program using the Project -> Run... menu item (and associated toolbar icon). As with the build process, you can specify the directory in which the executable will be run, and any command line arguments.

7.7 CTAGS Support

Peppy uses the [Exuberant CTAGS](#) program to generate cross references among the source files within the project. Installing this program will provide a context menu entry when you right click on a syntax element in the source code:



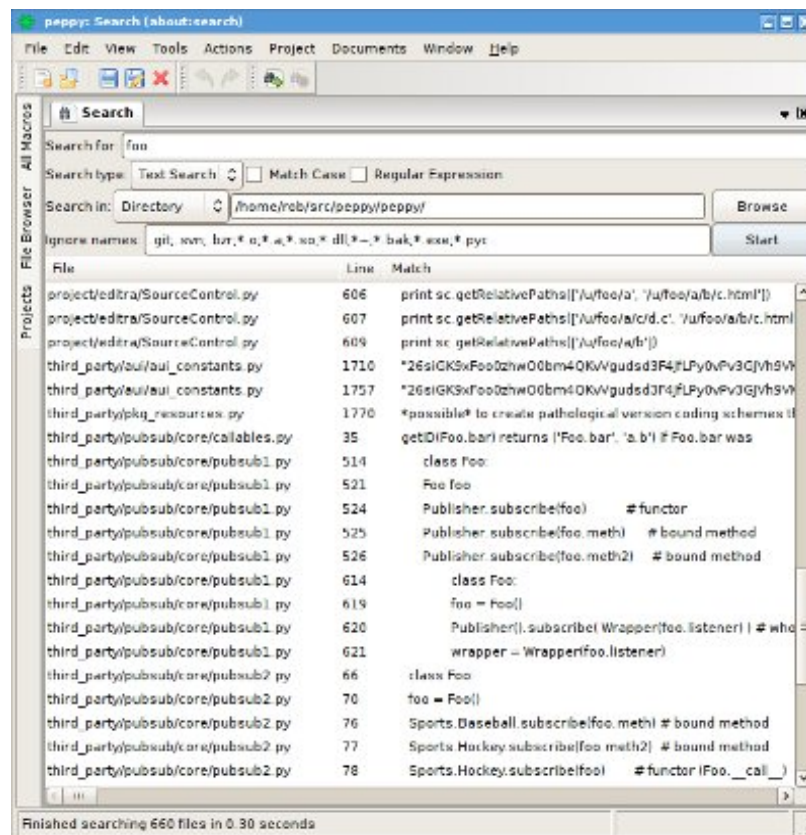
Selecting the cross reference will cause peppy to load the referenced file (or switch tabs to the file if it's already loaded) and jump to the line number of the reference.

Currently, the tags file must be kept up to date manually. Select the Project -> Rebuild Tag File menu option to recreate it.

SEARCH MODE

Search Mode scans through a set of files looking for matches. It is similar to the ‘grep’ command in unix, but is much more powerful because it can be extended to search for matches based on context or based on characteristics in certain types of files.

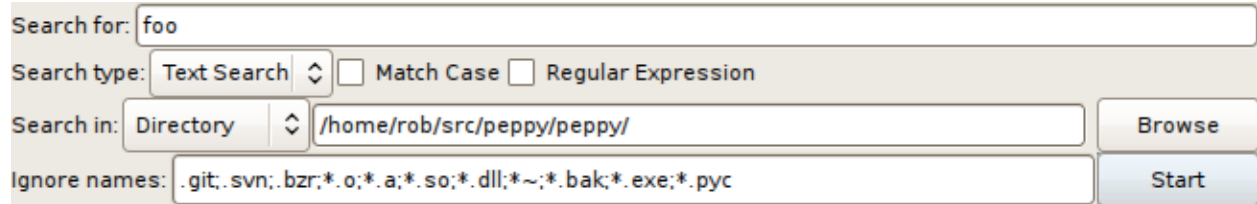
8.1 The Search Window



The main search mode window is split into two sections, the top is for entering search criteria, and the bottom is a list that displays the results of the search.

8.1.1 Search Criteria

The top section of the search mode window contains the search criteria, defining what you are searching for and which files to scan. There are four sections to the search criteria: the search string, the type of search, how to choose the files to search, and the files to ignore.

The image shows a graphical user interface for search criteria. It consists of four main sections: 1. 'Search for:' with a text input field containing 'foo'. 2. 'Search type:' with a dropdown menu set to 'Text Search', and two checkboxes for 'Match Case' and 'Regular Expression'. 3. 'Search in:' with a dropdown menu set to 'Directory', a text input field containing '/home/rob/src/peppy/peppy/', and a 'Browse' button. 4. 'Ignore names:' with a text input field containing '.git;.svn;.bzip;*.*.o;*.a;*.so;*.dll;*.~;*.bak;*.exe;*.pyc' and a 'Start' button.

The first line is the search string, which in the simplest case is the exact string that you wish to find. In more complex cases, it can be a regular expression, a comparison, or another format depending on the type of search.

The type of search is what makes Peppy's search so powerful. The type is selected by a pull-down list, and additional types can be added through plugins. The default search types are **Text Search** which provides the usual text search operations (whether or not the case is significant, and regular expressions for advanced pattern matching), and **Numeric Search** which scans through text files and compares numbers that it finds with the comparison defined by the search string. The search types are described in more detail below.

Files can be chosen in a number of ways by selecting from a pull-down list. (Note this can also be extended by plugins.) Currently, the available options are **Directory**, **Project**, and **Open Documents**. The **Directory** option will perform its search starting with all files in the named directory, and continue recursively through all subdirectories. The **Project** option will limit itself to only those files contained in the selected project (see *Peppy's Concept of Projects* for more information and how to define projects). The **Open Documents** option limits the search to only those documents that are already open in peppy; that is, only those documents that appear in the **Documents** menu.

Finally, the **Ignore Names** criteria is a list of filename extensions to skip in the search process.

Pressing the **Start** button (or using either the **Actions -> Start Search** menu item or the start search icon on the toolbar) will begin the search process. An error that causes the search to fail will show up in the status bar, otherwise the search will display results as they are discovered. Note that you can continue to work in other tabs or peppy windows as the search operates because the search is performed in a separate thread.

8.1.2 Search Results

The bottom section of the search mode window is a list that will be populated with the results of the search. The list is broken up into three columns: the filename of the match, the line number, and the matching line. Each line in the list represents a matching line in a file.

Search for:

Search type: ☒ Numeric Search ☐ Hexadecimal

Search in:

Ignore names:

File	Line	Match
samples/validate.py	1087	VdtValueTooLongError: the value "1234" is too long.
samples/validate.py	1392	>>> v.get_default_value(u'string(min=4, default="1234")')
samples/validate.py	1393	u'1234'
samples/validate.py	1394	>>> v.check(u'string(min=4, default="1234")', u'test')
samples/write-protected.txt	7	remote 192.168.190.254 1194
samples/wxpython-demo/run.py	9	# RCS-ID: \$Id: run.py 53286 2008-04-21 15:33:51Z RD \$
samples/wxpython-demo/run.py	10	# Copyright: (c) 2000 by Total Control Software
test_bufferedreader.py	12	self.s = '0123456789'

The first column contains the filename of the match. Common directory prefixes will be removed to save space if searching in a directory or project where all the matches have a common directory prefix.

The second column contains the line number of the match. Line numbers start from 1. Note that multiple matches on the same line will only be shown once in the list.

The third column contains the matching line. In future versions, multiple matches in the line will be highlighted, but at the moment only the line itself is displayed.

8.2 Search Types

One of the differences between peppy and other search programs is the search type. Using the search type, text can be matched based on criteria, not just matching characters.

Currently, there are two search types distributed with peppy: **Text Search** and **Numeric Search**.

8.2.1 Text Search

Text search is much like the basic search capabilities of normal editors. You can search with or without case sensitivity, and you can choose to use regular expressions for advanced searching.

By default, text is matched without regard to case. If you would like case to be significant in the match, check the **Match Case** checkbox.

Regular expressions are available by checking the **Regular Expression** checkbox. The search string is then taken as a regular expression (or *regex*) and lines in files will be matched against the regex. Regular expressions, if you haven't used them much, can be complicated. From the [Python](#) documentation, here's a short summary of regular expressions:

Regular expressions can contain both special and ordinary characters. Most ordinary characters, like **A**, **a**, or **0**, are the simplest regular expressions; they simply match themselves. You can concatenate ordinary characters, so **last** matches the string 'last'. (In the rest of this section, we'll write regexes in this bold style, and strings to be matched 'in single quotes'.)

Some characters, like **|** or **(**, are special. Special characters either stand for classes of ordinary characters, or affect how the regular expressions around them are interpreted.

Some of the special characters include:

- (Dot.) In the default mode, this matches any character except a newline. If the DOTALL flag has been specified, this matches any character including a newline.
- ^ (Caret.) Matches the start of the string.
- \$ Matches the end of the string or just before the newline at the end of the string. **foo** matches both 'foo' and 'foobar', while the regular expression **foo\$** matches only 'foo'.
- * Causes the resulting RE to match 0 or more repetitions of the preceding RE, as many repetitions as are possible. **ab*** will match 'a', 'ab', or 'a' followed by any number of 'b's.
- + Causes the resulting RE to match 1 or more repetitions of the preceding RE. **ab+** will match 'a' followed by any non-zero number of 'b's; it will not match just 'a'.
- ? Causes the resulting RE to match 0 or 1 repetitions of the preceding RE. **ab?** will match either 'a' or 'ab'.
- \ Either escapes special characters (permitting you to match characters like "\", "?", and so forth), or signals a special sequence; special sequences are discussed below.
- [] Used to indicate a set of characters. Characters can be listed individually, or a range of characters can be indicated by giving two characters and separating them by a "-". Special characters are not active inside sets. For example, **[akm\$]** will match any of the characters "a", "k", "m", or "\$"; **[a-z]** will match any lowercase letter, and **[a-zA-Z0-9]** matches any letter or digit. Character classes such as **\w** or **\S** (defined [here](#)) are also acceptable inside a range. If you want to include a "]" or a "-" inside a set, precede it with a backslash, or place it as the first character. The pattern **[]** will match ']', for example.

You can match the characters not within a range by complementing the set. This is indicated by including a "^" as the first character of the set; "^" elsewhere will simply match the "^" character. For example, **[^5]** will match any character except "5", and **[^^]** will match any character except "^".
- | A|B, where A and B can be arbitrary REs, creates a regular expression that will match either A or B. An arbitrary number of REs can be separated by the "|" in this way. This can be used inside groups (see below) as well. As the target string is scanned, REs separated by "|" are tried from left to right. When one pattern completely matches, that branch is accepted. This means that once A matches, B will not be tested further, even if it would produce a longer overall match. In other words, the "|" operator is never greedy. To match a literal "|", use **\|**, or enclose it inside a character class, as in **[|]**.
- (...) Matches whatever regular expression is inside the parentheses, and indicates the start and end of a group; the contents of a group can be matched later in the string with the **\number** special sequence, described below. To match the literals "(" or ")", use **\(** (or **\)**), or enclose them inside a character class: **[(]** or **[)]**.
- \number** Matches the contents of the group of the same number. Groups are numbered starting from 1. For example, **(.+)** **1** matches 'the the' or '55 55', but not 'the end' (note the space after the group). This special sequence can only be used to match one of the first 99 groups. If the first digit of number is 0, or number is 3 octal digits long, it will not be interpreted as a group match, but as the character with octal value number. Inside the "[" and "]" of a character class, all numeric escapes are treated as characters.

See <http://docs.python.org/release/2.5/lib/re-syntax.html> for descriptions of more advanced regular expression components, or see the following for additional references:

- http://en.wikipedia.org/wiki/Regular_expression
- <http://www.regular-expressions.info/reference.html>

8.2.2 Numeric Search

Beyond simple text or regular expression searching, Peppy can search through text files for numbers and perform matches based on comparisons with a reference number. Selecting the **Numeric Search** item from the **Search Type** pulldown menu activates this search type.

The screenshot shows the Peppy search interface with the following fields and controls:

- Search for:** A text input field containing the value `>1000`.
- Search type:** A dropdown menu currently showing "Numeric Se" (partially visible) and a checkbox for "Hexadecimal" which is unchecked.
- Search in:** A dropdown menu showing "Directory" and a text input field containing the path `/home/rob/src/peppy/tests`. To the right of this field is a "Browse" button.
- Ignore names:** A text input field containing the pattern `.git;.svn;.bzz;*.o;*.a;*.so;*.dll;*.~;*.bak;*.exe;*.pyc`. To the right of this field is a "Start" button.

The **Search for:** field then becomes the place to specify the reference number and the comparison operator. For example, specifying `>1000` will result in matches for all lines that have a number in them where the number is greater than 1000. Operators available are `<`, `<=`, `=`, `>=`, `>` and all whitespace is ignored.

By default the search is performed using decimal numbers (both floating point and integer values are found), but the **Hexadecimal** option may be selected by selecting the checkbox. Hex search finds all hex values in the text that are formatted with a leading **0x** or trailing **h** (like `0x2468`, `0xdeadbeef`, `fdb97531h`, `2badh`, etc.) and compares them to the reference value. The same comparison operators are available when using hexadecimal mode.

8.2.3 Expandability

The types of searches can be expanded through Peppy plugins, so more types may be available.

WORK IN PROGRESS

The documentation for all major modes is not yet complete. If you'd like to help, an easy way would be to pick a major mode that you use regularly and write up a sample page using `reStructuredText` and submit a new ticket to the peppy bug tracker at:

<http://trac.flipturn.org/index.fcgi/newticket?component=documentation&type=enhancement>

I'd appreciate the help!

PYTHON MAJOR MODE

10.1 Introducing Python Mode

Python mode is a specialization of *Fundamental Mode* that provides features dedicated to editing files from the Python language.

10.1.1 Automatic Indentation

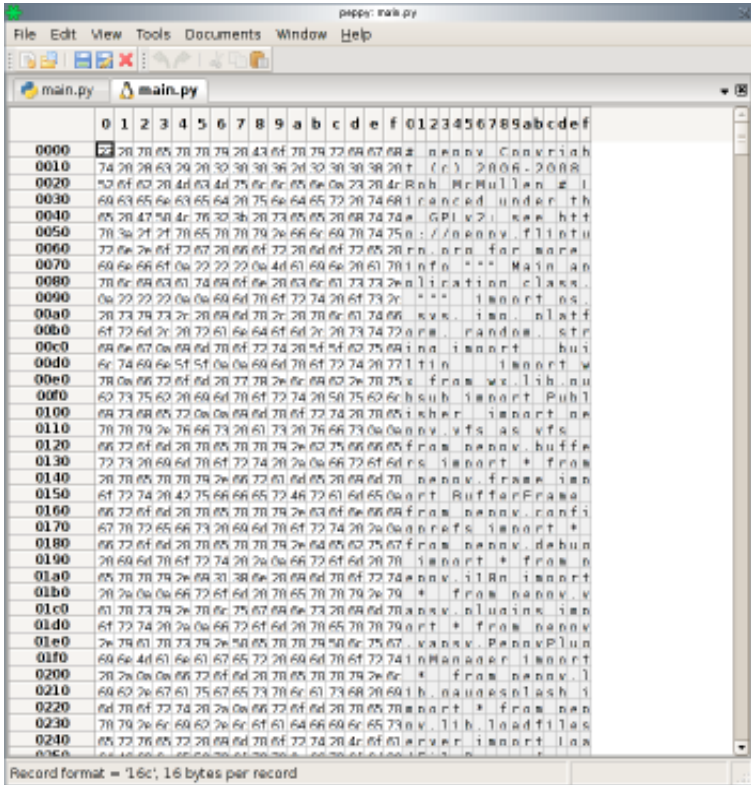
10.2 Running Python Scripts

HEXEDIT MAJOR MODE

HexEdit mode provides a means to edit binary data, that is: to edit individual bytes of a file. It doesn't matter what type of file it is, whether it's a text file or an image or an mp3 or a file full of IEEE floating point values – HexEdit can display the file.

Currently there are a small number of limitations. First is that the file size is limited: peppy can only edit files that can fit into memory. And second, if there are other views of the file (say it is a text file and one view is in Fundamental Mode and the other is HexEdit), the file must be viewed as raw bytes without an encoding. This is due to limitations in the styled text control that is used as the data storage mechanism.

11.1 The Editing Window



Peppy provides an editing window that shows the data fairly standard hex editor form: two sections, one showing the bytes as hexadecimal values, and the other section showing the same bytes as ASCII characters.

HYPERSENSPECTRAL IMAGE MAJOR MODE

12.1 What is Hyperspectral Imagery?

Commonly used in remote sensing applications, hyperspectral images have many more “colors” than just red, green, and blue. Typically, the images are structured in layers called “bands”, where each band represents a specific wavelength of light. You can think of them as a stack of grayscale images, one on top of the other, where each image in the stack is looking at the same field of view, but at a different wavelength of light.

Because each pixel at the same row and column in the band refers to the same spot on the ground, each pixel can be associated with a spectrum. Because there can be hundreds of bands in an image, the spectrum can be quite high resolution. Because of the large number of samples in the spectrum, the spectrum can be used to identify the material(s) in the pixel.

The Federation of American Scientists has a good tutorial about remote sensing, and it includes sections about hyperspectral imagery.

12.1.1 Hyperspectral Image Data

Hyperspectral images are very large, typically on the order of hundreds of megabytes for a standard scene. Most data comes from either government or commercial sensors mounted on either aircraft or spacecraft. NASA’s [AVIRIS sensor](#) is very well known in the field of earth remote sensing.

There are many different image formats for hyperspectral images, but they fall in two general categories: raw and compressed. `peppy` doesn’t support compressed formats directly; you must use GDAL to load compressed images like compressed NITF, ECW, jpeg2000, etc.

Raw Formats

A common raw image format is the ENVI format, which consists of two files: a data file in a raw format, and a header file that describes the type of data.

ENVI format is currently the only type of format supported in `peppy` without extra libraries.

Compressed Formats and GDAL

[GDAL](#) is a C++ library with a python binding that supports many different file formats through one API. Note that `peppy` only supports the `ngpython` bindings of GDAL, available by default starting with GDAL 1.5.

In addition, the libecw2 library from ERMapper can be used to support stand-alone JPEG2000 images and JPEG2000 images embedded in other formats like NITF.

12.1.2 Sample Images

ENVI Free data from [AVIRIS](#)

GDAL (with libecw2) jpeg2000: [MicroImages JPEG2000 image gallery](#)

ECW: [McElhanney sample images](#)

12.2 Hyperspectral Images in Peppy

The HSI mode in peppy provides simple capability to view hyperspectral images, navigate through the available bands, and show various profile and spectrum plots.

Because of the extremely large size of hyperspectral images, peppy is capable of viewing images much larger than can reside in physical memory. It uses the technique of memory mapping to accomplish this.

12.3 Starting HSI Mode

12.4 Changing View Parameters

12.5 Using Profile Plots